mod_perl 2.0 API

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

Last modified Sun Feb 16 01:36:08 2014 GMT

Part I: Apache2:: Core API

- 1. Apache2::Access - A Perl API for Apache request object: Access, Authentication and Authorization.
The API provided by this module deals with access, authentication and authorization phases.

- 2. Apache2::CmdParms - Perl API for Apache command parameters object
`Apache2::CmdParms` provides the Perl API for Apache command parameters object.

- 3. Apache2::Command - Perl API for accessing Apache module command information
`Apache2::Command` provides the Perl API for accessing Apache module command information

- 4. Apache2::Connection - Perl API for Apache connection object
`Apache2::RequestRec` provides the Perl API for Apache connection record object.

- 5. Apache2::ConnectionUtil - Perl API for Apache connection utils
`Apache2::ConnectionUtil` provides the *Apache connection record object* utilities API.

- 6. Apache2::Const - Perl Interface for Apache Constants
This package contains constants specific to `Apache` features.

- 7. Apache2::Directive - Perl API for manipulating the Apache configuration tree
`Apache2::Directive` provides the Perl API for manipulating the Apache configuration tree

- 8. Apache2::Filter - Perl API for Apache 2.0 Filtering
`Apache2::Filter` provides Perl API for Apache 2.0 filtering framework.

- 9. Apache2::FilterRec - Perl API for manipulating the Apache filter record
`Apache2::FilterRec` provides an access to the filter record structure.

- 10. Apache2::HookRun - Perl API for Invoking Apache HTTP phases
`Apache2::HookRun` exposes parts of the Apache HTTP protocol implementation, responsible for invoking callbacks for each *HTTP Request cycle phase*.

- 11. Apache2::Log - Perl API for Apache Logging Methods
`Apache2::Log` provides the Perl API for Apache logging methods.

- 12. Apache2::MPM - Perl API for accessing Apache MPM information
`Apache2::MPM` provides the Perl API for accessing Apache MPM information.

- 13. Apache2::Module - Perl API for creating and working with Apache modules
`Apache2::Module` provides the Perl API for creating and working with Apache modules

- 14. Apache2::PerlSections - write Apache configuration files in Perl
With `<Perl>...</Perl>` sections, it is possible to configure your server entirely in Perl.

- 15. Apache2::Process - Perl API for Apache process record
`Apache2::Process` provides the API for the Apache process object, which you can retrieve with *`$s->process`*:

- 16. Apache2::RequestIO - Perl API for Apache request record IO
  `Apache2::RequestIO` provides the API to perform IO on the *Apache request object*.

- 17. Apache2::RequestRec - Perl API for Apache request record accessors
  `Apache2::RequestRec` provides the Perl API for Apache request_rec object.

- 18. Apache2::RequestUtil - Perl API for Apache request record utils
  `Apache2::RequestUtil` provides the *Apache request object* utilities API.

- 19. Apache2::Response - Perl API for Apache HTTP request response methods
  `Apache2::Response` provides the *Apache request object* utilities API for dealing with HTTP response generation process.

- 20. Apache2::ServerRec - Perl API for Apache server record accessors
  `Apache2::ServerRec` provides the Perl API for Apache server_rec object.

- 21. Apache2::ServerUtil - Perl API for Apache server record utils
  `Apache2::ServerUtil` provides the *Apache server object* utilities API.

- 22. Apache2::SubProcess -- Executing SubProcesses under mod_perl
  `Apache2::SubProcess` provides the Perl API for running and communicating with processes spawned from mod_perl handlers.

- 23. Apache2::SubRequest - Perl API for Apache subrequests
  `Apache2::SubRequest` contains API for creating and running of Apache sub-requests.

- 24. Apache2::URI - Perl API for manipulating URIs
  While `APR::URI` provides a generic API to dissect, adjust and put together any given URI string, `Apache2::URI` provides an API specific to Apache, by taking the information directly from the `$r` object. Therefore when manipulating the URI of the current HTTP request usually methods from both classes are used.

- 25. Apache2::Util - Perl API for Misc Apache Utility functions
  Various Apache utilities that don't fit into any other group.

Part II: APR:: Core API

- 26. APR - Perl Interface for Apache Portable Runtime (libapr and libaprutil Libraries)
  On load this modules prepares the APR enviroment (initializes memory pools, data structures, etc.)

- 27. APR::Base64 - Perl API for APR base64 encoding/decoding functionality
  `APR::Base64` provides the access to APR's base64 encoding and decoding API.

- 28. APR::Brigade - Perl API for manipulating APR Bucket Brigades
  `APR::Brigade` allows you to create, manipulate and delete APR bucket brigades.

- 29. APR::Bucket - Perl API for manipulating APR Buckets
  `APR::Bucket` allows you to create, manipulate and delete APR buckets.

- 30. APR::BucketAlloc - Perl API for Bucket Allocation
`APR::BucketAlloc` is used for bucket allocation.

- 31. APR::BucketType - Perl API for APR bucket types
`APR::BucketType` allows you to query bucket object type properties.

- 32. APR::Const - Perl Interface for APR Constants
This package contains constants specific to `APR` features.

- 33. APR::Date - Perl API for APR date manipulating functions
`APR::Socket` provides the Perl interface to APR date manipulating functions.

- 34. APR::Error - Perl API for APR/Apache/mod_perl exceptions
`APR::Error` handles APR/Apache/mod_perl exceptions for you, while leaving you in control.

- 35. APR::Finfo - Perl API for APR fileinfo structure
APR fileinfo structure provides somewhat similar information to Perl's `stat()` call, but you will want to use this module's API to query an already `stat()'ed` filehandle to avoid an extra system call or to query attributes specific to APR file handles.

- 36. APR::IpSubnet - Perl API for accessing APRs ip_subnet structures
`APR::IpSubnet` object represents a range of IP addresses (IPv4/IPv6). A socket connection can be matched against this range to test whether the IP it's coming from is inside or outside of this range.

- 37. APR::OS - Perl API for Platform-specific APR API
`APR::OS` provides the Perl interface to platform-specific APR API.

- 38. APR::PerlIO -- Perl IO layer for APR
`APR::PerlIO` implements a Perl IO layer using APR's file manipulation API internally.

- 39. APR::Pool - Perl API for APR pools
`APR::Pool` provides an access to APR pools, which are used for an easy memory management.

- 40. APR::SockAddr - Perl API for APR socket address structure
`APR::SockAddr` provides an access to a socket address structure fields.

- 41. APR::Socket - Perl API for APR sockets
`APR::Socket` provides the Perl interface to APR sockets.

- 42. APR::Status - Perl Interface to the APR_STATUS_IS_* macros
An interface to *apr_errno.h* composite error codes.

- 43. APR::String - Perl API for manipulating APR UUIDs
`APR::String` provides strings manipulation API.

- 44. APR::Table - Perl API for manipulating APR opaque string-content tables
`APR::Table` allows its users to manipulate opaque string-content tables.

- 45. APR::ThreadMutex - Perl API for APR thread mutexes
  `APR::ThreadMutex` interfaces APR thread mutexes.

- 46. APR::ThreadRWLock - Perl API for APR thread read/write locks
  `APR::ThreadRWLock` interfaces APR thread read/write locks.

- 47. APR::URI - Perl API for URI manipulations
  `APR::URI` allows you to parse URI strings, manipulate each of the URI elements and deparse them back into URIs.

- 48. APR::Util - Perl API for Various APR Utilities
  Various APR utilities that don't fit into any other group.

- 49. APR::UUID - Perl API for manipulating APR UUIDs
  `APR::UUID` is used to get and manipulate random UUIDs.

Part III: ModPerl::

- 50. ModPerl::Const -- ModPerl Constants
  This package contains constants specific to mod_perl features.

- 51. ModPerl::Global -- Perl API for manipulating special Perl lists
  `ModPerl::Global` provides an API to manipulate special perl lists. At the moment only the `END` blocks list is supported.

- 52. ModPerl::MethodLookup -- Lookup mod_perl 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, to find out out what methods defined by some module, what methods can be called on a given object, etc.

- 53. 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`.

- 54. ModPerl::PerlRun - Run unaltered CGI scripts under mod_perl
  META: document that for now we don't chdir() into the script's dir, because it affects the whole process under threads. *`ModPerl::PerlRunPrefork`* should be used by those who run only under prefork MPM.

- 55. ModPerl::PerlRunPrefork - Run unaltered CGI scripts under mod_perl

- 56. 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.

- 57. 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.

- 58. 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`*.

- 59. 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.

- 60. ModPerl::RegistryPrefork - Run unaltered CGI scripts under mod_perl

- 61. ModPerl::Util - Helper mod_perl Functions
  `ModPerl::Util` provides mod_perl utilities API.

Part IV: Helper Modules / Applications

- 62. Apache2::compat -- 1.0 backward compatibility functions deprecated in 2.0
  `Apache2::compat` provides mod_perl 1.0 compatibility layer and can be used to smooth the transition process to mod_perl 2.0.

- 63. Apache2::porting -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting
  `Apache2::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 to trap methods calls 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.

- 64. Apache2::Reload - Reload Perl Modules when Changed on Disk
  `Apache2::Reload` reloads modules that change on the disk.

- 65. Apache2::Resource - Limit resources used by httpd children
  `Apache2::Resource` uses the `BSD::Resource` module, which uses the C function `setrlimit` to set limits on system resources such as memory and cpu usage.

- 66. Apache2::Status - Embedded interpreter status information
  The `Apache2::Status` module provides some information about the status of the Perl interpreter embedded in the server.

- 67. Apache2::SizeLimit - Because size does matter.
  This module is highly platform dependent, please read the Caveats section. It also does not work under threaded MPMs.

Part V: Internal Modules

- 68. 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.

Part VI: Related Modules

- 69. libapreq modules
  Apache::Request, Apache::Cookie, etc.

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

1 Apache2::Access - A Perl API for Apache request object: Access, Authentication and Authorization.

1.1 Synopsis

```
use Apache2::Access ();

# allow only GET method
$r->allow_methods(1, qw(GET));

# Apache Options value
$options = $r->allow_options();

# Apache AllowOverride value
$allow_override = $r->allow_overrides();

# which Options are allowed by AllowOverride (since Apache 2.2)
$allow_override_opts = $r->allow_override_opts();

# auth name ("foo bar")
$auth_name = $r->auth_name();

# auth type
$auth_type = $r->auth_type();
$r->auth_type("Digest");

# Basic authentication process
my ($rc, $passwd) = $r->get_basic_auth_pw();

# the login name of the remote user (RFC1413)
$remote_logname = $r->get_remote_logname();

# dynamically figure out which auth has failed
$r->note_auth_failure();

# note Basic auth failure
$r->note_basic_auth_failure();

# note Digest auth failure
$r->note_digest_auth_failure();

# Apache Request value(s)
$requires = $r->requires();

# Apache Satisfy value (as a number)

$satisfy = $r->satisfies();

# check whether some auth is configured
$need_auth = $r->some_auth_required();
```

1.2 Description

The API provided by this module deals with access, authentication and authorization phases.

Apache2::Access extends Apache2::RequestRec.

1.3 API

Apache2::Access provides the following functions and/or methods:

1.3.1 allow_methods

Specify which HTTP methods are allowed

```
$r->allow_methods($reset);
$r->allow_methods($reset, @methods);
```

- **obj: $r (Apache2::RequestRec object)**

  The current request

- **arg1: $reset (boolean)**

  If a true value is passed all the previously allowed methods are removed. Otherwise the list is left intact.

- **opt arg2: @methods (array of strings)**

  a list of HTTP methods to be allowed (e.g. GET and POST)

- **ret: no return value**
- **since: 2.0.00**

For example: here is how to allow only GET and POST methods, regardless to what was the previous setting:

```
$r->allow_methods(1, qw(GET POST));
```

1.3.2 allow_options

Retrieve the value of Options for this request

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

- **obj: $r (Apache2::RequestRec object)**

  The current request

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

  the Options bitmask. Normally used with bitlogic operators against Apache2::Const :options constants.

- **since: 2.0.00**

For example if the configuration for the current request was:

```
Options None
Options Indexes FollowSymLinks
```

The following applies:

```
use Apache2::Const -compile => qw(:options);
$r->allow_options & Apache2::Const::OPT_INDEXES;   # TRUE
$r->allow_options & Apache2::Const::OPT_SYM_LINKS; # TRUE
$r->allow_options & Apache2::Const::OPT_EXECCGI;   # FALSE
```

1.3.3 `allow_overrides`

Retrieve the value of `AllowOverride` for this request

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  the `AllowOverride` bitmask. Normally used with bitlogic operators against `Apache2::Const :override constants`.

- **since: 2.0.00**

For example if the configuration for the current request was:

```
AllowOverride AuthConfig
```

The following applies:

```
use Apache2::Const -compile => qw(:override);
$r->allow_overrides & Apache2::Const::OR_AUTHCFG; # TRUE
$r->allow_overrides & Apache2::Const::OR_LIMIT; # FALSE
```

1.3.4 `allow_override_opts`

Retrieve the bitmask of allowed `Options` set by `AllowOverride Options=...` for this request

```
$override_opts = $r->allow_override_opts();
```

Enabling single options was introduced in Apache 2.2. For Apache 2.0 this function returns `Apache2::Const::OPT_UNSET` | `Apache2::Const::OPT_ALL` | `Apache2::Const::OPT_INCNOEXEC` | `Apache2::Const::OPT_SYM_OWNER` | `Apache2::Const::OPT_MULTI`, which corresponds to the default value (if not set) for Apache 2.2.

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  the override options bitmask. Normally used with bitlogic operators against `Apache2::Const :options constants`.

- **since: 2.0.3**

For example if the configuration for the current request was:

```
AllowOverride Options=Indexes,ExecCGI
```

The following applies:

```
use Apache2::Const -compile => qw(:options);
$r->allow_override_opts & Apache2::Const::OPT_EXECCGI; # TRUE
$r->allow_override_opts & Apache2::Const::OPT_SYM_LINKS; # FALSE
```

1.3.5 `auth_name`

Get/set the current Authorization realm (the per directory configuration directive `AuthName`):

```
$auth_name = $r->auth_name();
$auth_name = $r->auth_name($new_auth_name);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **opt arg1: `$new_auth_name` (string)**

  If `$new_auth_name` is passed a new `AuthName` value is set

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

  The current value of `AuthName`

- **since: 2.0.00**

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

1.3.6 auth_type

Get/set the type of authorization required for this request (the per directory configuration directive AuthType):

```
$auth_type = $r->auth_type();
$auth_type = $r->auth_type($new_auth_type);
```

- **obj: $r (Apache2::RequestRec object)**

  The current request

- **opt arg1: $new_auth_type (string)**

  If $new_auth_type is passed a new AuthType value is set

- **ret: $ (integer)**

  The current value of AuthType

- **since: 2.0.00**

Normally AuthType would be set to Basic to use the basic authentication scheme defined in RFC 1945, *Hypertext Transfer Protocol -- HTTP/1.0*. However, you could set to something else and implement your own authentication scheme.

1.3.7 get_basic_auth_pw

Get the password from the request headers

```
my ($rc, $passwd) = $r->get_basic_auth_pw();
```

- **obj: $r (Apache2::RequestRec object)**

  The current request

- **ret1: $rc (Apache2::Const constant)**

  Apache2::Const::OK if the $passwd value is set (and assured a correct value in $r->user); otherwise it returns an error code, either Apache2::Const::HTTP_INTERNAL_SERVER_ERROR if things are really confused, Apache2::Const::HTTP_UNAUTHORIZED if no authentication at all seemed to be in use, or Apache2::Const::DECLINED if there was authentication, but it wasn't Basic (in which case, the caller should presumably decline as well).

- **ret2: $ret (string)**

  The password as set in the headers (decoded)

- **since: 2.0.00**

If `AuthType` is not set, this handler first sets it to `Basic`.

1.3.8 `get_remote_logname`

Retrieve the login name of the remote user (RFC1413)

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  The username of the user logged in to the client machine, or an empty string if it could not be determined via RFC1413, which involves querying the client's identd or auth daemon.

- **since: 2.0.00**

Do not confuse this method with `$r->user`, which provides the username provided by the user during the server authentication.

1.3.9 `note_auth_failure`

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();
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **ret: no return value**
- **since: 2.0.00**

This method requires `AuthType` to be set to `Basic` or `Digest`. Depending on the setting it'll call either `$r->note_basic_auth_failure` or `$r->note_digest_auth_failure`.

1.3.10 `note_basic_auth_failure`

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();
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **ret: no return value**
- **since: 2.0.00**

1.3.11 `note_digest_auth_failure`

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();
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **ret: no return value**
- **since: 2.0.00**

1.3.12 `requires`

Retrieve information about all of the requires directives for this request

```
$requires = $r->requires
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **ret: `$requires` (ARRAY ref)**

  Returns an array reference of hash references, containing information related to the `require` directive.

- **since: 2.0.00**

This is normally used for access control.

For example if the configuration had the following require directives:

```
Require user  goo bar
Require group bar tar
```

this method will return the following datastructure:

```
[
  {
    'method_mask' => -1,
    'requirement' => 'user goo bar'
  },
  {
    'method_mask' => -1,
    'requirement' => 'group bar tar'
  }
];
```

The *requirement* field is what was passed to the `Require` directive. The *method_mask* field is a bitmask which can be modified by the `Limit` directive, but normally it can be safely ignored as it's mostly used internally. For example if the configuration was:

```
Require user goo bar
Require group bar tar
<Limit POST>
   Require valid-user
</Limit>
```

and the request method was `POST`, `$r->requires` will return:

```
[
  {
    'method_mask' => -1,
    'requirement' => 'user goo bar'
  },
  {
    'method_mask' => -1,
    'requirement' => 'group bar tar'
  }
  {
    'method_mask' => 4,
    'requirement' => 'valid-user'
  }
];
```

But if the request method was `GET`, it will return only:

```
[
  {
    'method_mask' => -1,
    'requirement' => 'user goo bar'
  },
  {
    'method_mask' => -1,
    'requirement' => 'group bar tar'
  }
];
```

As you can see Apache gives you the requirements relevant for the current request, so the *method_mask* is irrelevant.

It is also a good time to remind that in the general case, access control directives should not be placed within a <Limit> section. Refer to the Apache documentation for more information.

Using the same configuration and assuming that the request was of type POST, the following code inside an Auth handler:

```
my %require =
    map { my ($k, $v) = split /\s+/, $_->{requirement}, 2; ($k, $v||'') }
    @{ $r->requires };
```

will populate `%require` with the following pairs:

```
'group' => 'bar tar',
'user' => 'goo bar',
'valid-user' => '',
```

1.3.13 `satisfies`

How the requires lines must be met. What's the applicable value of the `Satisfy` directive:

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  How the requirements must be met. One of the `Apache2::Const :satisfy constants`:

  `Apache2::Const::SATISFY_ANY`, `Apache2::Const::SATISFY_ALL` and `Apache2::Const::SATISFY_NOSPEC`.

- **since: 2.0.00**

See the documentation for the `Satisfy` directive in the Apache documentation.

1.3.14 `some_auth_required`

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

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  TRUE if authentication is required, FALSE otherwise

- **since: 2.0.00**

1.4 See Also

mod_perl 2.0 documentation.

1.5 Copyright

1.6 Authors

The mod_perl development team and numerous contributors.

2 Apache2::CmdParms - Perl API for Apache command parameters object

2.1 Synopsis

```
use Apache2::CmdParms ();
use Apache2::Module ();
use Apache2::Const -compile => qw(NOT_IN_LOCATION);

my @directives = (
  {
    name => 'MyDirective',
    cmd_data => 'some extra data',
  },
);

Apache2::Module::add(__PACKAGE__, \@directives);

sub MyDirective {
    my ($self, $parms, $args) = @_;

    # push config
    $parms->add_config(['ServerTokens off']);

    # this command's command object
    $cmd = $parms->cmd;

    # check the current command's context
    $error = $parms->check_cmd_context(Apache2::Const::NOT_IN_LOCATION);

    # this command's context
    $context = $parms->context;

    # this command's directive object
    $directive = $parms->directive;

    # the extra information passed thru cmd_data to
    # Apache2::Module::add()
    $info = $parms->info;

    # which methods are <Limit>ed ?
    $is_limited = $parms->method_is_limited('GET');

    # which allow-override bits are set
    $override = $parms->override;


    # which Options are allowed by AllowOverride (since Apache 2.2)
    $override = $parms->override_opts;

    # the path this command is being invoked in
    $path = $parms->path;

    # this command's pool
    $p = $parms->pool;

    # this command's configuration time pool
    $p = $parms->temp_pool;
}
```

2.2 Description

`Apache2::CmdParms` provides the Perl API for Apache command parameters object.

2.3 API

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

2.3.1 `add_config`

Dynamically add Apache configuration at request processing runtime:

```
$parms->add_config($lines);
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **arg1: `$lines` (ARRAY ref)**

  An ARRAY reference containing configuration lines per element, without the new line terminators.

- **ret: no return value**
- **since: 2.0.00**

See also: `$s->add_config`, `$r->add_config`

2.3.2 `check_cmd_context`

Check the current command against a context bitmask of forbidden contexts.

```
$error = $parms->check_cmd_context($check);
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **arg1: `$check` (`Apache2::Const :context constant`)**

  the context to check against.

- **ret: `$error` (string / undef)**

  If the context is forbidden, this method returns a textual description of why it was forbidden. If the context is permitted, this method returns `undef`.

- **since: 2.0.00**

For example here is how to check whether a command is allowed in the `<Location>` container:

```
use Apache2::Const -compile qw(NOT_IN_LOCATION);
if (my $error = $parms->check_cmd_context(Apache2::Const::NOT_IN_LOCATION)) {
    die "directive ... not allowed in <Location> context"
}
```

2.3.3 cmd

This module's command information

```
$cmd = $parms->cmd();
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **ret: `$cmd` (`Apache2::Command object`)**
- **since: 2.0.00**

2.3.4 directive

This command's directive object in the configuration tree

```
$directive = $parms->directive;
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **ret: `$directive` (`Apache2::Directive object`)**

  The current directive node in the configuration tree

- **since: 2.0.00**

2.3.5 info

The extra information passed through `cmd_data` in `Apache2::Module::add()`.

```
$info = $parms->info;
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **ret: `$info` (string)**

  The string passed in `cmd_data`

- **since: 2.0.00**

For example here is how to pass arbitrary information to a directive subroutine:

```
my @directives = (
  {
    name => 'MyDirective1',
    func => \&MyDirective,
    cmd_data => 'One',
  },
  {
    name => 'MyDirective2',
    func => \&MyDirective,
    cmd_data => 'Two',
  },
);
```

```
Apache2::Module::add(__PACKAGE__, \@directives);

sub MyDirective {
  my ($self, $parms, $args) = @_;
  my $info = $parms->info;
}
```

In this example `$info` will either be `'One'` or `'Two'` depending on whether the directive was called as *MyDirective1* or *MyDirective2*.

2.3.6 method_is_limited

Discover if a method is <Limit>ed in the current scope

```
$is_limited = $parms->method_is_limited($method);
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **arg1: `$method` (string)**

  The name of the method to check for

- **ret: `$is_limited` (boolean)**
- **since: 2.0.00**

For example, to check if the `GET` method is being `<Limit>`ed in the current scope, do:

```
if ($parms->method_is_limited('GET') {
    die "...";
}
```

2.3.7 override

Which allow-override bits are set (`AllowOverride` directive)

```
$override = $parms->override;
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **ret: `$override` (bitmask)**

  the allow-override bits bitmask, which can be tested against `Apache2::Const :override constants`.

- **since: 2.0.00**

For example to check that the `AllowOverride`'s `AuthConfig` and `FileInfo` options are enabled for this command, do:

```
use Apache2::Const -compile qw(:override);
$wanted = Apache2::Const::OR_AUTHCFG | Apache2::Const::OR_FILEINFO;
$masked = $parms->override & $wanted;
unless ($wanted == $masked) {
    die "...";
}
```

2.3.8 override_opts

Which options are allowed to be overridden by `.htaccess` files. This is set by `AllowOverride Options=...`.

```
$override_opts = $parms->override_opts;
```

Enabling single options was introduced with Apache 2.2. For Apache 2.0 this function simply returns a bitmask with all options allowed.

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **ret: `$override_opts` (bitmask)**

  the bitmask, which can be tested against `Apache2::Const :options constants`.

- **since: 2.0.3**

2.3.9 path

The current pathname/location/match of the block this command is in

```
$path = $parms->path;
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **ret: `$path` (string / `undef`)**

  If configuring for a block like <Location>, <LocationMatch>, <Directory>, etc., the pathname part of that directive. Otherwise, `undef` is returned.

- **since: 2.0.00**

For example for a container block:

```
<Location /foo>
...
</Location>
```

'/foo' will be returned.

2.3.10 pool

Pool associated with this command

```
$p = $parms->pool;
```

- **obj: $parms (Apache2::CmdParms object)**
- **ret: $p (APR::Pool object)**
- **since: 2.0.00**

2.3.11 server

The (vhost) server this command was defined in *httpd.conf*

```
$s = $parms->server;
```

- **obj: $parms (Apache2::CmdParms object)**
- **ret: $s (Apache2::Server object)**
- **since: 2.0.00**

2.3.12 temp_pool

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

```
$temp_pool = $parms->temp_pool;
```

- **obj: $parms (Apache2::CmdParms object)**
- **ret: $temp_pool (APR::Pool object)**
- **since: 2.0.00**

Most likely you shouldn't use this pool object, unless you know what you are doing. Use `$parms->pool` instead.

2.4 Unsupported API

`Apache2::CmdParms` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

2.4.1 context

Get context containing pointers to modules' per-dir config structures.

```
$context = $parms->context;
```

- **obj: `$parms` (`Apache2::CmdParms object`)**
- **ret: `$newval` (`Apache2::ConfVector object`)**

  Returns the commands' per-dir config structures

- **since: 2.0.00**

2.5 See Also

mod_perl 2.0 documentation.

2.6 Copyright

2.7 Authors

The mod_perl development team and numerous contributors.

3 Apache2::Command - Perl API for accessing Apache module command information

3.1 Synopsis

```
use Apache2::Module ();
use Apache2::Command ();
my $module = Apache2::Module::find_linked_module('mod_perl.c');

for (my $cmd = $module->cmds; $cmd; $cmd = $cmd->next) {
    $cmd->args_how();
    $cmd->errmsg();
    $cmd->name();
    $cmd->req_override();
}
```

3.2 Description

`Apache2::Command` provides the Perl API for accessing Apache module command information

3.3 API

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

3.3.1 `args_how`

What the command expects as arguments:

```
$how = $cmd->args_how();
```

- **obj: `$cmd` (`Apache2::Command object`)**
- **ret: `$how` (`Apache2::Const :cmd_how constant`)**

  The flag value representing the type of this command (i.e. `Apache2::Const::ITERATE`, `Apache2::Const::TAKE2`).

- **since: 2.0.00**

3.3.2 `errmsg`

Get *usage* message for that command, in case of syntax errors:

```
$error = $cmd->errmsg();
```

- **obj: `$cmd` (`Apache2::Command object`)**
- **ret: `$error` (string)**

  The error message

- **since: 2.0.00**

3.3.3 name

Get the name of this command:

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

- **obj: $cmd (Apache2::Command object)**
- **ret: $name (string)**

  The command name

- **since: 2.0.00**

3.3.4 next

Get the next command in the chain of commands for this module:

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

- **obj: $cmd (Apache2::Command object)**
- **ret: $next (Apache2::Command object)**

  Returns the next command in the chain for this module, `undef` for the last command.

- **since: 2.0.00**

3.3.5 req_override

What overrides need to be allowed to enable this command:

```
$override = $cmd->req_override
```

- **obj: $cmd (Apache2::Command object)**
- **ret: $override (Apache2::Const :override constant)**

  The bit mask representing the overrides this command is allowed in (i.e `Apache2::Const::OR_ALL`/`Apache2::Const::ACCESS_CONF`).

- **since: 2.0.00**

For example:

```
use Apache2::Const -compile => qw(:override);
$cmd->req_override() & Apache2::Const::OR_AUTHCFG;
$cmd->req_override() & Apache2::Const::OR_LIMIT;
```

3.4 See Also

mod_perl 2.0 documentation.

3.5 Copyright

3.6 Authors

The mod_perl development team and numerous contributors.

4 Apache2::Connection - Perl API for Apache connection object

4.1 Synopsis

```
use Apache2::Connection ();
use Apache2::RequestRec ();

my $c = $r->connection;

my $c = $r->connection;
# is connection still open?
$status = $c->aborted;

# base server
$base_server = $c->base_server();

# needed for creating buckets/brigades
$ba = $c->bucket_alloc();

# client's socket
$socket = $c->client_socket;

# unique connection id
$id = $c->id();

# connection filters stack
$input_filters = $c->input_filters();
$output_filters = $c->output_filters();

# keep the connection alive?
$status = $c->keepalive();

# how many requests served over the current connection
$served = $c->keepalives();

# this connection's local and remote socket addresses
$local_sa  = $c->local_addr();
$remote_sa = $c->remote_addr();

# local and remote hostnames
$local_host = $c->local_host();
$remote_host = $c->get_remote_host();
$remote_host = $c->remote_host();

# server and remote client's IP addresses
$local_ip = $c->local_ip();
$remote_ip = $c->remote_ip();

# connection level Apache notes
$notes = $c->notes();

# this connection's pool
$p = $c->pool();
```

4.2 Description

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

4.3 API

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

4.3.1 `aborted`

Check whether the connection is still open

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

- **obj: `$c` (`Apache2::Connection object`)**
- **ret: `$status` (boolean)**

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

- **since: 2.0.00**

4.3.2 `base_server`

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

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

- **obj: `$c` (`Apache2::Connection object`)**
- **ret: `$base_server` (`Apache2::Server object`)**
- **since: 2.0.00**

4.3.3 `bucket_alloc`

The bucket allocator to use for all bucket/brigade creations

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

- **obj: `$c` (`Apache2::Connection object`)**
- **ret: `$ba` (`APR::BucketAlloc object`)**
- **since: 2.0.00**

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

4.3.4 client_socket

Get/set the client socket

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

- **obj: $c (Apache2::Connection object)**
- **opt arg1: $new_socket (APR::Socket object object)**

  If passed a new socket will be set.

- **ret: $socket (APR::Socket object object)**

  current client socket

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

- **since: 2.0.00**

4.3.5 get_remote_host

Lookup the client's DNS hostname or IP address

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

- **obj: $c (Apache2::Connection object)**

  The current connection

- **opt arg1: $type (:remotehost constant)**

  The type of lookup to perform:

  - **Apache2::Const::REMOTE_DOUBLE_REV**

    will always force a DNS lookup, and also force a double reverse lookup, regardless of the HostnameLookups setting. The result is the (double reverse checked) hostname, or undef if any of the lookups fail.

  - **Apache2::Const::REMOTE_HOST**

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

  - **Apache2::Const::REMOTE_NAME**

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

- **`Apache2::Const::REMOTE_NOLOOKUP`**

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

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

- **opt arg2: `$dir_config` (`Apache2::ConfVector object`)**

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

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

- **ret: `$remote_host` (string/undef)**

  The remote hostname. If the configuration directive **HostNameLookups** is set to off, this returns the dotted decimal representation of the client's IP address instead. Might return `undef` if the hostname is not known.

- **since: 2.0.00**

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

4.3.6 `id`

ID of this connection; unique at any point in time

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

- **obj: `$c` (`Apache2::Connection object`)**
- **ret: `$id` (integer)**
- **since: 2.0.00**

4.3.7 `input_filters`

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

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

- **obj: `$c` (`Apache2::Connection object`)**
- **opt arg1: `$new_input_filters`**

Set a new value

- **ret: `$input_filters( Apache2::Filter object )`**

  The first filter in the connection input filters chain.

  If `$new_input_filters` was passed, returns the previous value.

- **since: 2.0.00**

For an example see: Bucket Brigades-based Protocol Module

4.3.8 `keepalive`

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

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

- **obj: `$c ( Apache2::Connection object )`**
- **opt arg1: `$new_status ( :conn_keepalive constant )`**

  Normally you should not mess with setting this option when handling the HTTP protocol. If you do (for example when sending your own headers set with `$r->assbackwards`) -- take a look at the ap_set_keepalive() function in *httpd-2.0/modules/http/http_protocol.c*.

- **ret: `$status ( :conn_keepalive constant )`**

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

- **since: 2.0.00**

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

For example:

```
use Apache2::RequestRec ();
use Apache2::Connection ();

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

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

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

4.3.9 keepalives

How many requests were already served over the current connection.

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

- **obj: $c (Apache2::Connection object)**
- **opt arg1: $new_served (integer)**

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

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

  How many requests were already served over the current connection.

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

- **since: 2.0.00**

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

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

4.3.10 local_addr

Get this connection's local socket address

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

- **obj: $c (Apache2::Connection object)**
- **ret: $local_sa (APR::SockAddr object)**
- **since: 2.0.00**

4.3.11 local_host

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

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

- **obj: $c (Apache2::Connection object)**
- **ret: $local_host (string)**
- **since: 2.0.00**

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

4.3.12 local_ip

server IP address

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

- **obj: $c (Apache2::Connection object)**
- **ret: $local_ip (string)**
- **since: 2.0.00**

4.3.13 notes

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

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

- **obj: $c (Apache2::Connection object)**
- **opt arg1: $new_notes (APR::Table object)**
- **ret: $notes (APR::Table object)**

  the current notes table.

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

- **since: 2.0.00**

Also see $r->notes

4.3.14 output_filters

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

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

- **obj: $c (Apache2::Connection object)**
- **opt arg1: $new_output_filters**

  Set a new value

- **ret: $output_filters (Apache2::Filter object)**

  The first filter in the connection output filters chain.

  If $new_output_filters was passed, returns the previous value.

- **since: 2.0.00**

For an example see: Bucket Brigades-based Protocol Module

4.3.15 pool

Pool associated with this connection

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

- **obj: $c (Apache2::Connection object)**
- **ret: $p (APR::Pool object)**
- **since: 2.0.00**

4.3.16 remote_addr

Get this connection’s remote socket address

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

- **obj: $c (Apache2::Connection object)**
- **ret: $remote_sa (APR::SockAddr object)**
- **since: 2.0.00**

4.3.17 remote_ip

Client’s IP address

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

- **obj: `$c` (`Apache2::Connection object`)**
- **opt arg1: `$new_remote_ip` (string)**

  If passed a new value will be set

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

  current remote ip address

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

- **since: 2.0.00**

4.3.18 `remote_host`

Client's DNS name:

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

- **obj: `$c` (`Apache2::Connection object`)**
- **ret: `$remote_host` (string/undef)**

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

- **since: 2.0.00**

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

4.4 Unsupported API

`Apache2::Connection` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

4.4.1 `conn_config`

Config vector containing pointers to connections per-server config structures

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

- **obj: `$c` (`Apache2::Connection object`)**
- **ret: `$ret` (`Apache2::ConfVector object`)**
- **since: 2.0.00**

4.4.2 sbh

META: Autogenerated - needs to be reviewed/completed

handle to scoreboard information for this connection

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

- **obj: `$c` (`Apache2::Connection object`)**
- **ret: `$sbh` (XXX)**
- **since: 2.0.00**

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

4.5 See Also

mod_perl 2.0 documentation.

4.6 Copyright

4.7 Authors

The mod_perl development team and numerous contributors.

5 Apache2::ConnectionUtil - Perl API for Apache connection utils

5.1 Synopsis

```
use Apache2::Connection     ();
use Apache2::ConnectionUtil ();
use Apache2::RequestRec     ();

# grab the connection object;
my $c = $r->connection;

# share perl objects like $r->pnotes
$old_val = $c->pnotes($key => $value);
```

5.2 Description

`Apache2::ConnectionUtil` provides the Apache connection record object utilities API.

5.3 API

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

5.3.1 `pnotes`

Share Perl variables between requests over the lifetime of the connection.

```
$old_val  = $c->pnotes($key => $val);
$val      = $c->pnotes($key);
$hash_ref = $c->pnotes();
```

- **obj: `$c` (`Apache2::Connection object`)**
- **opt arg1: `$key` (string)**

  A key value

- **opt arg2: `$val` (SCALAR)**

  Any scalar value (e.g. a reference to an array)

- **ret: (3 different possible values)**

  if both, `$key` and `$val` are passed the previous value for `$key` is returned if such existed, otherwise undef is returned.

  if only `$key` is passed, the current value for the given key is returned.

  if no arguments are passed, a hash reference is returned, which can then be directly accessed without going through the `pnotes()` interface.

- **since: 2.0.3**

See (`Apache2::RequestUtil::pnotes`) for the details of the `pnotes` method usage. The usage is identical except for a few differences. First is the use of `$c` instead of `$r` as the invocant. The second is that the the data persists for the lifetime of the connection instead of the lifetime of the request. If the connection is lost, so is the data stored in `pnotes`.

5.4 See Also

Apache2::Connection.

Apache2::RequestUtil::pnotes.

mod_perl 2.0 documentation.

5.5 Copyright

5.6 Authors

The mod_perl development team and numerous contributors.

6 Apache2::Const - Perl Interface for Apache Constants

6.1 Synopsis

```
# make the constants available but don't import them
use Apache2::Const -compile => qw(constant names ...);

# w/o the => syntax sugar
use Apache2::Const ("-compile", qw(constant names ...));

# compile and import the constants
use Apache2::Const qw(constant names ...);
```

6.2 Description

This package contains constants specific to `Apache` features.

mod_perl 2.0 comes with several hundreds of constants, which you don't want to make available to your Perl code by default, due to CPU and memory overhead. Therefore when you want to use a certain constant you need to explicitly ask to make it available.

For example, the code:

```
use Apache2::Const -compile => qw(FORBIDDEN OK);
```

makes the constants `Apache2::Const::FORBIDDEN` and `Apache2::Const::OK` available to your code, but they aren't imported. In which case you need to use a fully qualified constants, as in:

```
return Apache2::Const::OK;
```

If you drop the argument `-compile` and write:

```
use Apache2::Const qw(FORBIDDEN OK);
```

Then both constants are imported into your code's namespace and can be used standalone like so:

```
return OK;
```

Both, due to the extra memory requirement, when importing symbols, and since there are constants in other namespaces (e.g., `APR::` and `ModPerl::`, and non-mod_perl modules) which may contain the same names, it's not recommended to import constants. I.e. you want to use the `-compile` construct.

Finaly, in Perl `=>` is almost the same as the comma operator. It can be used as syntax sugar making it more clear when there is a key-value relation between two arguments, and also it automatically parses its lefthand argument (the key) as a string, so you don't need to quote it.

If you don't want to use that syntax, instead of writing:

```
use Apache2::Const -compile => qw(FORBIDDEN OK);
```

you could write:

```
use Apache2::Const "-compile", qw(FORBIDDEN OK);
```

and for parentheses-lovers:

```
use Apache2::Const ("-compile", qw(FORBIDDEN OK));
```

6.3 Constants

6.3.1 :cmd_how

```
use Apache2::Const -compile => qw(:cmd_how);
```

The `:cmd_how` constants group is used in `Apache2::Module::add()` and `$cmds->args_how`.

6.3.1.1 Apache2::Const::FLAG

One of *On* or *Off* (full description).

- **since: 2.0.00**

6.3.1.2 Apache2::Const::ITERATE

One argument, occuring multiple times (full description).

- **since: 2.0.00**

6.3.1.3 Apache2::Const::ITERATE2

Two arguments, the second occurs multiple times (full description).

- **since: 2.0.00**

6.3.1.4 Apache2::Const::NO_ARGS

No arguments at all (full description).

- **since: 2.0.00**

6.3.1.5 Apache2::Const::RAW_ARGS

The command will parse the command line itself (full description).

- **since: 2.0.00**

6.3.1.6 Apache2::Const::TAKE1

One argument only (full description).

- **since: 2.0.00**

6.3.1.7 Apache2::Const::TAKE12

One or two arguments (full description).

- **since: 2.0.00**

6.3.1.8 Apache2::Const::TAKE123

One, two or three arguments (full description).

- **since: 2.0.00**

6.3.1.9 Apache2::Const::TAKE13

One or three arguments (full description).

- **since: 2.0.00**

6.3.1.10 Apache2::Const::TAKE2

Two arguments (full description).

- **since: 2.0.00**

6.3.1.11 Apache2::Const::TAKE23

Two or three arguments (full description).

- **since: 2.0.00**

6.3.1.12 Apache2::Const::TAKE3

Three arguments (full description).

- **since: 2.0.00**

6.3.2 :common

```
use Apache2::Const -compile => qw(:common);
```

The `:common` group is for XXX constants.

6.3.2.1 Apache2::Const::AUTH_REQUIRED

- **since: 2.0.00**

6.3.2.2 Apache2::Const::DECLINED

- **since: 2.0.00**

6.3.2.3 Apache2::Const::DONE

- **since: 2.0.00**

6.3.2.4 Apache2::Const::FORBIDDEN

- **since: 2.0.00**

6.3.2.5 Apache2::Const::NOT_FOUND

- **since: 2.0.00**

6.3.2.6 Apache2::Const::OK

- **since: 2.0.00**

6.3.2.7 Apache2::Const::REDIRECT

- **since: 2.0.00**

6.3.2.8 Apache2::Const::SERVER_ERROR

- **since: 2.0.00**

6.3.3 :config

```
use Apache2::Const -compile => qw(:config);
```

The `:config` group is for XXX constants.

6.3.3.1 Apache2::Const::DECLINE_CMD

- **since: 2.0.00**

6.3.4 :conn_keepalive

```
use Apache2::Const -compile => qw(:conn_keepalive);
```

The `:conn_keepalive` constants group is used by the (`$c->keepalive`) method.

6.3.4.1 Apache2::Const::CONN_CLOSE

The connection will be closed at the end of the current HTTP request.

- **since: 2.0.00**

6.3.4.2 Apache2::Const::CONN_KEEPALIVE

The connection will be kept alive at the end of the current HTTP request.

- **since: 2.0.00**

6.3.4.3 Apache2::Const::CONN_UNKNOWN

The connection is at an unknown state, e.g., initialized but not open yet.

- **since: 2.0.00**

6.3.5 :context

```
use Apache2::Const -compile => qw(:context);
```

The `:context` group is used by the `$parms->check_cmd_context` method.

6.3.5.1 Apache2::Const::NOT_IN_VIRTUALHOST

The command is not in a <VirtualHost> block.

- **since: 2.0.00**

6.3.5.2 Apache2::Const::NOT_IN_LIMIT

The command is not in a <Limit> block.

- **since: 2.0.00**

6.3.5.3 Apache2::Const::NOT_IN_DIRECTORY

The command is not in a <Directory> block.

- **since: 2.0.00**

6.3.5.4 Apache2::Const::NOT_IN_LOCATION

The command is not in a <Location>/<LocationMatch> block.

- **since: 2.0.00**

6.3.5.5 Apache2::Const::NOT_IN_FILES

The command is not in a <Files>/<FilesMatch> block.

- **since: 2.0.00**

6.3.5.6 Apache2::Const::NOT_IN_DIR_LOC_FILE

The command is not in a <Files>/<FilesMatch>, <Location>/<LocationMatch> or <Directory> block.

- **since: 2.0.00**

6.3.5.7 Apache2::Const::GLOBAL_ONLY

The directive appears outside of any container directives.

- **since: 2.0.00**

6.3.6 :filter_type

```
use Apache2::Const -compile => qw(:filter_type);
```

The `:filter_type` group is for XXX constants.

6.3.6.1 Apache2::Const::FTYPE_CONNECTION

- **since: 2.0.00**

6.3.6.2 Apache2::Const::FTYPE_CONTENT_SET

- **since: 2.0.00**

6.3.6.3 Apache2::Const::FTYPE_NETWORK

- **since: 2.0.00**

6.3.6.4 Apache2::Const::FTYPE_PROTOCOL

- **since: 2.0.00**

6.3.6.5 Apache2::Const::FTYPE_RESOURCE

- **since: 2.0.00**

6.3.6.6 Apache2::Const::FTYPE_TRANSCODE

- **since: 2.0.00**

6.3.7 :http

```
use Apache2::Const -compile => qw(:http);
```

The `:http` group is for XXX constants.

6.3.7.1 Apache2::Const::HTTP_ACCEPTED

- **since: 2.0.00**

6.3.7.2 Apache2::Const::HTTP_BAD_GATEWAY

- **since: 2.0.00**

6.3.7.3 Apache2::Const::HTTP_BAD_REQUEST

- **since: 2.0.00**

6.3.7.4 Apache2::Const::HTTP_CONFLICT

- **since: 2.0.00**

6.3.7.5 Apache2::Const::HTTP_CONTINUE

- **since: 2.0.00**

6.3.7.6 Apache2::Const::HTTP_CREATED

- **since: 2.0.00**

6.3.7.7 Apache2::Const::HTTP_EXPECTATION_FAILED

- **since: 2.0.00**

6.3.7.8 Apache2::Const::HTTP_FAILED_DEPENDENCY

- **since: 2.0.00**

6.3.7.9 Apache2::Const::HTTP_FORBIDDEN

- **since: 2.0.00**

6.3.7.10 Apache2::Const::HTTP_GATEWAY_TIME_OUT

- **since: 2.0.00**

6.3.7.11 Apache2::Const::HTTP_GONE

- **since: 2.0.00**

6.3.7.12 Apache2::Const::HTTP_INSUFFICIENT_STORAGE

- **since: 2.0.00**

6.3.7.13 Apache2::Const::HTTP_INTERNAL_SERVER_ERROR

- **since: 2.0.00**

6.3.7.14 Apache2::Const::HTTP_LENGTH_REQUIRED

- **since: 2.0.00**

6.3.7.15 Apache2::Const::HTTP_LOCKED

- **since: 2.0.00**

6.3.7.16 Apache2::Const::HTTP_METHOD_NOT_ALLOWED

- **since: 2.0.00**

6.3.7.17 Apache2::Const::HTTP_MOVED_PERMANENTLY

- **since: 2.0.00**

6.3.7.18 Apache2::Const::HTTP_MOVED_TEMPORARILY

- **since: 2.0.00**

6.3.7.19 Apache2::Const::HTTP_MULTIPLE_CHOICES

- **since: 2.0.00**

6.3.7.20 Apache2::Const::HTTP_MULTI_STATUS

- **since: 2.0.00**

6.3.7.21 Apache2::Const::HTTP_NON_AUTHORITATIVE

- **since: 2.0.00**

6.3.7.22 Apache2::Const::HTTP_NOT_ACCEPTABLE

- **since: 2.0.00**

6.3.7.23 Apache2::Const::HTTP_NOT_EXTENDED

- **since: 2.0.00**

6.3.7.24 Apache2::Const::HTTP_NOT_FOUND

- **since: 2.0.00**

6.3.7.25 Apache2::Const::HTTP_NOT_IMPLEMENTED

- **since: 2.0.00**

6.3.7.26 Apache2::Const::HTTP_NOT_MODIFIED

- **since: 2.0.00**

6.3.7.27 Apache2::Const::HTTP_NO_CONTENT

- **since: 2.0.00**

6.3.7.28 Apache2::Const::HTTP_OK

- **since: 2.0.00**

6.3.7.29 Apache2::Const::HTTP_PARTIAL_CONTENT

- **since: 2.0.00**

6.3.7.30 Apache2::Const::HTTP_PAYMENT_REQUIRED

- **since: 2.0.00**

6.3.7.31 Apache2::Const::HTTP_PRECONDITION_FAILED

- **since: 2.0.00**

6.3.7.32 Apache2::Const::HTTP_PROCESSING

- **since: 2.0.00**

6.3.7.33 Apache2::Const::HTTP_PROXY_AUTHENTICATION_REQUIRED

- **since: 2.0.00**

6.3.7.34 Apache2::Const::HTTP_RANGE_NOT_SATISFIABLE

- **since: 2.0.00**

6.3.7.35 Apache2::Const::HTTP_REQUEST_ENTITY_TOO_LARGE

- **since: 2.0.00**

6.3.7.36 Apache2::Const::HTTP_REQUEST_TIME_OUT

- **since: 2.0.00**

6.3.7.37 Apache2::Const::HTTP_REQUEST_URI_TOO_LARGE

- **since: 2.0.00**

6.3.7.38 Apache2::Const::HTTP_RESET_CONTENT

- **since: 2.0.00**

6.3.7.39 Apache2::Const::HTTP_SEE_OTHER

- **since: 2.0.00**

6.3.7.40 Apache2::Const::HTTP_SERVICE_UNAVAILABLE

- **since: 2.0.00**

6.3.7.41 Apache2::Const::HTTP_SWITCHING_PROTOCOLS

- **since: 2.0.00**

6.3.7.42 Apache2::Const::HTTP_TEMPORARY_REDIRECT

- **since: 2.0.00**

6.3.7.43 Apache2::Const::HTTP_UNAUTHORIZED

- **since: 2.0.00**

6.3.7.44 Apache2::Const::HTTP_UNPROCESSABLE_ENTITY

- **since: 2.0.00**

6.3.7.45 Apache2::Const::HTTP_UNSUPPORTED_MEDIA_TYPE

- **since: 2.0.00**

6.3.7.46 Apache2::Const::HTTP_UPGRADE_REQUIRED

- **since: 2.0.00**

6.3.7.47 Apache2::Const::HTTP_USE_PROXY

- **since: 2.0.00**

6.3.7.48 Apache2::Const::HTTP_VARIANT_ALSO_VARIES

- **since: 2.0.00**

6.3.8 :input_mode

```
use Apache2::Const -compile => qw(:input_mode);
```

The `:input_mode` group is used by `get_brigade`.

6.3.8.1 Apache2::Const::MODE_EATCRLF

- **since: 2.0.00**

See `Apache2::Filter::get_brigade()`.

6.3.8.2 Apache2::Const::MODE_EXHAUSTIVE

- **since: 2.0.00**

See `Apache2::Filter::get_brigade()`.

6.3.8.3 Apache2::Const::MODE_GETLINE

- **since: 2.0.00**

See `Apache2::Filter::get_brigade()`.

6.3.8.4 Apache2::Const::MODE_INIT

- **since: 2.0.00**

See `Apache2::Filter::get_brigade()`.

6.3.8.5 Apache2::Const::MODE_READBYTES

- **since: 2.0.00**

See `Apache2::Filter::get_brigade()`.

6.3.8.6 Apache2::Const::MODE_SPECULATIVE

- **since: 2.0.00**

See `Apache2::Filter::get_brigade()`.

6.3.9 :log

```
use Apache2::Const -compile => qw(:log);
```

The `:log` group is for constants used by `Apache2::Log`.

6.3.9.1 Apache2::Const::LOG_ALERT

- **since: 2.0.00**

See `Apache2::Log`.

6.3.9.2 Apache2::Const::LOG_CRIT

- **since: 2.0.00**

See `Apache2::Log`.

6.3.9.3 Apache2::Const::LOG_DEBUG

- **since: 2.0.00**

See `Apache2::Log`.

6.3.9.4 Apache2::Const::LOG_EMERG

- **since: 2.0.00**

See Apache2::Log.

6.3.9.5 Apache2::Const::LOG_ERR

- **since: 2.0.00**

See Apache2::Log.

6.3.9.6 Apache2::Const::LOG_INFO

- **since: 2.0.00**

See Apache2::Log.

6.3.9.7 Apache2::Const::LOG_LEVELMASK

- **since: 2.0.00**

See Apache2::Log.

6.3.9.8 Apache2::Const::LOG_NOTICE

- **since: 2.0.00**

See Apache2::Log.

6.3.9.9 Apache2::Const::LOG_STARTUP

- **since: 2.0.00**

See Apache2::Log.

6.3.9.10 Apache2::Const::LOG_TOCLIENT

- **since: 2.0.00**

See Apache2::Log.

6.3.9.11 Apache2::Const::LOG_WARNING

- **since: 2.0.00**

See Apache2::Log.

6.3.10 :methods

```
use Apache2::Const -compile => qw(:methods);
```

The `:methods` constants group is used in conjunction with `$r->method_number`.

6.3.10.1 Apache2::Const::METHODS

- **since: 2.0.00**

6.3.10.2 Apache2::Const::M_BASELINE_CONTROL

- **since: 2.0.00**

6.3.10.3 Apache2::Const::M_CHECKIN

- **since: 2.0.00**

6.3.10.4 Apache2::Const::M_CHECKOUT

- **since: 2.0.00**

6.3.10.5 Apache2::Const::M_CONNECT

- **since: 2.0.00**

6.3.10.6 Apache2::Const::M_COPY

- **since: 2.0.00**

6.3.10.7 Apache2::Const::M_DELETE

- **since: 2.0.00**

6.3.10.8 Apache2::Const::M_GET

- **since: 2.0.00**

corresponds to the HTTP `GET` method

6.3.10.9 Apache2::Const::M_INVALID

- **since: 2.0.00**

6.3.10.10 Apache2::Const::M_LABEL

- **since: 2.0.00**

6.3.10.11 Apache2::Const::M_LOCK

- **since: 2.0.00**

6.3.10.12 Apache2::Const::M_MERGE

- **since: 2.0.00**

6.3.10.13 Apache2::Const::M_MKACTIVITY

- **since: 2.0.00**

6.3.10.14 Apache2::Const::M_MKCOL

- **since: 2.0.00**

6.3.10.15 Apache2::Const::M_MKWORKSPACE

- **since: 2.0.00**

6.3.10.16 Apache2::Const::M_MOVE

- **since: 2.0.00**

6.3.10.17 Apache2::Const::M_OPTIONS

- **since: 2.0.00**

6.3.10.18 Apache2::Const::M_PATCH

- **since: 2.0.00**

6.3.10.19 Apache2::Const::M_POST

- **since: 2.0.00**

corresponds to the HTTP `POST` method

6.3.10.20 Apache2::Const::M_PROPFIND

- **since: 2.0.00**

6.3.10.21 Apache2::Const::M_PROPPATCH

- **since: 2.0.00**

6.3.10.22 Apache2::Const::M_PUT

- **since: 2.0.00**

corresponds to the HTTP PUT method

6.3.10.23 Apache2::Const::M_REPORT

- **since: 2.0.00**

6.3.10.24 Apache2::Const::M_TRACE

- **since: 2.0.00**

6.3.10.25 Apache2::Const::M_UNCHECKOUT

- **since: 2.0.00**

6.3.10.26 Apache2::Const::M_UNLOCK

- **since: 2.0.00**

6.3.10.27 Apache2::Const::M_UPDATE

- **since: 2.0.00**

6.3.10.28 Apache2::Const::M_VERSION_CONTROL

- **since: 2.0.00**

6.3.11 :mpmq

```
use Apache2::Const -compile => qw(:mpmq);
```

The :mpmq group is for querying MPM properties.

6.3.11.1 Apache2::Const::MPMQ_NOT_SUPPORTED

- **since: 2.0.00**

6.3.11.2 Apache2::Const::MPMQ_STATIC

- **since: 2.0.00**

6.3.11.3 Apache2::Const::MPMQ_DYNAMIC

- **since: 2.0.00**

6.3.11.4 Apache2::Const::MPMQ_MAX_DAEMON_USED

- **since: 2.0.00**

6.3.11.5 Apache2::Const::MPMQ_IS_THREADED

- **since: 2.0.00**

6.3.11.6 Apache2::Const::MPMQ_IS_FORKED

- **since: 2.0.00**

6.3.11.7 Apache2::Const::MPMQ_HARD_LIMIT_DAEMONS

- **since: 2.0.00**

6.3.11.8 Apache2::Const::MPMQ_HARD_LIMIT_THREADS

- **since: 2.0.00**

6.3.11.9 Apache2::Const::MPMQ_MAX_THREADS

- **since: 2.0.00**

6.3.11.10 Apache2::Const::MPMQ_MIN_SPARE_DAEMONS

- **since: 2.0.00**

6.3.11.11 Apache2::Const::MPMQ_MIN_SPARE_THREADS

- **since: 2.0.00**

6.3.11.12 Apache2::Const::MPMQ_MAX_SPARE_DAEMONS

- **since: 2.0.00**

6.3.11.13 Apache2::Const::MPMQ_MAX_SPARE_THREADS

- **since: 2.0.00**

6.3.11.14 Apache2::Const::MPMQ_MAX_REQUESTS_DAEMON

- **since: 2.0.00**

6.3.11.15 Apache2::Const::MPMQ_MAX_DAEMONS

- **since: 2.0.00**

6.3.12 :options

```
use Apache2::Const -compile => qw(:options);
```

The `:options` group contains constants corresponding to the `Options` configuration directive. For examples see: `$r->allow_options`.

6.3.12.1 Apache2::Const::OPT_ALL

- **since: 2.0.00**

6.3.12.2 Apache2::Const::OPT_EXECCGI

- **since: 2.0.00**

6.3.12.3 Apache2::Const::OPT_INCLUDES

- **since: 2.0.00**

6.3.12.4 Apache2::Const::OPT_INCNOEXEC

- **since: 2.0.00**

6.3.12.5 Apache2::Const::OPT_INDEXES

- **since: 2.0.00**

6.3.12.6 Apache2::Const::OPT_MULTI

- **since: 2.0.00**

6.3.12.7 Apache2::Const::OPT_NONE

- **since: 2.0.00**

6.3.12.8 Apache2::Const::OPT_SYM_LINKS

- **since: 2.0.00**

6.3.12.9 Apache2::Const::OPT_SYM_OWNER

- **since: 2.0.00**

6.3.12.10 Apache2::Const::OPT_UNSET

- **since: 2.0.00**

6.3.13 :override

```
use Apache2::Const -compile => qw(:override);
```

The `:override` group contains constants corresponding to the `AllowOverride` configuration directive. For examples see: `$r->allow_options`.

6.3.13.1 Apache2::Const::ACCESS_CONF

**.conf* inside `<Directory>` or `<Location>`

- **since: 2.0.00**

6.3.13.2 Apache2::Const::EXEC_ON_READ

Force directive to execute a command which would modify the configuration (like including another file, or `IFModule`)

- **since: 2.0.00**

6.3.13.3 Apache2::Const::OR_ALL

`Apache2::Const::OR_LIMIT` | `Apache2::Const::OR_OPTIONS` | `Apache2::Const::OR_FILEINFO` | `Apache2::Const::OR_AUTHCFG` | `Apache2::Const::OR_INDEXES`

- **since: 2.0.00**

6.3.13.4 Apache2::Const::OR_AUTHCFG

*.*conf* inside `<Directory>` or `<Location>` and *.htaccess* when `AllowOverride AuthConfig`

- **since: 2.0.00**

6.3.13.5 Apache2::Const::OR_FILEINFO

*.*conf* anywhere and *.htaccess* when `AllowOverride FileInfo`

- **since: 2.0.00**

6.3.13.6 Apache2::Const::OR_INDEXES

*.*conf* anywhere and *.htaccess* when `AllowOverride Indexes`

- **since: 2.0.00**

6.3.13.7 Apache2::Const::OR_LIMIT

*.*conf* inside `<Directory>` or `<Location>` and *.htaccess* when `AllowOverride Limit`

- **since: 2.0.00**

6.3.13.8 Apache2::Const::OR_NONE

*.*conf* is not available anywhere in this override

- **since: 2.0.00**

6.3.13.9 Apache2::Const::OR_OPTIONS

*.*conf* anywhere and *.htaccess* when `AllowOverride Options`

- **since: 2.0.00**

6.3.13.10 Apache2::Const::OR_UNSET

Unset a directive (in `Allow`)

- **since: 2.0.00**

6.3.13.11 Apache2::Const::RSRC_CONF

*.*conf* outside `<Directory>` or `<Location>`

- **since: 2.0.00**

6.3.14 :platform

```
use Apache2::Const -compile => qw(:platform);
```

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

6.3.14.1 Apache2::Const::CRLF

- **since: 2.0.00**

6.3.14.2 Apache2::Const::CR

- **since: 2.0.00**

6.3.14.3 Apache2::Const::LF

- **since: 2.0.00**

6.3.15 :remotehost

```
use Apache2::Const -compile => qw(:remotehost);
```

The `:remotehost` constants group is is used by the `$c->get_remote_host` method.

6.3.15.1 Apache2::Const::REMOTE_DOUBLE_REV

- **since: 2.0.00**

6.3.15.2 Apache2::Const::REMOTE_HOST

- **since: 2.0.00**

6.3.15.3 Apache2::Const::REMOTE_NAME

- **since: 2.0.00**

6.3.15.4 Apache2::Const::REMOTE_NOLOOKUP

- **since: 2.0.00**

6.3.16 :satisfy

```
use Apache2::Const -compile => qw(:satisfy);
```

The `:satisfy` constants group is used in conjunction with `$r->satisfies`.

6.3.16.1 Apache2::Const::SATISFY_ALL

- **since: 2.0.00**

All of the requirements must be met.

6.3.16.2 Apache2::Const::SATISFY_ANY

- **since: 2.0.00**

any of the requirements must be met.

6.3.16.3 Apache2::Const::SATISFY_NOSPEC

- **since: 2.0.00**

There are no applicable satisfy lines

6.3.17 :types

```
use Apache2::Const -compile => qw(:types);
```

The `:types` group is for XXX constants.

6.3.17.1 Apache2::Const::DIR_MAGIC_TYPE

- **since: 2.0.00**

6.3.18 :proxy

```
use Apache2::Const -compile => qw(:proxy);
```

The `:proxy` constants group is used in conjunction with `$r->proxyreq`.

6.3.18.1 Apache2::Const::PROXYREQ_NONE

- **since: 2.0.2**

6.3.18.2 Apache2::Const::PROXYREQ_PROXY

- **since: 2.0.2**

6.3.18.3 Apache2::Const::PROXYREQ_REVERSE

- **since: 2.0.2**

6.3.18.4 `Apache2::Const::PROXYREQ_RESPONSE`

- **since: 2.0.5**

6.4 See Also

mod_perl 2.0 documentation.

HTTP Status Codes.

6.5 Copyright

6.6 Authors

The mod_perl development team and numerous contributors.

7 Apache2::Directive - Perl API for manipulating the Apache configuration tree

7.1 Synopsis

```
use Apache2::Directive ();

my $tree = Apache2::Directive::conftree();

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

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

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

my $node = $tree;
while ($node) {
    print $node->as_string;

    #do something with $node

    my $directive = $node->directive;
    my $args = $node->args;
    my $filename = $node->filename;
    my $line_num = $node->line_num;

    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;
        }
    }
}
```

7.2 Description

`Apache2::Directive` provides the Perl API for manipulating the Apache configuration tree

7.3 API

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

7.3.1 args

Get the arguments for the current directive:

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

- **obj: $node (Apache2::Directive object)**
- **ret: $args (string)**

  Arguments are separated by a whitespace in the string.

- **since: 2.0.00**

For example, in *httpd.conf*:

```
PerlSwitches -M/opt/lib -M/usr/local/lib -wT
```

And later:

```
my $tree = Apache2::Directive::conftree();
my $node = $tree->lookup('PerlSwitches');
my $args = $node->args;
```

$args now contains the string "-M/opt/lib -M/usr/local/lib -wT"

7.3.2 as_hash

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

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

- **obj: $conftree (Apache2::Directive object)**

  The config tree to stringify

- **ret: $config_hash (HASH reference)**
- **since: 2.0.00**

For example: in *httpd.conf*:

```
<Location /test>
  SetHandler perl-script
  PerlHandler Test::Module
</Location>
```

And later:

```
my $tree = Apache2::Directive::conftree();
my $node = $tree->lookup('Location', '/test/');
my $hash = $node->as_hash;
```

`$hash` now is:

```
{
  'SetHandler'  => 'perl-script',
  'PerlHandler' => 'Test::Module',
}
```

7.3.3 as_string

Get a string representation of the configuration node, in *httpd.conf* format.

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

- **obj: `$node` (`Apache2::Directive object`)**

  The config tree to stringify

- **ret: `$string` (string)**
- **since: 2.0.00**

For example: in *httpd.conf*:

```
<Location /test>
  SetHandler perl-script
  PerlHandler Test::Module
</Location>
```

And later:

```
my $tree = Apache2::Directive::conftree();
my $node = $tree->lookup('Location', '/test/');
my $string = $node->as_string;
```

`$string` is now:

```
SetHandler perl-script
PerlHandler Test::Module
```

7.3.4 conftree

Get the root of the configuration tree:

```
$conftree = Apache2::Directive::conftree();
```

- **obj: `Apache2::Directive` (class name)**
- **ret: `$conftree` (`Apache2::Directive object`)**
- **since: 2.0.00**

7.3.5 directive

Get the name of the directive in $node:

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

- **obj: $node (Apache2::Directive object)**
- **ret: $name (string)**
- **since: 2.0.00**

7.3.6 filename

Get the *filename* the configuration node was created from:

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

- **obj: $node (Apache2::Directive object)**
- **ret: $filename (string)**
- **since: 2.0.00**

For example:

```
my $tree = Apache2::Directive::conftree();
my $node = $tree->lookup('VirtualHost', 'example.com');
my $filename = $node->filename;
```

$filename is now the full path to the *httpd.conf* that VirtualHost was defined in.

If the directive was added with add_config(), the filename will be the path to the *httpd.conf* that trigerred that Perl code.

7.3.7 first_child

Get the first child node of this directive:

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

- **obj: $node (Apache2::Directive object)**
- **ret: $child_node (Apache2::Directive object)**

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

- **since: 2.0.00**

7.3.8 line_num

Get the line number in a *filename* this node was created at:

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

- **obj: `$node` (`Apache2::Directive object`)**
- **arg1: `$lineno` (integer)**
- **since: 2.0.00**

7.3.9 lookup

Get the node(s) matching a certain value.

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

- **obj: `$conftree` (`Apache2::Directive object`)**

  The config tree to stringify

- **arg1: `$directive` (string)**

  The name of the directive to search for

- **opt arg2: `args` (string)**

  Optional args to the directive to filter for

- **ret: `$string` (string / ARRAY of HASH refs)**

  In LIST context, it returns all matching nodes.

  In SCALAR context, it returns only the first matching node.

  If called with only `$directive` value, this method returns all nodes from that directive. For example:

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

  returns all nodes for `Alias` directives.

  If called with an extra `$args` argument, it returns only nodes where both the directive and the args matched. For example:

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

- **since: 2.0.00**

7.3.10 next

Get the next directive node in the tree:

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

- **obj: $node (Apache2::Directive object)**
- **ret: $next_node (Apache2::Directive object)**

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

- **since: 2.0.00**

7.3.11 parent

Get the parent node of this directive:

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

- **obj: $node (Apache2::Directive object)**
- **ret: parent_node (Apache2::Directive object)**

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

- **since: 2.0.00**

7.4 See Also

mod_perl 2.0 documentation.

7.5 Copyright

7.6 Authors

The mod_perl development team and numerous contributors.

8 Apache2::Filter - Perl API for Apache 2.0 Filtering

8.1 Synopsis

```
use Apache2::Filter ();

# filter attributes
my $c = $f->c;
my $r = $f->r;
my $frec = $f->frec();
my $next_f = $f->next;

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

# bucket brigade filtering API
$rc = $f->next->get_brigade($bb, $mode, $block, $readbytes);
$rc = $f->next->pass_brigade($bb);
$rc = $f->fflush($bb);

# streaming filtering API
while ($filter->read(my $buffer, $wanted)) {
    # transform $buffer here
    $filter->print($buffer);
}
if ($f->seen_eos) {
    $filter->print("filter signature");
}

# filter manipulations
$r->add_input_filter(\&callback);
$c->add_input_filter(\&callback);
$r->add_output_filter(\&callback);
$c->add_output_filter(\&callback);
$f->remove;
```

8.2 Description

`Apache2::Filter` provides Perl API for Apache 2.0 filtering framework.

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

8.3 Common Filter API

The following methods can be called from any filter handler:

8.3.1 c

Get the current connection object from a connection or a request filter:

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

- **obj: `$f` (`Apache2::Filter object`)**
- **ret: `$c` (`Apache2::Connection object`)**
- **since: 2.0.00**

8.3.2 `ctx`

Get/set the filter context data.

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

- **obj: `$f` (`Apache2::Filter object`)**
- **opt arg2: `$ctx` (SCALAR)**

  next context

- **ret: `$ctx` (SCALAR)**

  current context

- **since: 2.0.00**

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 datastructure 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 primary purpose of storing context data across multiple filter invocations, this method is also useful when used 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);
}
```

8.3.3 frec

Get/set the `Apache2::FilterRec` (filter record) object.

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

- **obj: `$f` (`Apache2::Filter object`)**
- **ret: `$frec` (`Apache2::FilterRec object`)**
- **since: 2.0.00**

For example you can call `$frec->name` to get filter's name.

8.3.4 next

Return the `Apache2::Filter` object of the next filter in chain.

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

- **obj: `$f` (`Apache2::Filter object`)**

  The current filter object

- **ret: `$next_f` (`Apache2::Filter object`)**

  The next filter object in chain

- **since: 2.0.00**

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.

The next filter can be a mod_perl one or not, it's easy to tell which one is that by calling `$f->frec->name`.

8.3.5 r

Inside an HTTP request filter retrieve the current request object:

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

- **obj: $f (Apache2::Filter object)**
- **ret: $r (Apache2::RequestRec object)**
- **since: 2.0.00**

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

8.3.6 remove

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

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

- **obj: $f (Apache2::Filter object)**
- **ret: no return value**
- **since: 2.0.00**

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 `Apache2::Const::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 (which may be fixed in 2.1). 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.

Remember that if the connection is `$c->keepalive`) and the connection filter is removed, it won't be added until the connection is closed. Which may happen after many HTTP requests. You may want to keep the filter in place and pass the data through unmodified, by returning `Apache2::Const::DECLINED`. If you need to reset the whole or parts of the filter context between requests, use the technique based on `$c->keepalives` counting.

This method works for native Apache (non-mod_perl) filters too.

8.4 Bucket Brigade Filter API

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

8.4.1 `fflush`

Flush a bucket brigade down the filter stack.

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

- **obj: `$f` (`Apache2::Filter object`)**

  The current filter

- **arg1: `$bb` (`Apache2::Brigade object`)**

  The brigade to flush

- **ret: `$rc` (`APR::Const status constant`)**

  Refer to the `pass_brigade()` entry.

- **excpt: `APR::Error`**

  Exceptions are thrown only when this function is called in the VOID context. Refer to the `get_brigade()` entry for details.

- **since: 2.0.00**

`fflush` is a shortcut method. So instead of doing:

```
my $b = APR::Bucket::flush_create($f->c->bucket_alloc);
$bb->insert_tail($b);
$f->pass_brigade($bb);
```

one can just write:

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

8.4.2 `get_brigade`

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

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

- **obj: `$next_f` (`Apache2::Filter object`)**

  The next filter in the filter chain.

Inside filter handlers it's usually `$f->next`. Inside protocol handlers: `$c->input_filters`.

- **arg1: `$bb` (`APR::Brigade object`)**

  The original bucket brigade passed to `get_brigade()`, which must be empty.

  Inside input filter handlers it's usually the second argument to the filter handler.

  Otherwise it should be created:

  ```
  my $bb = APR::Brigade->new($c->pool, $c->bucket_alloc);
  ```

  On return it gets populated with the next bucket brigade. That brigade may contain nothing if there was no more data to read. The return status tells the outcome.

- **opt arg2: `$mode` (`Apache2::Const :input_mode constant`)**

  The filter mode in which the data should be read.

  If inside the filter handler, you should normally pass the same mode that was passed to the filter handler (the third argument).

  At the end of this section the available modes are presented.

  If the argument `$mode` is not passed, `Apache2::Const::MODE_READBYTES` is used as a default value.

- **opt arg3: `$block` (`APR::Const :read_type constant`)**

  You may ask the reading operation to be blocking: `APR::Const::BLOCK_READ`, or nonblocking: `APR::Const::NONBLOCK_READ`.

  If inside the filter handler, you should normally pass the same blocking mode argument that was passed to the filter handler (the forth argument).

  If the argument `$block` is not passed, `APR::Const::BLOCK_READ` is used as a default value.

- **opt arg4: `$readbytes` (integer)**

  How many bytes to read from the next filter.

  If inside the filter handler, you may want the same number of bytes, as the upstream filter, i.e. the argument that was passed to the filter handler (the fifth argument).

  If the argument `$block` is not passed, 8192 is used as a default value.

- **ret: `$rc` (`APR::Const status constant`)**

  On success, `APR::Const::SUCCESS` is returned and `$bb` is populated (see the `$bb` entry).

In case of a failure -- a failure code is returned, in which case normally it should be returned to the caller.

If the bottom-most filter doesn't read from the network, then `Apache2::NOBODY_READ` is returned (META: need to add this constant).

Inside protocol handlers the return code can also be `APR::Const::EOF`, which is success as well.

- **excpt: `APR::Error`**

  You don't have to ask for the return value. If this function is called in the VOID context, e.g.:

  ```
  $f->next->get_brigade($bb, $mode, $block, $readbytes);
  ```

  mod_perl will do the error checking on your behalf, and if the return code is not `APR::Const::SUCCESS`, an `APR::Error exception` will be thrown. The only time you want to do the error checking yourself, is when return codes besides `APR::Const::SUCCESS` are considered as successful and you want to manage them by yourself.

- **since: 2.0.00**

Available input filter modes (the optional second argument `$mode`) are:

- **`Apache2::Const::MODE_READBYTES`**

  The filter should return at most readbytes data

- **`Apache2::Const::MODE_GETLINE`**

  The filter should return at most one line of CRLF data. (If a potential line is too long or no CRLF is found, the filter may return partial data).

- **`Apache2::Const::MODE_EATCRLF`**

  The filter should implicitly eat any CRLF pairs that it sees.

- **`Apache2::Const::MODE_SPECULATIVE`**

  The filter read should be treated as speculative and any returned data should be stored for later retrieval in another mode.

- **`Apache2::Const::MODE_EXHAUSTIVE`**

  The filter read should be exhaustive and read until it can not read any more. Use this mode with extreme caution.

- **`Apache2::Const::MODE_INIT`**

  The filter should initialize the connection if needed, NNTP or FTP over SSL for example.

Either compile all these constants with:

```
use Apache2::Const -compile => qw(:input_mode);
```

But it's a bit more efficient to compile only those constants that you need.

Example:

Here is a fragment of a filter handler, that receives a bucket brigade from the upstream filter:

```
use Apache2::Filter ();
use APR::Const    -compile => qw(SUCCESS);
use Apache2::Const -compile => qw(OK);
sub filter {
    my ($f, $bb, $mode, $block, $readbytes) = @_;

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

    # ... process $bb

    return Apache2::Const::OK;
}
```

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

You can see that in case of a failure, the handler returns immediately with that failure code, which gets propagated to the downstream filter.

If you decide not check the return code, you can write it as:

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

    $f->next->get_brigade($bb, $mode, $block, $readbytes);

    # ... process $bb

    return Apache2::Const::OK;
}
```

and the error checking will be done on your behalf.

You will find many more examples in `the filter handlers|docs::2.0::user::handlers::filters` and `the protocol handlers|docs::2.0::user::handlers::protocols` tutorials.

8.4.3 `pass_brigade`

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

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

- **obj: `$next_f` (`Apache2::Filter object`)**

  The next filter in the filter chain.

  Inside output filter handlers it's usually `$f->next`. Inside protocol handlers: `$c->output_filters`.

- **arg1: `$bb` (`APR::Brigade object`)**

  The bucket brigade to pass.

  Inside output filter handlers it's usually the second argument to the filter handler (after potential manipulations).

- **ret: `$rc` (`APR::Const status constant`)**

  On success, `APR::Const::SUCCESS` is returned.

  In case of a failure -- a failure code is returned, in which case normally it should be returned to the caller.

  If the bottom-most filter doesn't write to the network, then `Apache2::NOBODY_WROTE` is returned (META: need to add this constant).

  Also refer to the `get_brigade()` entry to see how to avoid checking the errors explicitly.

- **excpt: `APR::Error`**

  Exceptions are thrown only when this function is called in the VOID context. Refer to the `get_brigade()` entry for details.

- **since: 2.0.00**

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

Example:

Here is a fragment of a filter handler, that passes a bucket brigade to the downstream filter (after some potential processing of the buckets in the bucket brigade):

```
use Apache2::Filter ();
use APR::Const    -compile => qw(SUCCESS);
use Apache2::Const -compile => qw(OK);
sub filter {
    my ($f, $bb) = @_;

    # ... process $bb

    my $rc = $f->next->pass_brigade($bb);
```

```
    return $rc unless $rc == APR::Const::SUCCESS;

    return Apache2::Const::OK;
}
```

8.5 Streaming Filter API

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

8.5.1 `print`

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

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

- **obj: `$f` (`Apache2::Filter object`)**
- **arg1: `$buffer` (string)**

  The data to send.

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

  How many characters were sent. There is no need to check, since all should go through and if something goes work an exception will be thrown.

- **excpt: `APR::Error`**
- **since: 2.0.00**

This method should be used only in streaming filters.

8.5.2 `read`

Read data from the filter

```
$read = $f->read($buffer, $wanted);
```

- **obj: `$f` (`Apache2::Filter object`)**
- **arg1: `$buffer` (SCALAR)**

  The buffer to fill. All previous data will be lost.

- **opt arg2: `$wanted` (integer)**

  How many bytes to attempt to read.

  If this optional argument is not specified -- the default 8192 will be used.

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

  How many bytes were actually read.

  `$buffer` gets populated with the string that is read. It will contain an empty string if there was nothing to read.

- **excpt: `APR::Error`**
- **since: 2.0.00**

Reads at most `$wanted` characters into `$buffer`. The returned value `$read` tells exactly how many were read, making it easy to use it in a while loop:

```
while ($filter->read(my $buffer, $wanted)) {
    # transform $buffer here
    $filter->print($buffer);
}
```

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.

8.5.3 `seen_eos`

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

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

- **obj: `$f` (`Apache2::Filter object`)**

  The filter to remove

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

  a true value if EOS has been seen, otherwise a false value

- **since: 2.0.00**

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, $wanted)) {
    # do something with $buffer
}
if ($f->seen_eos) {
    # do something
}
```

The technique in this example 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.

8.6 Other Filter-related API

Other methods which affect filters, but called on non-`Apache2::Filter` objects:

8.6.1 `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);
```

- **obj: `$c` (`Apache2::Connection object`) or `$r` (`Apache2::RequestRec object`)**
- **arg1: `&callback` (CODE ref)**
- **ret: no return value**
- **since: 2.0.00**

[META: It seems that you can't add a filter when another filter is called. I've tried to add an output connection filter from the input connection filter when it was called for the first time. It didn't have any affect for the first request (over keepalive connection). The only way I succeeded to do that is from that input connection filter's filter_init handler. In fact it does work if there is any filter additional filter of the same kind configured from httpd.conf or via filter_init. It looks like there is a bug in httpd, where it doesn't prepare the chain of 3rd party filter if none were inserted before the first filter was called.]

8.6.2 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);
```

- **obj: $c (Apache2::Connection object) or $r (Apache2::RequestRec object)**
- **arg1: &callback (CODE ref)**
- **ret: no return value**
- **since: 2.0.00**

8.7 Filter Handler Attributes

Packages using filter attributes have to subclass Apache2::Filter:

```
package MyApache2::FilterCool;
use base qw(Apache2::Filter);
```

Attributes are parsed during the code compilation, by the function MODIFY_CODE_ATTRIBUTES, inherited from the Apache2::Filter package.

8.7.1 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 MyApache2::FilterOutputReq;
sub handler : FilterRequestHandler { ... }
```

and add the configuration entry:

```
PerlOutputFilterHandler MyApache2::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.

8.7.2 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 MyApache2::FilterOutputCon;
sub handler : FilterConnectionHandler { ... }
```

and add the configuration entry:

```
PerlOutputFilterHandler MyApache2::FilterOutputCon
```

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

8.7.3 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 Apache2::Const::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.

8.7.4 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 MyApache2::FilterBar;
use base qw(Apache2::Filter);
sub init   : FilterInitHandler { ... }
sub filter : FilterRequestHandler FilterHasInitHandler(\&init) {
    my ($f, $bb) = @_;
    # ...
    return Apache2::Const::OK;
}
```

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

8.8 Configuration

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

8.8.1 PerlInputFilterHandler

See `PerlInputFilterHandler`.

8.8.2 PerlOutputFilterHandler

See `PerlOutputFilterHandler`.

8.8.3 PerlSetInputFilter

See `PerlSetInputFilter`.

8.8.4 PerlSetOutputFilter

See `PerlSetInputFilter`.

8.9 TIE Interface

`Apache2::Filter` also implements a tied interface, so you can work with the `$f` object as a hash reference.

The TIE interface is mostly unimplemented and might be implemented post 2.0 release.

8.9.1 TIEHANDLE

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

- **obj: `$stashsv` (SCALAR)**
- **arg1: `$sv` (SCALAR)**
- **ret: `$ret` (SCALAR)**
- **since: subject to change**

8.9.2 PRINT

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

- **obj: . . . (XXX)**
- **ret: `$ret` (integer)**
- **since: subject to change**

8.10 See Also

mod_perl 2.0 documentation.

8.11 Copyright

8.12 Authors

The mod_perl development team and numerous contributors.

9 Apache2::FilterRec - Perl API for manipulating the Apache filter record

9.1 Synopsis

```
use Apache2::Filter ();
use Apache2::FilterRec ();

my $frec = $filter->frec;
print "filter name is:", $frec->name;
```

9.2 Description

`Apache2::FilterRec` provides an access to the filter record structure.

The `Apache2::FilterRec` object is retrieved by calling `frec()`:

```
$frec = $filter->frec;
```

9.3 API

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

9.3.1 name

The registered name for this filter

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

- **obj: `$frec` (`Apache2::FilterRec object`)**
- **ret: `$name` (string)**
- **since: 2.0.00**

mod_perl filters have four names:

```
modperl_request_output
modperl_request_input
modperl_connection_output
modperl_connection_input
```

You can see the names of the non-mod_perl filters as well. By calling `$filter->next->frec->name` you can get the name of the next filter in the chain.

Example:

Let’s print the name of the current and the filter that follows it:

```
use Apache2::Filter ();
use Apache2::FilterRec ();
for my $frec ($filter->frec, $filter->next->frec) {
    print "Name: ", $frec->name;
}
```

9.4 See Also

mod_perl 2.0 documentation.

9.5 Copyright

9.6 Authors

The mod_perl development team and numerous contributors.

10 Apache2::HookRun - Perl API for Invoking Apache HTTP phases

10.1 Synopsis

```
# httpd.conf
PerlProcessConnectionHandler MyApache2::PseudoHTTP::handler

#file:MyApache2/PseudoHTTP.pm
#--------------------------
package MyApache2::PseudoHTTP;

use Apache2::HookRun ();
use Apache2::RequestUtil ();
use Apache2::RequestRec ();

use Apache2::Const -compile => qw(OK DECLINED DONE SERVER_ERROR);

# implement the HTTP protocol cycle in protocol handler
sub handler {
    my $c = shift;
    my $r = Apache2::RequestRec->new($c);

    # register any custom callbacks here, e.g.:
    # $r->push_handlers(PerlAccessHandler => \&my_access);

    $rc = $r->run_post_read_request();
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    $rc = $r->run_translate_name;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    $rc = $r->run_map_to_storage;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    # this must be run all a big havoc will happen in the following
    # phases
    $r->location_merge($path);

    $rc = $r->run_header_parser;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    my $args = $r->args || '';
    if ($args eq 'die') {
        $r->die(Apache2::Const::SERVER_ERROR);

        return Apache2::Const::DONE;
    }

    $rc = $r->run_access_checker;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    $rc = $r->run_auth_checker;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    $rc = $r->run_check_user_id;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    $rc = $r->run_type_checker;
```

```
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    $rc = $r->run_fixups;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    # $r->run_handler is called internally by $r->invoke_handler,
    # invoke_handler sets all kind of filters, and does a few other
    # things but it's possible to call $r->run_handler, bypassing
    # invoke_handler
    $rc = $r->invoke_handler;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    $rc = $r->run_log_transaction;
    return $rc unless $rc == Apache2::Const::OK or $rc == Apache2::Const::DECLINED;

    return Apache2::Const::OK;
}
```

10.2 Description

`Apache2::HookRun` exposes parts of the Apache HTTP protocol implementation, responsible for invoking callbacks for each HTTP Request cycle phase.

Armed with that API, you could run some of the http protocol framework parts when implementing your own protocols. For example see how HTTP AAA (access, auth and authz) hooks are called from a protocol handler, implementing a command server, which has nothing to do with HTTP. Also you can see in Synopsis how to re-implement Apache HTTP cycle in the protocol handler.

Using this API you could probably also change the normal Apache behavior (e.g. invoking some hooks earlier than normal, or later), but before doing that you will probably need to spend some time reading through the Apache C code. That's why some of the methods in this document, point you to the specific functions in the Apache source code. If you just try to use the methods from this module, without understanding them well, don't be surprised if you will get some nasty crashes, from which mod_perl can't protect you.

10.3 API

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

10.3.1 `die`

Kill the current request

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$type` (integer)**

  Why the request is dieing. Expects an Apache status constant.

- **ret: no return value**
- **since: 2.0.00**

This method doesn't really abort the request, it just handles the sending of the error response, logging the error and such. You want to take a look at the internals of `ap_die()` in *httpd-2.0/modules/http/http_request.c* for more details.

10.3.2 `invoke_handler`

Run the response phase.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  The status of the current phase run: `Apache2::Const::OK`, `Apache2::HTTP_...`

- **since: 2.0.00**

`invoke_handler()` allows modules to insert filters, sets a default handler if none is set, runs `run_handler()` and handles some errors.

For more details see `ap_invoke_handler()` in *httpd-2.0/server/config.c*.

10.3.3 `run_access_checker`

Run the resource access control phase.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  the current request

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

  The status of the current phase run: `Apache2::Const::OK`, `Apache2::Const::DECLINED`, `Apache2::HTTP_...`.

- **since: 2.0.00**

This phase 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.

10.3.4 run_auth_checker

Run the authentication phase.

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

- **obj: $r (Apache2::RequestRec object)**

  the current request

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

  The status of the current phase run: Apache2::Const::OK, Apache2::Const::DECLINED, Apache2::HTTP_....

- **since: 2.0.00**

This phase 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).

10.3.5 run_check_user_id

Run the authorization phase.

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

- **obj: $r (Apache2::RequestRec object)**

  The current request

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

  The status of the current phase run: Apache2::Const::OK, Apache2::Const::DECLINED, Apache2::HTTP_....

- **since: 2.0.00**

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.

10.3.6 run_fixups

Run the fixup phase.

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

- **obj: $r (Apache2::RequestRec object)**

  The current request

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

  The status of the current phase run: Apache2::Const::OK, Apache2::Const::DECLINED, Apache2::HTTP_....

- **since: 2.0.00**

This phase allows modules to perform module-specific fixing of HTTP header fields. This is invoked just before the response phase.

10.3.7 run_handler

Run the response phase.

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

- **obj: $r (Apache2::RequestRec object)**

  The request_rec

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

  The status of the current phase run: Apache2::Const::OK, Apache2::Const::DECLINED, Apache2::HTTP_....

- **since: 2.0.00**

run_handler() is called internally by invoke_handler(). Use run_handler() only if you want to bypass the extra functionality provided by invoke_handler().

10.3.8 run_header_parser

Run the header parser phase.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  `Apache2::Const::OK` or `Apache2::Const::DECLINED`.

- **since: 2.0.00**

10.3.9 run_log_transaction

Run the logging phase.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  The status of the current phase run: `Apache2::Const::OK`, `Apache2::Const::DECLINED`, `Apache2::HTTP_...`

- **since: 2.0.00**

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

10.3.10 run_map_to_storage

Run the map_to_storage phase.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

`Apache2::Const::DONE` (or `Apache2::HTTP_*`) if this contextless request was just fulfilled (such as `TRACE`), `Apache2::Const::OK` if this is not a file, and `Apache2::Const::DECLINED` if this is a file. The core map_to_storage (`Apache2::HOOK_RUN_LAST`) will `directory_walk()` and `file_walk()` the `$r->filename` (all internal C functions).

- **since: 2.0.00**

This phase allows 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.

10.3.11 run_post_read_request

Run the post_read_request phase.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  The status of the current phase run: `Apache2::Const::OK` or `Apache2::Const::DECLINED`.

- **since: 2.0.00**

This phase is 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

10.3.12 run_translate_name

Run the translate phase.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  The status of the current phase run: `Apache2::Const::OK`, `Apache2::Const::DECLINED`, `Apache2::HTTP_...`.

- **since: 2.0.00**

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

10.3.13 `run_type_checker`

Run the type_checker phase.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  the current request

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

  The status of the current phase run: `Apache2::Const::OK`, `Apache2::Const::DECLINED`, `Apache2::HTTP_...`.

- **since: 2.0.00**

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

10.4 See Also

mod_perl 2.0 documentation.

10.5 Copyright

10.6 Authors

The mod_perl development team and numerous contributors.

11 Apache2::Log - Perl API for Apache Logging Methods

11.1 Synopsis

```
# in startup.pl
#--------------
use Apache2::Log;

use Apache2::Const -compile => qw(OK :log);
use APR::Const    -compile => qw(:error SUCCESS);

my $s = Apache2::ServerUtil->server;

$s->log_error("server: log_error");
$s->log_serror(__FILE__, __LINE__, Apache2::Const::LOG_ERR,
               APR::Const::SUCCESS, "log_serror logging at err level");
$s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_DEBUG,
               APR::Const::ENOTIME, "debug print");
Apache2::ServerRec->log_error("routine warning");

Apache2::ServerRec::warn("routine warning");

# in a handler
#-------------
package Foo;

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

use Apache2::Log;

use Apache2::Const -compile => qw(OK :log);
use APR::Const    -compile => qw(:error SUCCESS);

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

    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(Apache2::Log::LOG_MARK, Apache2::Const::LOG_DEBUG,
                   APR::Const::ENOTIME, "in debug");

    $s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_INFO,
                   APR::Const::SUCCESS, "server info");

    $s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_ERR,
                   APR::Const::ENOTIME, "fatal error");

    $r->log_reason("fatal error");
```

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

    return Apache2::Const::OK;
}
1;

# in a registry script
# httpd.conf: PerlOptions +GlobalRequest
use Apache2::ServerRec qw(warn); # override warn locally
print "Content-type: text/plain\n\n";
warn "my warning";
```

11.2 Description

`Apache2::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_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_WARNING,
               APR::Const::ENOTIME, "warning!");
```

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

```
$r->log_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_INFO,
               APR::Const::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 `Apache2::Const::LOG_DEBUG` is used (because that's how Apache 2.0 logging mechanism works).

Note: On Win32 Apache attempts to lock all writes to a file whenever it's opened for append (which is the case with logging functions), as Unix has this behavior built-in, while Win32 does not. Therefore `Apache2::Log` functions could be slower than Perl's print()/warn().

11.3 Constants

Log level constants can be compiled all at once:

```
use Apache2::Const -compile => qw(:log);
```

or individually:

```
use Apache2::Const -compile => qw(LOG_DEBUG LOG_INFO);
```

11.3.1 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:

11.3.1.1 `Apache2::Const::LOG_EMERG`

11.3.1.2 `Apache2::Const::LOG_ALERT`

11.3.1.3 `Apache2::Const::LOG_CRIT`

11.3.1.4 `Apache2::Const::LOG_ERR`

11.3.1.5 `Apache2::Const::LOG_WARNING`

11.3.1.6 `Apache2::Const::LOG_NOTICE`

11.3.1.7 `Apache2::Const::LOG_INFO`

11.3.1.8 `Apache2::Const::LOG_DEBUG`

11.3.2 Other Constants

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

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

Here is the rest of the logging related constants:

11.3.2.1 `Apache2::Const::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;
```

11.3.2.2 `Apache2::Const::LOG_TOCLIENT`

used to give content handlers the option of including the error text in the `ErrorDocument` sent back to the client. When `Apache2::Const::LOG_TOCLIENT` is passed to `log_rerror()` 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 `Apache2::Const::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:

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(ENOTIME);
$r->log_rerror(Apache2::Log::LOG_MARK,
               Apache2::Const::LOG_ERR|Apache2::Const::LOG_TOCLIENT,
               APR::Const::ENOTIME,
               "request log_rerror");
```

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.

11.3.2.3 `Apache2::Const::LOG_STARTUP`

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

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(SUCCESS);
$s->log_serror(Apache2::Log::LOG_MARK,
               Apache2::Const::LOG_INFO,
               APR::Const::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 `Apache2::Const::LOG_STARTUP` is binary ORed as in:

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(SUCCESS);
$s->log_serror(Apache2::Log::LOG_MARK,
               Apache2::Const::LOG_INFO|Apache2::Const::LOG_STARTUP,
               APR::Const::SUCCESS,
               "This log message comes with no header");
```

then the logging will be:

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

11.4 Server Logging Methods

11.4.1 `$s->log`

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

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

- **obj: `$s` (`Apache2::ServerRec object`)**
- **ret: `$slog` (`Apache2::Log::Server` object)**

  `Apache2::Log::Server` object to be used with LogLevel methods.

- **since: 2.0.00**

11.4.2 `$s->log_error`

just logs the supplied message to *error_log*

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

- **obj: `$s` (`Apache2::ServerRec object`)**
- **arg1: `@message` (strings ARRAY)**

  what to log

- **ret: no return value**
- **since: 2.0.00**

For example:

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

11.4.3 `$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);
```

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

  The file in which this function is called

- **arg2: `$line` (number)**

  The line number on which this function is called

- **arg3: `$level` (`Apache2::LOG_* constant`)**

  The level of this error message

- **arg4: `$status` (`APR::Const status constant`)**

  The status code from the last command (similar to $! in perl), usually `APR::Const constant` or coming from an exception object.

- **arg5: `@message` (strings ARRAY)**

  The log message(s)

- **ret: no return value**
- **since: 2.0.00**

For example:

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(ENOTIME SUCCESS);
$s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_ERR,
               APR::Const::SUCCESS, "log_serror logging at err level");

$s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_DEBUG,
               APR::Const::ENOTIME, "debug print");
```

11.4.4 `$s->warn`

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

is the same as:

```
$s->log_serror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_WARNING,
               APR::Const::SUCCESS, @warnings)
```

- **obj: `$s` (`Apache2::ServerRec` object)**
- **arg1: `@warnings` (strings ARRAY)**

  array of warning strings

- **ret: no return value**
- **since: 2.0.00**

For example:

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

11.5 Request Logging Methods

11.5.1 `$r->log`

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

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

- **obj: `$r` (`Apache2::RequestRec` object)**
- **ret: `$rlog` (`Apache2::Log::Request` object)**

`Apache2::Log::Request` object to be used with LogLevel methods.

- **since: 2.0.00**

11.5.2 `$r->log_error`

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

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `@message` (strings ARRAY)**

  what to log

- **ret: no return value**
- **since: 2.0.00**

For example:

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

11.5.3 `$r->log_reason`

This function provides a convenient way to log errors in a preformatted way:

```
$r->log_reason($message);
$r->log_reason($message, $filename);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$message` (string)**

  the message to log

- **opt arg2: `$filename` (string)**

  where to report the error as coming from (e.g. `__FILE__`)

- **ret: no return value**
- **since: 2.0.00**

For example:

```
$r->log_reason("There is no enough data");
```

will generate a log entry similar to the following:

```
[Fri Sep 24 11:58:36 2004] [error] access to /someuri
failed for 127.0.0.1, reason: There is no enough data.
```

11.5.4 $r->log_rerror

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

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

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

- **since: 2.0.00**

For example:

```
use Apache2::Const -compile => qw(:log);
use APR::Const    -compile => qw(ENOTIME SUCCESS);
$r->log_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_ERR,
               APR::Const::SUCCESS, "log_rerror logging at err level");

$r->log_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_DEBUG,
               APR::Const::ENOTIME, "debug print");
```

11.5.5 $r->warn

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

is the same as:

```
$r->log_rerror(Apache2::Log::LOG_MARK, Apache2::Const::LOG_WARNING,
               APR::Const::SUCCESS, @warnings)
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `@warnings` (strings ARRAY)**

  array of warning strings

- **ret: no return value**
- **since: 2.0.00**

For example:

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

11.6 Other Logging Methods

11.6.1 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);
```

- **obj: `$slog` (server or request log handle)**
- **arg1: `@message` (strings ARRAY)**
- **ret: no return value**
- **since: 2.0.00**

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.

11.6.2 `alert`

See LogLevel Methods.

11.6.3 `crit`

See LogLevel Methods.

11.6.4 `debug`

See LogLevel Methods.

11.6.5 `emerg`

See LogLevel Methods.

11.6.6 error

See LogLevel Methods.

11.6.7 info

See LogLevel Methods.

11.6.8 notice

See LogLevel Methods.

Though Apache treats `notice()` calls as special. The message is always logged regardless the value of `ErrorLog`, unless the error log is set to use syslog. (For details see httpd-2.0/server/log.c.)

11.6.9 warn

See LogLevel Methods.

11.7 General Functions

11.7.1 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.

```
my ($file, $line) = Apache2::Log::LOG_MARK();
```

- **ret1: `$file` (string)**
- **ret2: `$line` (number)**
- **since: 2.0.00**

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`).

11.8 Virtual Hosts

Code running from within a virtual host needs to be able to log into its `ErrorLog` file, if different from the main log. Calling any of the logging methods on the `$r` and `$s` objects will do the logging correctly.

If the core `warn()` is called, it'll be always logged to the main log file. Here is how to make it log into the vhost *error_log* file. Let's say that we start with the following code:

```
warn "the code is smoking";
```

1. First, we need to use mod_perl's logging function, instead of `CORE::warn`

   Either replace `warn` with `Apache2::ServerRec::warn`:

   ```
   use Apache2::Log ();
   Apache2::ServerRec::warn("the code is smoking");
   ```

   or import it into your code:

   ```
   use Apache2::ServerRec qw(warn); # override warn locally
   warn "the code is smoking";
   ```

   or override `CORE::warn`:

   ```
   use Apache2::Log ();
   *CORE::GLOBAL::warn = \&Apache2::ServerRec::warn;
   warn "the code is smoking";
   ```

   Avoid using the latter suggestion, since it'll affect all the code running on the server, which may break things. Of course you can localize that as well:

   ```
   use Apache2::Log ();
   local *CORE::GLOBAL::warn = \&Apache2::ServerRec::warn;
   warn "the code is smoking";
   ```

   Chances are that you need to make the internal Perl warnings go into the vhost's *error_log* file as well. Here is how to do that:

   ```
   use Apache2::Log ();
   local $SIG{__WARN__} = \&Apache2::ServerRec::warn;
   eval q[my $x = "aaa" + 1;]; # this issues a warning
   ```

   Notice that it'll override any previous setting you may have had, disabling modules like `CGI::Carp` which also use `$SIG{__WARN__}`

2. Next we need to figure out how to get hold of the vhost's server object.

   Inside HTTP request handlers this is possible via `Apache2->request`. Which requires either `PerlOptions +GlobalRequest` setting or can be also done at runtime if `$r` is available:

   ```
   use Apache2::RequestUtil ();
   sub handler {
       my $r = shift;
       Apache2::RequestUtil->request($r);
       ...
   ```

   Outside HTTP handlers at the moment it is not possible, to get hold of the vhost's *error_log* file. This shouldn't be a problem for the code that runs only under mod_perl, since the always available `$s` object can invoke a plethora of methods supplied by `Apache2::Log`. This is only a problem for modules, which are supposed to run outside mod_perl as well.

META: To solve this we think to introduce 'PerlOptions +GlobalServer', a big brother for 'PerlOptions +GlobalRequest', which will be set in modperl_hook_pre_connection.

11.9 Unsupported API

`Apache2::Log` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

11.9.1 `log_pid`

META: what is this method good for? it just calls getpid and logs it. In any case it has nothing to do with the logging API. And it uses static variables, it probably shouldn't be in the Apache public API.

Log the current pid

```
Apache2::Log::log_pid($pool, $fname);
```

- **obj: `$p` (`APR::Pool object`)**

  The pool to use for logging

- **arg1: `$fname` (file path)**

  The name of the file to log to

- **ret: no return value**
- **since: subject to change**

11.10 See Also

mod_perl 2.0 documentation.

11.11 Copyright

11.12 Authors

The mod_perl development team and numerous contributors.

12 Apache2::MPM - Perl API for accessing Apache MPM information

12.1 Synopsis

```
use Apache2::MPM ();

# check whether Apache MPM is threaded
if (Apache2::MPM->is_threaded) { do_something() }

# which mpm is used
my $mpm = lc Apache2::MPM->show;

# query mpm properties
use Apache2::Const -compile => qw(:mpmq);
if (Apache2::MPM->query(Apache2::Const::MPMQ_STATIC)) { ... }
```

12.2 Description

`Apache2::MPM` provides the Perl API for accessing Apache MPM information.

12.3 API

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

12.3.1 query

Query various attributes of the MPM

```
my $query = Apache2::MPM->query($const);
```

- **obj: `$class` (`Apache2::MPM class`)**

  the class name

- **arg1: `$const` (`Apache2::Const :mpmq group constant`)**

  The MPM attribute to query.

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

  the result of the query

- **since: 2.0.00**

For example to test whether the mpm is static:

```
use Apache2::Const -compile => qw(MPMQ_STATIC);
if (Apache2::MPM->query(Apache2::Const::MPMQ_STATIC)) { ... }
```

12.3.2 `is_threaded`

Check whether the running Apache MPM is threaded.

```
my $is_threaded = Apache2::MPM->is_threaded;
```

- **obj: `$class` (`Apache2::MPM class`)**

  the class name

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

  threaded or not

- **since: 2.0.00**

Note that this functionality is just a shortcut for:

```
use Apache2::Const -compile => qw(MPMQ_IS_THREADED);
my $is_threaded = Apache2::MPM->query(Apache2::Const::MPMQ_IS_THREADED);
```

12.3.3 `show`

What mpm is used

```
my $mpm = Apache2::MPM->show();
```

- **obj: `$class` (`Apache2::MPM class`)**

  the class name

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

  the name of the MPM. e.g., "Prefork".

- **since: 2.0.00**

12.4 See Also

mod_perl 2.0 documentation.

12.5 Copyright

12.6 Authors

The mod_perl development team and numerous contributors.

13 Apache2::Module - Perl API for creating and working with Apache modules

13.1 Synopsis

```
use Apache2::Module ();

#Define a configuration directive
my @directives = (
    {
        name => 'MyDirective',
    }
);

Apache2::Module::add(__PACKAGE__, \@directives);

# iterate over the whole module list
for (my $modp = Apache2::Module::top_module(); $modp; $modp = $modp->next) {
    my $name                 = $modp->name;
    my $index                = $modp->module_index;
    my $ap_api_major_version  = $modp->ap_api_major_version;
    my $ap_api_minor_version  = $modp->ap_api_minor_version;
    my $commands             = $modp->cmds;
}

# find a specific module
my $module = Apache2::Module::find_linked_module('mod_ssl.c');

# remove a specific module
$module->remove_loaded_module();

# access module configuration from a directive
sub MyDirective {
    my ($self, $parms, $args) = @_;
    my  $srv_cfg = Apache2::Module::get_config($self, $parms->server);
    [...]
}

# test if an Apache module is loaded
if (Apache2::Module::loaded('mod_ssl.c')) {
    [...]
}

# test if a Perl module is loaded
if (Apache2::Module::loaded('Apache2::Status')) {

    [...]
}
```

13.2 Description

`Apache2::Module` provides the Perl API for creating and working with Apache modules

See Apache Server Configuration Customization in Perl.

13.3 API

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

13.3.1 `add`

Add a module's custom configuration directive to Apache.

```
Apache2::Module::add($package, $cmds);
```

- **arg1: `$package` (string)**

  the package of the module to add

- **arg2: `$cmds` (ARRAY of HASH refs)**

  the list of configuration directives to add

- **ret: no return value**
- **since: 2.0.00**

See also Apache Server Configuration Customization in Perl.

13.3.2 `ap_api_major_version`

Get the httpd API version this module was build against, **not** the module's version.

```
$major_version = $module->ap_api_major_version();
```

- **obj: `$module` (`Apache2::Module object`)**
- **ret: `$major_version` (integer)**
- **since: 2.0.00**

This methid is used to check that module is compatible with this version of the server before loading it.

13.3.3 `ap_api_minor_version`

Get the module API minor version.

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

- **obj: `$module` (`Apache2::Module object`)**
- **ret: `$minor_version` (integer)**
- **since: 2.0.00**

`ap_api_minor_version()` provides API feature milestones.

It's not checked during module init.

13.3.4 cmds

Get the `Apache2::Command` object, describing all of the directives this module defines.

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

- **obj: `$module` (`Apache2::Module object`)**
- **ret: `$command` (`Apache2::Command object`)**
- **since: 2.0.00**

13.3.5 get_config

Retrieve a module's configuration. Used by configuration directives.

```
$cfg = Apache2::Module::get_config($class, $server, $dir_config);
$cfg = Apache2::Module::get_config($class, $server);
$cfg =          $self->get_config($server, $dir_config);
$cfg =          $self->get_config($server);
```

- **obj: `$module` (`Apache2::Module object`)**
- **arg1: `$class` (string)**

  The Perl package this configuration is for

- **arg2: `$server` (`Apache2::ServerRec object`)**

  The current server, typically `$r->server` or `$parms->server`.

- **opt arg3: `$dir_config` (`Apache2::ConfVector object`)**

  By default, the configuration returned is the server level one. To retrieve the per directory configuration, use `$r->per_dir_config` as a last argument.

- **ret: `$cfg` (HASH reference)**

  A reference to the hash holding the module configuration data.

- **since: 2.0.00**

See also Apache Server Configuration Customization in Perl.

13.3.6 find_linked_module

Find a module based on the name of the module

```
$module = Apache2::Module::find_linked_module($name);
```

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

  The name of the module ending in `.c`

- **ret: `$module` (`Apache2::Module object`)**

  The module object if found, `undef` otherwise.

- **since: 2.0.00**

For example:

```
my $module = Apache2::Module::find_linked_module('mod_ssl.c');
```

13.3.7 loaded

Determine if a certain module is loaded

```
$loaded = Apache2::Module::loaded($module);
```

- **name: `$module` (string)**

  The name of the module to search for.

  If `$module` ends with `.c`, search all the modules, statically compiled and dynamically loaded.

  If `$module` ends with `.so`, search only the dynamically loaded modules.

  If `$module` doesn't contain a `.`, search the loaded Perl modules (checks `%INC`).

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

  Returns true if the module is loaded, false otherwise.

- **since: 2.0.00**

For example, to test if this server supports ssl:

```
if (Apache2::Module::loaded('mod_ssl.c')) {
    [...]
}
```

To test is this server dynamically loaded mod_perl:

```
if (Apache2::Module::loaded('mod_perl.so')) {
    [...]
}
```

To test if `Apache2::Status` is loaded:

```
if (Apache2::Module::loaded('Apache2::Status')) {
    [...]
}
```

13.3.8 module_index

Get the index to this modules structures in config vectors.

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

- **obj: `$module` (`Apache2::Module object`)**
- **ret: `$index` (integer)**
- **since: 2.0.00**

13.3.9 name

Get the name of the module's *.c* file

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

- **obj: `$module` (`Apache2::Module object`)**
- **ret: `$name` (string)**
- **since: 2.0.00**

For example a mod_perl module, will return: *mod_perl.c*.

13.3.10 next

Get the next module in the list, `undef` if this is the last module in the list.

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

- **obj: `$module` (`Apache2::Module object`)**
- **ret: `$next_module` (`Apache2::Module object`)**
- **since: 2.0.00**

13.3.11 `remove_loaded_module`

Remove a module from the list of loaded modules permanently.

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

- **obj: `$module` (`Apache2::Module object`)**
- **ret: no return value**
- **since: 2.0.00**

13.3.12 `top_module`

Returns the first module in the module list. Usefull to start a module iteration.

```
$module = Apache2::Module::top_module();
```

- **ret: `$module` (`Apache2::Module object`)**
- **since: 2.0.00**

13.4 See Also

mod_perl 2.0 documentation.

13.5 Copyright

13.6 Authors

The mod_perl development team and numerous contributors.

14 Apache2::PerlSections - write Apache configuration files in Perl

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{"/~dougm/"} = {
  AuthUserFile   => '/tmp/htpasswd',
  AuthType       => 'Basic',
  AuthName       => 'test',
  DirectoryIndex => [qw(index.html index.htm)],
  Limit          => {
      "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

```
Apache2->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 API

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

14.3.1 server

Get the current server's object for the <Perl> section

```
<Perl>
  $s = Apache2::PerlSections->server();
</Perl>
```

- **obj: `Apache2::PerlSections` (class name)**
- **ret: `$s` (`Apache2::ServerRec object`)**
- **since: 2.0.03**

14.4 @PerlConfig and $PerlConfig

This array and scalar can be used to introduce literal configuration into the apache configuration. For example:

```
push @PerlConfig, 'Alias /foo /bar';
```

Or: $PerlConfig .= "Alias /foo /bar\n";

See also `$r->add_config`

14.5 Configuration Variables

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

14.5.1 $Apache2::PerlSections::Save

Each `<Perl>` section is evaluated in its unique namespace, by default residing in a sub-namespace of `Apache2::ReadConfig::`, therefore any local variables will end up in that namespace. For example if a `<Perl>` section happened to be in file */tmp/httpd.conf* starting on line 20, the namespace: `Apache2::ReadConfig::tmp::httpd_conf::line_20` will be used. Now if it had:

```
<Perl>
  $foo     = 5;
  my $bar  = 6;
  $My::tar = 7;
</Perl>
```

The local global variable `$foo` becomes `$Apache2::ReadConfig::tmp::httpd_conf::line_20::foo`, the other variable remain where they are.

By default, the namespace in which `<Perl>` sections are evaluated is cleared after each block closes. In our example nuking `$Apache2::ReadConfig::tmp::httpd_conf::line_20::foo`, leaving the rest untouched.

By setting `$Apache2::PerlSections::Save` to a true value, the content of those namespaces will be preserved and will be available for inspection by `Apache2::Status` and `Apache2::PerlSections->dump` In our example `$Apache2::ReadConfig::tmp::httpd_conf::line_20::foo` will still be accessible from other perl code, after the `<Perl>` section was parsed.

14.6 PerlSections Dumping

14.6.1 Apache2::PerlSections->dump

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

```
my $dump = Apache2::PerlSections->dump;
```

- **ret: `$dump` (string / `undef`)**

  A string dump of all the Perl code encountered in <Perl> blocks, suitable to be read back via `eval`

For example:

```
<Perl>

$Apache2::PerlSections::Save = 1;

$Listen = 8529;

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

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

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

<Perl>
print Apache2::PerlSections->dump;
</Perl>
```

This will print something like this:

```
$Listen = 8529;

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

$Location{'/perl'} = (
```

```
    PerlHandler => 'Apache2::Registry',
    SetHandler => 'perl-script',
    Options => 'ExecCGI'
);

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

1;
__END__
```

It is important to put the call to `dump` in it's own `<Perl>` section, otherwise the content of the current `<Perl>` section will not be dumped.

14.6.2 `Apache2::PerlSections->store`

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

```
Apache2::PerlSections->store($filename);
```

- **arg1: `$filename` (string)**

  The filename to save the dump output to

- **ret: no return value**

14.7 Advanced API

mod_perl 2.0 now introduces the same general concept of handlers to `<Perl>` sections. Apache2::Perl-Sections 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 in encountered, the code within will first be evaluated, then the handler routine will be invoked with 3 arguments:

- **arg1: `$self`**

  self-explanatory

- **arg2: `$parms` (`Apache2::CmdParms`)**

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

- **arg3: `$args` (`APR::Table object`)**

  the table object of the section arguments. The 2 guaranteed ones will be:

  ```
  $args->{'handler'} = 'My::PerlSection::Handler';
  $args->{'package'} = 'Apache2::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 Apache2::Server->add_config("directive"), for example:

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

Would create a new alias. The source code of `Apache2::PerlSections` is a good place to look for a practical example.

14.8 Verifying `<Perl>` Sections

If the `<Perl>` sections include no code requiring a running mod_perl, it is possible to check those from the command line. But the following trick should be used:

```
# file: httpd.conf
<Perl>
#!perl

# ... code here ...

__END__
</Perl>
```

Now you can run:

```
% perl -c httpd.conf
```

14.9 Bugs

14.9.1 <Perl> directive missing closing '>'

httpd-2.0.47 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>
```

14.9.2 <Perl>[...]> was not closed.

On encountering a one-line <Perl> block, httpd's configuration parser will cause a startup failure with an error similar to this one:

```
Starting httpd:
Syntax error on line ... of /etc/httpd/conf/httpd.conf:
<Perl>use> was not closed.
```

If you have written a simple one-line <Perl> section like this one :

```
<Perl>use Apache::DBI;</Perl>
```

change it to be:

```
<Perl>
use Apache::DBI;
</Perl>
```

This is caused by a limitation of httpd's configuration parser and is not likely to be changed to allow one-line block like the example above. Use multi-line blocks instead.

14.10 See Also

mod_perl 2.0 documentation.

14.11 Copyright

14.12 Authors

The mod_perl development team and numerous contributors.

15 Apache2::Process - Perl API for Apache process record

15.1 Synopsis

```
use Apache2::Process ();
use Apache2::ServerRec ();
my $proc = $s->process;

# global pool cleared on exit
my $global_pool = $proc->pool;

# configuration pool cleared on restart
my $pconf = $proc->pconf;

# short program name (e.g. httpd)
my $proc_name = $proc->short_name;
```

15.2 Description

`Apache2::Process` provides the API for the Apache process object, which you can retrieve with `$s->process`:

```
use Apache2::ServerRec ();
$proc = $s->process;
```

15.3 API

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

15.3.1 pconf

Get configuration pool object.

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

- **obj: `$proc` (`Apache2::Process object`)**
- **ret: `$p` (`APR::Pool object`)**
- **since: 2.0.00**

This pool object gets cleared on server restart.

15.3.2 pool

Get the global pool object.

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

- **obj: `$proc` (`Apache2::Process object`)**
- **ret: `$p` (`APR::Pool object`)**
- **since: 2.0.00**

This pool object gets cleared only on (normal) server exit

15.3.3 `short_name`

The name of the program used to execute the program

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

- **obj: `$proc` (`Apache2::Process object`)**
- **ret: `$short_name` (string)**

  e.g. `httpd`

- **since: 2.0.00**

15.4 See Also

mod_perl 2.0 documentation.

15.5 Copyright

15.6 Authors

The mod_perl development team and numerous contributors.

16 Apache2::RequestIO - Perl API for Apache request record IO

16.1 Synopsis

```
use Apache2::RequestIO ();

$rc = $r->discard_request_body();

$r->print("foo", "bar");
$r->puts("foo", "bar"); # same as print, but no flushing
$r->printf("%s $d", "foo", 5);

$r->read($buffer, $len);

$r->rflush();

$r->sendfile($filename);

$r->write("foobartarcar", 3, 5);
```

16.2 Description

`Apache2::RequestIO` provides the API to perform IO on the Apache request object.

16.3 API

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

16.3.1 `discard_request_body`

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.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  `APR::Const status constant` if request is malformed, `Apache2::Const::OK` otherwise.

- **since: 2.0.00**

Since we return an error status if the request is malformed, this routine should be called at the beginning of a no-body handler, e.g.,

```
use Apache2::Const -compile => qw(OK);
$rc = $r->discard_request_body;
return $rc if $rc != Apache2::Const::OK;
```

16.3.2 print

Send data to the client.

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

- **obj: $r (Apache2::RequestRec object)**
- **arg1: @msg (ARRAY)**

  Data to send

- **ret: $cnt (number)**

  How many bytes were sent (or buffered). If zero bytes were sent, `print` will return `0E0`, or "zero but true," which will still evaluate to `0` in a numerical context.

- **excpt: APR::Error**
- **since: 2.0.00**

The data is flushed only if STDOUT stream's `$|` is true. Otherwise it's buffered up to the size of the buffer, flushing only excessive data.

16.3.3 printf

Format and send data to the client (same as `printf`).

```
$cnt = $r->printf($format, @args);
```

- **obj: $r (Apache2::RequestRec object)**
- **arg1: $format (string)**

  Format string, as in the Perl core `printf` function.

- **arg2: @args (ARRAY)**

  Arguments to be formatted, as in the Perl core `printf` function.

- **ret: $cnt (number)**

  How many bytes were sent (or buffered)

- **excpt: APR::Error**
- **since: 2.0.00**

The data is flushed only if STDOUT stream's `$|` is true. Otherwise it's buffered up to the size of the buffer, flushing only excessive data.

16.3.4 puts

Send data to the client

```
$cnt = $r->puts(@msg);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `@msg` (ARRAY)**

  Data to send

- **ret: `$cnt` (number)**

  How many bytes were sent (or buffered)

- **excpt: `APR::Error`**
- **since: 2.0.00**

`puts()` is similar to `print()`, but it won't attempt to flush data, no matter what the value of STDOUT stream's `$|` is. Therefore assuming that STDOUT stream's `$|` is true, this method should be a tiny bit faster than `print()`, especially if small strings are printed.

16.3.5 read

Read data from the client.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$buffer` (SCALAR)**

  The buffer to populate with the read data

- **arg2: `$len` (number)**

  How many bytes to attempt to read

- **opt arg3: `$offset` (number)**

  If a non-zero `$offset` is specified, the read data will be placed at that offset in the `$buffer`.

  META: negative offset and \0 padding are not supported at the moment

- **ret: `$cnt` (number)**

How many characters were actually read

- **excpt: `APR::Error`**
- **since: 2.0.00**

This method shares a lot of similarities with the Perl core `read()` function. The main difference in the error handling, which is done via `APR::Error exceptions`

16.3.6 `rflush`

Flush any buffered data to the client.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: no return value**
- **since: 2.0.00**

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

16.3.7 `sendfile`

Send a file or a part of it

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$filename` (string)**

  The full path to the file (using `/` on all systems)

- **opt arg2: `$offset` (integer)**

  Offset into the file to start sending.

  No offset is used if `$offset` is not specified.

- **opt arg3: `$len` (integer)**

  How many bytes to send.

  If not specified the whole file is sent (or a part of it, if `$offset` if specified)

- **ret: `$rc` (`APR::Const status constant`)**

On success, APR::Const::SUCCESS is returned.

In case of a failure -- a failure code is returned, in which case normally it should be returned to the caller.

- **excpt: APR::Error**

  Exceptions are thrown only when this function is called in the VOID context. So if you don't want to handle the errors, just don't ask for a return value and the function will handle all the errors on its own.

- **since: 2.0.00**

16.3.8 write

Send partial string to the client

```
$cnt = $r->write($buffer);
$cnt = $r->write($buffer, $len);
$cnt = $r->write($buffer, $len, $offset);
```

- **obj: $r (Apache2::RequestRec object)**
- **arg1: $buffer (SCALAR)**

  The string with data

- **opt arg2: $len (SCALAR)**

  How many bytes to send. If not specified, or -1 is specified, all the data in $buffer (or starting from $offset) will be sent.

- **opt arg3: $offset (number)**

  Offset into the $buffer string.

- **ret: $cnt (number)**

  How many bytes were sent (or buffered)

- **excpt: APR::Error**
- **since: 2.0.00**

Examples:

Assuming that we have a string:

```
$string = "123456789";
```

Then:

```
$r->write($string);
```

sends:

```
123456789
```

Whereas:

```
$r->write($string, 3);
```

sends:

```
123
```

And:

```
$r->write($string, 3, 5);
```

sends:

```
678
```

Finally:

```
$r->write($string, -1, 5);
```

sends:

```
6789
```

16.4 TIE Interface

The TIE interface implementation. This interface is used for HTTP request handlers, when running under `SetHandler perl-script` and Perl doesn't have perlio enabled.

See the *perltie* manpage for more information.

16.4.1 BINMODE

- **since: 2.0.00**

NoOP

See the *binmode* Perl entry in the *perlfunc* manpage

16.4.2 CLOSE

- **since: 2.0.00**

NoOP

See the *close* Perl entry in the *perlfunc* manpage

16.4.3 FILENO

- **since: 2.0.00**

See the *fileno* Perl entry in the *perlfunc* manpage

16.4.4 GETC

- **since: 2.0.00**

See the *getc* Perl entry in the *perlfunc* manpage

16.4.5 OPEN

- **since: 2.0.00**

See the *open* Perl entry in the *perlfunc* manpage

16.4.6 PRINT

- **since: 2.0.00**

See the *print* Perl entry in the *perlfunc* manpage

16.4.7 PRINTF

- **since: 2.0.00**

See the *printf* Perl entry in the *perlfunc* manpage

16.4.8 READ

- **since: 2.0.00**

See the *read* Perl entry in the *perlfunc* manpage

16.4.9 TIEHANDLE

- **since: 2.0.00**

See the *tie* Perl entry in the *perlfunc* manpage

16.4.10 UNTIE

- **since: 2.0.00**

NoOP

See the *untie* Perl entry in the *perlfunc* manpage

16.4.11 WRITE

- **since: 2.0.00**

See the *write* Perl entry in the *perlfunc* manpage

16.5 Deprecated API

The following methods are deprecated, Apache plans to remove those in the future, therefore avoid using them.

16.5.1 get_client_block

This method is deprecated since the C implementation is buggy and we don't want you to use it at all. Instead use the plain `$r->read()`.

16.5.2 setup_client_block

This method is deprecated since `$r->get_client_block` is deprecated.

16.5.3 should_client_block

This method is deprecated since `$r->get_client_block` is deprecated.

16.6 See Also

mod_perl 2.0 documentation.

16.7 Copyright

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

16.8 Authors

The mod_perl development team and numerous contributors.

17 Apache2::RequestRec - Perl API for Apache request record accessors

17.1 Synopsis

```
use Apache2::RequestRec ();

# set supported by the handler HTTP methods
$allowed = $r->allowed();

# auth type
$auth_type = $r->ap_auth_type();

# QUERY_STRING
$args = $r->args();

# non-parsed-headers handler
$status = $r->assbackwards();

# how many bytes were sent
$bytes_sent = $r->bytes_sent();

# client connection record
$c = $r->connection();

# "Content-Encoding" HTTP response header
$r->content_encoding("gzip");

# the languages of the content
$languages = $r->content_languages();

# "Content-Encoding" HTTP response header
$r->content_type('text/plain');

# special response headers table
$err_headers_out = $r->err_headers_out();

# request mapped filename
$filename = $r->filename();

# request finfo
$finfo = $r->finfo();

# 'SetHandler perl-script' equivalent
$r->handler('perl-script');


# was it a HEAD request?
$status = $r->header_only();

# request input headers table
$headers_in = $r->headers_in();

# request output headers table
$headers_out = $r->headers_out();

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

```
# input filters stack
$input_filters = $r->input_filters();

# get the main request obj in a sub-request
$main_r = $r->main();

# what's the current request (GET/POST/etc)?
$method = $r->method();

# what's the current method number?
$methnum = $r->method_number();

# current resource last modified time
$mtime = $r->mtime();

# next request object (in redirect)
$next_r = $r->next();

# there is no local copy
$r->no_local_copy();

# Apache ascii notes table
$notes = $r->notes();

# output filters stack
$output_filters = $r->output_filters();

# PATH_INFO
$path_info = $r->path_info();

# used in configuration directives modules
$per_dir_config = $r->per_dir_config();

# pool with life span of the current request
$p = $r->pool();

# previous request object in the internal redirect
$prev_r = $r->prev();

# connection level input filters stack
$proto_input_filters = $r->proto_input_filters();

# HTTP protocol version number
$proto_num = $r->proto_num();

# connection level output filters stack
$proto_output_filters = $r->proto_output_filters();

# the protocol, the client speaks: "HTTP/1.0", "HTTP/1.1", etc.
$protocol = $r->protocol();

# is it a proxy request
$status = $r->proxyreq($val);

# Time when the request started
$request_time = $r->request_time();
```

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

# response status
$status = $r->status();

# response status line
$status_line = $r->status_line();

# manipulate %ENV of the subprocess

$r->subprocess_env;
$r->subprocess_env($key => $val);

# first HTTP request header
$request = $r->the_request();

# the URI without any parsing performed
$unparsed_uri = $r->unparsed_uri();

# The path portion of the URI
$uri = $r->uri();

# auth username
$user = $r->user();
```

17.2 Description

`Apache2::RequestRec` provides the Perl API for Apache request_rec object.

The following packages extend the `Apache2::RequestRec` functionality: `Apache2::Access`, `Apache2::Log`, `Apache2::RequestIO`, `Apache2::RequestUtil`, `Apache2::Response`, `Apache2::SubRequest` and `Apache2::URI`.

17.3 API

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

17.3.1 `allowed`

Get/set the allowed methods bitmask.

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

- **obj: `$r` (`Apache2::RequestRec` object)**
- **opt arg1: `$new_allowed` (bitmask)**

Set the bitvector.

- **ret: `$allowed` (bitmask)**

  returns `$allowed`, which is a bitvector of the allowed methods.

  If the `$new_allowed` argument is passed, the value before the change is returned.

- **since: 2.0.00**

A handler must ensure that the request method is one that it is capable of handling. Generally modules should `Apache2::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 `Apache2::Const::HTTP_METHOD_NOT_ALLOWED` (405) and `Apache2::Const::HTTP_NOT_IMPLEMENTED` (501) status codes.

Since the default Apache handler deals with the `OPTIONS` method, all response handlers can usually decline to deal with `OPTIONS`. For example if the response handler handles only `GET` and `POST` methods, and not `OPTIONS`, it may want to say:

```
use Apache2::Const -compile => qw(OK DECLINED M_GET M_POST M_OPTIONS);
if ($r->method_number == Apache2::Const::M_OPTIONS) {
    $r->allowed($r->allowed | (1<<Apache2::Const::M_GET) | (1<<Apache2::Const::M_POST));
    return Apache2::Const::DECLINED;
}
```

`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 `Apache2::Const::HTTP_METHOD_NOT_ALLOWED`. Unfortunately this means that a script `GET` handler can't be installed by mod_actions.

For example, if the module can handle only POST method it could start with:

```
use Apache2::Const -compile => qw(M_POST HTTP_METHOD_NOT_ALLOWED);
unless ($r->method_number == Apache2::Const::M_POST) {
    $r->allowed($r->allowed | (1<<Apache2::Const::M_POST));
    return Apache2::Const::HTTP_METHOD_NOT_ALLOWED;
}
```

17.3.2 `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);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$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.

- **since: 2.0.00**

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.

17.3.3 `args`

Get/set the request QUERY string

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_args` (string)**

  Optionally set the new QUERY string

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

  The current QUERY string

  If `$new_args` was passed, returns the value before the change.

- **since: 2.0.00**

17.3.4 `assbackwards`

When set to a true value, Apache won't send any HTTP response headers allowing you to send any headers.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$newval` (integer)**

  assign a new state.

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

  current state.

- **since: 2.0.00**

If you send your own set of headers, which includes the `Keep-Alive` HTTP response header, you must make sure to increment the number of requests served over this connection (which is normally done by the core connection output filter `ap_http_header_filter`, but skipped when `assbackwards` is enabled).

```
$r->connection->keepalives($r->connection->keepalives + 1);
```

otherwise code relying on the value of `$r->connection->keepalives` may malfunction. For example, this counter is used to tell when a new request is coming in over the same connection to a filter that wants to parse only HTTP headers (like `Apache2::Filter::HTTPHeadersFixup`). Of course you will need to set `$r->connection->keepalive(1)`) as well.

17.3.5 `bytes_sent`

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

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$bytes_sent` (integer)**
- **since: 2.0.00**

Though as of this writing in Apache 2.0 it doesn't really do what it did in Apache 1.3. It's just set to the size of the response body. The issue is that buckets from one request may get buffered and not sent during the lifetime of the request, so it's not easy to give a truly accurate count of "bytes sent to the network for this response".

17.3.6 `connection`

Get the client connection record

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$c` (`Apache2::Connection object`)**
- **since: 2.0.00**

17.3.7 content_encoding

Get/set content encoding (the "Content-Encoding" HTTP header). Content encodings are string like *"gzip"* or *"compress"*.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_ce` (string)**

  If passed, sets the content encoding to a new value. It must be a lowercased string.

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

  The current content encoding.

  If `$new_ce` is passed, then the previous value is returned.

- **since: 2.0.00**

For example, here is how to send a gzip'ed response:

```
require Compress::Zlib;
$r->content_type("text/plain");
$r->content_encoding("gzip");
$r->print(Compress::Zlib::memGzip("some text to be gzipped));
```

17.3.8 content_languages

Get/set content languages (the `"Content-Language"` HTTP header). Content languages are string like *"en"* or *"fr"*.

```
$languages = $r->content_languages();
$prev_lang = $r->content_languages($nev_lang);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_lang` (ARRAY ref)**

  If passed, sets the content languages to new values. It must be an ARRAY reference of language names, like *"en"* or *"fr"*

- **ret: `$languages` (ARRAY ref)**

  The current list of content languages, as an ARRAY reference.

  If `$new_lang` is passed, then the previous value is returned.

- **since: 2.0.00**

17.3.9 `content_type`

Get/set the HTTP response *Content-type* header value.

```
my $content_type      = $r->content_type();
my $prev_content_type = $r->content_type($new_content_type);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_content_type` (MIME type string)**

  Assign a new HTTP response content-type. It will affect the response only if HTTP headers weren't sent yet.

- **ret: `$content_type`**

  The current content-type value.

  If `$new_content_type` was passed, the previous value is returned instead.

- **since: 2.0.00**

For example, set the `Content-type` header to *text/plain*.

```
$r->content_type('text/plain');
```

If you set this header via the `headers_out` table directly, it will be ignored by Apache. So do not do that.

17.3.10 `err_headers_out`

Get/set MIME response headers, printed even on errors and persist across internal redirects.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$err_headers_out` (`APR::Table object`)**
- **since: 2.0.00**

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).

For example, if a handler wants to return a 404 response, but nevertheless to set a cookie, it has to be:

```
$r->err_headers_out->add('Set-Cookie' => $cookie);
return Apache2::Const::NOT_FOUND;
```

If the handler does:

```
$r->headers_out->add('Set-Cookie' => $cookie);
return Apache2::Const::NOT_FOUND;
```

the `Set-Cookie` header won't be sent.

17.3.11 `filename`

Get/set the filename on disk corresponding to this response (the result of the *URI --> filename* translation).

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_filename` (string)**

  new value

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

  the current filename, or the previous value if the optional `$new_filename` argument was passed

- **since: 2.0.00**

Note that if you change the filename after the `PerlMapToStorageHandler` phase was run and expect Apache to serve it, you need to update its `stat` record, like so:

```
use Apache2::RequestRec ();
use APR::Finfo ();
use APR::Const -compile => qw(FINFO_NORM);
$r->filename($newfile);
$r->finfo(APR::Finfo::stat($newfile, APR::Const::FINFO_NORM, $r->pool));
```

if you don't, Apache will still try to use the previously cached information about the previously set value of the filename.

17.3.12 `finfo`

Get and set the *finfo* request record member:

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$finfo` (`APR::Finfo object`)**
- **ret: `$finfo` (`APR::Finfo object`)**

Always returns the current object.

Due to the internal Apache implementation it's not possible to have two different objects originating from `$r->finfo` at the same time. Whenever `$r->finfo` is updated all objects will be updated too to the latest value.

- **since: 2.0.00**

Most of the time, this method is used to get the `finfo` member. The only reason you may want to set it is you need to use it before the Apache's default map_to_storage phase is called.

Examples:

- What Apache thinks is the current request filename (post the `PerlMapToStorageHandler` phase):

```
use Apache2::RequestRec ();
use APR::Finfo ();
print $r->finfo->fname;
```

- Populate the `finfo` member (normally, before the `PerlMapToStorageHandler` phase):

```
use APR::Finfo ();
use APR::Const -compile => qw(FINFO_NORM);

my $finfo = APR::Finfo::stat(__FILE__, APR::Const::FINFO_NORM, $r->pool);
$r->finfo($finfo);
```

17.3.13 handler

Get/set the equivalent of the `SetHandler` directive.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_handler` (string)**

  the new handler.

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

  the current handler.

  If `$new_handler` is passed, the previous value is returned.

- **since: 2.0.00**

17.3.14 header_only

Did the client has asked for headers only? e.g. if the request method was **HEAD**.

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $status (boolean)**

  Returns true if the client is asking for headers only, false otherwise

- **since: 2.0.00**

17.3.15 headers_in

Get/set the request MIME headers:

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $headers_in (APR::Table object)**
- **since: 2.0.00**

This table is available starting from the `PerlHeaderParserHandler` phase.

For example you can use it to retrieve the cookie value sent by the client, in the `Cookie:` header:

```
my $cookie = $r->headers_in->{Cookie} || '';
```

17.3.16 headers_out

Get/set MIME response headers, printed only on 2xx responses.

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $headers_out (APR::Table object)**
- **since: 2.0.00**

See also `err_headers_out`, which allows to set headers for non-2xx responses and persist across internal redirects.

17.3.17 hostname

Host, as set by full URI or Host:

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_hostname` (string)**

  new value

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

  the current hostname, or the previous value if the optional `$new_hostname` argument was passed

- **since: 2.0.00**

17.3.18 `input_filters`

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

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_input_filters`**

  Set a new value

- **ret: `$input_filters` (`Apache2::Filter object`)**

  The first filter in the request level input filters chain.

  If `$new_input_filters` was passed, returns the previous value.

- **since: 2.0.00**

For example instead of using `$r->read()` to read the POST data, one could use an explicit walk through incoming bucket brigades to get that data. The following function `read_post()` does just that (in fact that's what `$r->read()` does behind the scenes):

```
use APR::Brigade ();
use APR::Bucket ();
use Apache2::Filter ();

use Apache2::Const -compile => qw(MODE_READBYTES);
use APR::Const    -compile => qw(SUCCESS BLOCK_READ);

use constant IOBUFSIZE => 8192;

sub read_post {
    my $r = shift;

    my $bb = APR::Brigade->new($r->pool,
                               $r->connection->bucket_alloc);
```

```
    my $data = '';
    my $seen_eos = 0;
    do {
        $r->input_filters->get_brigade($bb, Apache2::Const::MODE_READBYTES,
                                       APR::Const::BLOCK_READ, IOBUFSIZE);

        for (my $b = $bb->first; $b; $b = $bb->next($b)) {
            if ($b->is_eos) {
                $seen_eos++;
                last;
            }

            if ($b->read(my $buf)) {
                $data .= $buf;
            }

            $b->remove; # optimization to reuse memory
        }

    } while (!$seen_eos);

    $bb->destroy;

    return $data;
}
```

As you can see `$r->input_filters` gives us a pointer to the last of the top of the incoming filters stack.

17.3.19 `main`

Get the main request record

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$main_r` (`Apache2::RequestRec object`)**

  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`.

- **since: 2.0.00**

17.3.20 method

Get/set the current request method (e.g. GET, HEAD, POST, etc.):

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

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_method (string)**

  a new value

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

  The current method as a string

  if $new_method was passed the previous value is returned.

- **since: 2.0.00**

17.3.21 method_number

Get/set the HTTP method, issued by the client (Apache2::Const::M_GET, Apache2::Const::M_POST, etc.)

```
$methnum      = $r->method_number();
$prev_methnum = $r->method_number($new_methnum);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_methnum (Apache2::Const :methods constant)**

  a new value

- **ret: $methnum (Apache2::Const :methods constant)**

  The current method as a number

  if $new_methnum was passed the previous value is returned.

- **since: 2.0.00**

See the $r->allowed entry for examples.

17.3.22 mtime

Last modified time of the requested resource

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_mtime` (epoch seconds).**

  a new value

- **ret: `$mtime` (epoch seconds).**

  the current value

  if `$new_mtime` was passed the previous value is returned.

- **since: 2.0.00**

17.3.23 next

Pointer to the redirected request if this is an external redirect

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$next_r` (`Apache2::RequestRec object`)**

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

- **since: 2.0.00**

17.3.24 no_local_copy

There is no local copy of this response

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$status` (integer)**
- **since: 2.0.00**

Used internally in certain sub-requests to prevent sending `Apache2::Const::HTTP_NOT_MODIFIED` for a fragment or error documents. For example see the implementation in *modules/filters/mod_include.c*.

Also used internally in `$r->meets_conditions` -- if set to a true value, the conditions are always met.

17.3.25 notes

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

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

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_notes (APR::Table object)**
- **ret: $notes (APR::Table object)**

  the current notes table.

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

- **since: 2.0.00**

If you want to pass Perl structures, you can use $r->pnotes.

Also see $c->notes

17.3.26 output_filters

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

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

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_output_filters**

  Set a new value

- **ret: $output_filters (Apache2::Filter object)**

  The first filter in the request level output filters chain.

  If $new_output_filters was passed, returns the previous value.

- **since: 2.0.00**

For example instead of using $r->print() to send the response body, one could send the data directly to the first output filter. The following function send_response_body() does just that:

```
use APR::Brigade ();
use APR::Bucket ();
use Apache2::Filter ();

sub send_response_body {
```

```
    my ($r, $data) = @_;

    my $bb = APR::Brigade->new($r->pool,
                               $r->connection->bucket_alloc);

    my $b = APR::Bucket->new($bb->bucket_alloc, $data);
    $bb->insert_tail($b);
    $r->output_filters->fflush($bb);
    $bb->destroy;
}
```

In fact that's what `$r->read()` does behind the scenes. But it also knows to parse HTTP headers passed together with the data and it also implements buffering, which the above function does not.

17.3.27 path_info

Get/set the `PATH_INFO`, what is left in the path after the *URI --> filename* translation:

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$path_info` (string)**

  Set a new value

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

  Return the current value.

  If the optional argument `$path_info` is passed, the previous value is returned.

- **since: 2.0.00**

17.3.28 per_dir_config

Get the dir config vector:

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$per_dir_config` (`Apache2::ConfVector object`)**
- **since: 2.0.00**

For an indepth discussion, refer to the Apache Server Configuration Customization in Perl chapter.

17.3.29 pool

The pool associated with the request

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $p (APR::Pool object)**
- **since: 2.0.00**

17.3.30 prev

Pointer to the previous request if this is an internal redirect

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $prev_r (Apache2::RequestRec object)**

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

- **since: 2.0.00**

17.3.31 proto_input_filters

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

```
$proto_input_filters      = $r->proto_input_filters();
$prev_proto_input_filters = $r->proto_input_filters($new_proto_input_filters);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_proto_input_filters**

  Set a new value

- **ret: $proto_input_filters (Apache2::Filter object)**

  The first filter in the protocol level input filters chain.

  If `$new_proto_input_filters` was passed, returns the previous value.

- **since: 2.0.00**

`$r->proto_input_filters` points to the same filter as `$r->connection->input_filters`.

17.3.32 proto_num

Get current request's HTTP protocol version number

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $proto_num (integer)**

  current request's HTTP protocol version number, e.g.: HTTP/1.0 == 1000, HTTP/1.1 = 1001

- **since: 2.0.00**

17.3.33 proto_output_filters

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

```
$proto_output_filters      = $r->proto_output_filters();
$prev_proto_output_filters = $r->proto_output_filters($new_proto_output_filters);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_proto_output_filters**

  Set a new value

- **ret: $proto_output_filters (Apache2::Filter object)**

  The first filter in the protocol level output filters chain.

  If `$new_proto_output_filters` was passed, returns the previous value.

- **since: 2.0.00**

`$r->proto_output_filters` points to the same filter as `$r->connection->output_filters`.

17.3.34 protocol

Get a string identifying the protocol that the client speaks.

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $protocl (string)**

  Typical values are `"HTTP/1.0"` or `"HTTP/1.1"`.

If the client didn't specify the protocol version, the default is `"HTTP/0.9"`

- **since: 2.0.00**

17.3.35 proxyreq

Get/set the *proxyrec* request record member and optionally adjust other related fields.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$val` (integer)**

  PROXYREQ_NONE, PROXYREQ_PROXY, PROXYREQ_REVERSE, PROXYREQ_RESPONSE

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

  If `$val` is defined the *proxyrec* member will be set to that value and previous value will be returned.

  If `$val` is not passed, and `$r->proxyreq` is not true, and the proxy request is matching the current vhost (scheme, hostname and port), the *proxyrec* member will be set to PROXYREQ_PROXY and that value will be returned. In addition `$r->uri` is set to `$r->unparsed_uri` and `$r->filename` is set to `"modperl-proxy:".$r->uri`. If those conditions aren't true 0 is returned.

- **since: 2.0.00**

For example to turn a normal request into a proxy request to be handled on the same server in the `PerlTransHandler` phase run:

```
my $real_url = $r->unparsed_uri;
$r->proxyreq(Apache2::Const::PROXYREQ_PROXY);
$r->uri($real_url);
$r->filename("proxy:$real_url");
$r->handler('proxy-server');
```

Also remember that if you want to turn a proxy request into a non-proxy request, it's not enough to call:

```
$r->proxyreq(Apache2::Const::PROXYREQ_NONE);
```

You need to adjust `$r->uri` and `$r->filename` as well if you run that code in `PerlPostReadRequestHandler` phase, since if you don't -- `mod_proxy`'s own post_read_request handler will override your settings (as it will run after the mod_perl handler).

And you may also want to add

```
$r->set_handlers(PerlResponseHandler => []);
```

so that any response handlers which match apache directives will not run in addition to the mod_proxy content handler.

17.3.36 request_time

Time when the request started

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $request_time (epoch seconds).**
- **since: 2.0.00**

17.3.37 server

Get the Apache2::Server object for the server the request $r is running under.

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $s (Apache2::ServerRec object)**
- **since: 2.0.00**

17.3.38 status

Get/set the reply status for the client request.

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

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_status (integer)**

  If $new_status is passed the new status is assigned.

  Normally you would use some Apache2::Const constant, e.g. Apache2::Const::REDIRECT.

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

  The current value.

  If $new_status is passed the old value is returned.

- **since: 2.0.00**

Usually you will set this value indirectly by returning the status code as the handler's function result. However, there are rare instances when you want to trick Apache into thinking that the module returned an `Apache2::Const::OK` status code, but actually send the browser a non-OK status. This may come handy when implementing an HTTP proxy handler. The proxy handler needs to send to the client, whatever status code the proxied server has returned, while returning `Apache2::Const::OK` to Apache. e.g.:

```
$r->status($some_code);
return Apache2::Const::OK
```

See also `$r->status_line`, which. if set, overrides `$r->status`.

17.3.39 `status_line`

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

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_status_line` (string)**
- **ret: `$status_line` (string)**
- **since: 2.0.00**

When discussing `$r->status` we have mentioned that sometimes a handler runs to a successful completion, but may need to return a different code, which is the case with the proxy server. Assuming that the proxy handler forwards to the client whatever response the proxied server has sent, it'll usually use `status_line()`, like so:

```
$r->status_line($response->code() . ' ' . $response->message());
return Apache2::Const::OK;
```

In this example `$response` could be for example an `HTTP::Response` object, if `LWP::UserAgent` was used to implement the proxy.

This method is also handy when you extend the HTTP protocol and add new response codes. For example you could invent a new error code and tell Apache to use that in the response like so:

```
$r->status_line("499 We have been FooBared");
return Apache2::Const::OK;
```

Here `499` is the new response code, and *We have been FooBared* is the custom response message.

17.3.40 subprocess_env

Get/set the Apache `subprocess_env` table, or optionally set the value of a named entry.

```
             $r->subprocess_env;
$env_table = $r->subprocess_env;

       $r->subprocess_env($key => $val);
$val = $r->subprocess_env($key);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$key` (string)**
- **opt arg2: `$val` (string)**
- **ret: `...`**
- **since: 2.0.00**

When called in VOID context with no arguments, it populate `%ENV` with special variables (e.g. `$ENV{QUERY_STRING}`) like mod_cgi does.

When called in a non-VOID context with no arguments, it returns an `APR::Table object`.

When the `$key` argument (string) is passed, it returns the corresponding value (if such exists, or `undef`. The following two lines are equivalent:

```
$val = $r->subprocess_env($key);
$val = $r->subprocess_env->get($key);
```

When the `$key` and the `$val` arguments (strings) are passed, the value is set. The following two lines are equivalent:

```
$r->subprocess_env($key => $val);
$r->subprocess_env->set($key => $val);
```

The `subprocess_env table` is used by `Apache2::SubProcess`, to pass environment variables to externally spawned processes. It's also used by various Apache modules, and you should use this table to pass the environment variables. For example if in `PerlHeaderParserHandler` you do:

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

you can then deploy `mod_include` and write in *.shtml* document:

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

17.3.41 the_request

First HTTP request header

```
$request = $r->the_request();
$old_request = $r->uri($new_request);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_request (string)**
- **ret: $request (string)**

  For example:

  ```
  GET /foo/bar/my_path_info?args=3 HTTP/1.0
  ```

- **since: 2.0.00**

17.3.42 unparsed_uri

The URI without any parsing performed

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

- **obj: $r (Apache2::RequestRec object)**
- **ret: $unparsed_uri (string)**
- **since: 2.0.00**

If for example the request was:

```
GET /foo/bar/my_path_info?args=3 HTTP/1.0
```

`$r->uri` returns:

```
/foo/bar/my_path_info
```

whereas `$r->unparsed_uri` returns:

```
/foo/bar/my_path_info?args=3
```

17.3.43 uri

The path portion of the URI

```
$uri         = $r->uri();
my $prec_uri = $r->uri($new_uri);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_uri (string)**
- **ret: $uri (string)**

- **since: 2.0.00**

See the example in the `$r->unparsed_uri` section.

17.3.44 `user`

Get the user name, if an authentication process was successful. Or set it.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_user` (string)**

  Pass `$new_user` to set a new value

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

  The current username if an authentication process was successful.

  If `$new_user` was passed, the previous value is returned.

- **since: 2.0.00**

For example, let's print the username passed by the client:

```
my ($res, $sent_pw) = $r->get_basic_auth_pw;
return $res if $res != Apache2::Const::OK;
print "User: ", $r->user;
```

17.4 Unsupported API

`Apache2::RequestRec` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

17.4.1 `allowed_methods`

META: Autogenerated - needs to be reviewed/completed

List of allowed methods

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$list` (*`Apache2::MethodList object`*)**
- **since: 2.0.00**

META: Apache2::MethodList is not available at the moment

17.4.2 `allowed_xmethods`

META: Autogenerated - needs to be reviewed/completed

Array of extension methods

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$array` (*`APR::ArrayHeader object`*)**
- **since: 2.0.00**

META: APR::ArrayHeader is not available at the moment

17.4.3 `request_config`

Config vector containing pointers to request's per-server config structures

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$newval` (`Apache2::ConfVector object`)**
- **since: 2.0.00**

17.4.4 `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);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$newval` (integer)**
- **since: 2.0.00**

17.5 See Also

mod_perl 2.0 documentation.

17.6 Copyright

17.7 Authors

The mod_perl development team and numerous contributors.

18 Apache2::RequestUtil - Perl API for Apache request record utils

18.1 Synopsis

```
use Apache2::RequestUtil ();

# add httpd config dynamically
$r->add_config([’require valid-user’]);

# dump the request object as a string
print $r->as_string();

# default content_type
$content_type = $r->default_type();

# get PerlSetVar/PerlAddVar values
@values = $r->dir_config->get($key);

# get server docroot
$docroot = $r->document_root();

# set server docroot
$r->document_root($new_root);

# what are the registered perl handlers for a given phase
my @handlers = @{ $r->get_handlers(’PerlResponseHandler’) || [] };

# push a new handler for a given phase
$r->push_handlers(PerlCleanupHandler => \&handler);

# set handlers for a given phase (resetting previous values)
$r->set_handlers(PerlCleanupHandler => []);

# what’s the request body limit
$limit = $r->get_limit_req_body();

# server and port names
$server = $r->get_server_name();
$port   = $r->get_server_port();

# what string Apache is going to send for a given status code
$status_line = Apache2::RequestUtil::get_status_line(404);

# are we in the main request?

$is_initial = $r->is_initial_req();

# directory level PerlOptions flags lookup
$r->subprocess_env unless $r->is_perl_option_enabled(’SetupEnv’);

# current <Location> value
$location = $r->location();

# merge a <Location> container in a request object
$r->location_merge($location);

# create a new Apache2::RequestRec object
$r = Apache2::RequestRec->new($c);
```

```
# tell the client not to cache the response
$r->no_cache($boolean);

# share perl objects by reference like $r->notes
$r->pnotes($key => [$obj1, $obj2]);

# get HTML signature
$sig = $r->psignature($prefix);

# get the global request object (requires PerlOptions +GlobalRequest)
$r = Apache2::RequestUtil->request;

# insert auth credentials into the request as if the client did that
$r->set_basic_credentials($username, $password);

# slurp the contents of $r->filename
my $content = ${ $r->slurp_filename() };

# terminate the current child after this request
$r->child_terminate();
```

18.2 Description

`Apache2::RequestUtil` provides the Apache request object utilities API.

18.3 API

18.3.1 `add_config`

Dynamically add Apache configuration at request processing runtime:

```
$r->add_config($lines);
$r->add_config($lines, $override);
$r->add_config($lines, $override, $path);
$r->add_config($lines, $override, $path, $override_opts);
```

Configuration directives are processed as if given in a `<Location>` block.

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$lines` (ARRAY ref)**

  An ARRAY reference containing configuration lines per element, without the new line terminators.

- **opt arg2: `$override` (`Apache2::Const override constant`)**

  Which allow-override bits are set

Default value is: `Apache2::Const::OR_AUTHCFG`

- **opt arg3: `$path` (string)**

  Set the `Apache2::CmdParms object path` component. This is the path of the `<Location>` block. Some directives need this, for example `ProxyPassReverse`.

  If an empty string is passed a `NULL` pointer is passed further at C-level. This is necessary to make something like this work:

```
$r->add_config( [
                 '<Directory />',
                 'AllowOverride Options AuthConfig',
                 '</Directory>',
                ], ~0, '' );
```

  Note: `AllowOverride` is valid only in directory context.

  Caution: Some directives need a non-empty path otherwise they cause segfaults. Thus, use the empty path with caution.

  Default value is: `/`

- **opt arg4: `$override_opts` (`Apache2::Const options constant`)**

  Apache limits the applicable directives in certain situations with `AllowOverride`. With Apache 2.2 comes the possibility to enable or disable single options, for example

```
AllowOverride AuthConfig Options=ExecCGI,Indexes
```

  Internally, this directive is parsed into 2 bit fields that are represented by the `$override` and `$override_opts` parameters to `add_config`. The above example is parsed into an `$override` with 2 bits set, one for `AuthConfig` the other for `Options` and an `$override_opts` with 2 bits set for ExecCGI and Indexes.

  When applying other directives, for example `AuthType` or `Options` the appropriate bits in `$override` must be set. For the `Options` directive additionally `$override_opts` bits must be set.

  The `$override` and `$override_opts` parameters to `add_config` are valid while applying `$lines`.

  `$override_opts` is new in Apache 2.2. The mod_perl implementation for Apache 2.0 lets you pass the parameter but ignores it.

  Default for `$override_opts` is: `Apache2::Const::OPT_UNSET` | `Apache2::Const::OPT_ALL` | `Apache2::Const::OPT_INCNOEXEC` | `Apache2::Const::OPT_SYM_OWNER` | `Apache2::Const::OPT_MULTI`

That means, all options are allowed.

- **ret: no return value**
- **since: 2.0.00, `$path` and `$override_opts` since 2.0.3**

See also: `$s->add_config`

For example:

```
use Apache2::RequestUtil ();
use Apache2::Access ();

$r->add_config(['require valid-user']);

# this regards the current AllowOverride setting
$r->add_config(['AuthName secret',
                'AuthType Basic',
                'Options ExecCGI'],
               $r->allow_override, $path, $r->allow_override_opts);
```

18.3.2 `as_string`

Dump the request object as a string

```
$dump = $r->as_string();
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$dump` (string)**
- **since: 2.0.00**

Dumps various request and response headers (mainly useful for debugging)

18.3.3 `child_terminate`

Terminate the current worker process as soon as the current request is over

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: no return value**
- **since: 2.0.00**

This method is not supported in threaded MPMs

18.3.4 `default_type`

Retrieve the value of the DefaultType directive for the current request. If not set `text/plain` is returned.

```
$content_type = $r->default_type();
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  The default type

- **since: 2.0.00**
- **removed from the `httpd` API in version 2.3.2**

18.3.5 `dir_config`

`$r->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 = $r->dir_config->get($key);
$r->dir_config($key, $val);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg2: `$key` (string)**

  Key string

- **opt arg3: `$val` (string)**

  Value string

- **ret: ...**

  Depends on the passed arguments, see further discussion

- **since: 2.0.00**

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.

```
$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 assosiated 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.

```
@values = $r->dir_config->get($key);
```

To receive a list of values you must use get() method from the APR::Table class.

```
$r->dir_config($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.

18.3.6 document_root

Retrieve the document root for this server

```
$docroot = $r->document_root();
$docroot = $r->document_root($new_root);
```

- **obj: $r (Apache2::RequestRec object)**

  The current request

- **opt arg1: $new_root**

  Sets the document root to a new value **only for the duration of the current request**.

  Note the limited functionality under threaded MPMs.

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

  The document root

- **since: 2.0.00**

18.3.7 get_handlers

Returns a reference to a list of handlers enabled for a given phase.

```
$handlers_list = $r->get_handlers($hook_name);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$hook_name` (string)**

  a string representing the phase to handle (e.g. `PerlLogHandler`)

- **ret: `$handlers_list` (ref to an ARRAY of CODE refs)**

  a list of handler subroutines CODE references

- **since: 2.0.00**

See also: `$s->add_config`

For example:

A list of handlers configured to run at the response phase:

```
my @handlers = @{ $r->get_handlers('PerlResponseHandler') || [] };
```

18.3.8 `get_limit_req_body`

Return the limit on bytes in request msg body

```
$limit = $r->get_limit_req_body();
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  the maximum number of bytes in the request msg body

- **since: 2.0.00**

18.3.9 `get_server_name`

Get the current request's server name

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  the server name

- **since: 2.0.00**

For example, consruct a hostport string:

```
use Apache2::RequestUtil ();
my $hostport = join ':', $r->get_server_name, $r->get_server_port;
```

18.3.10 `get_server_port`

Get the current server port

```
$port = $r->get_server_port();
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  The server's port number

- **since: 2.0.00**

For example, consruct a hostport string:

```
use Apache2::RequestUtil ();
my $hostport = join ':', $r->get_server_name, $r->get_server_port;
```

18.3.11 `get_status_line`

Return the `Status-Line` for a given status code (excluding the HTTP-Version field).

```
$status_line = Apache2::RequestUtil::get_status_line($status);
```

- **arg1: `$status` (integer)**

  The HTTP status code

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

  The Status-Line

  If an invalid or unknown status code is passed, `"500 Internal Server Error"` will be returned.

- **since: 2.0.00**

For example:

```
use Apache2::RequestUtil ();
print Apache2::RequestUtil::get_status_line(400);
```

will print:

```
400 Bad Request
```

18.3.12 is_initial_req

Determine whether the current request is the main request or a sub-request

```
$is_initial = $r->is_initial_req();
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  A request or a sub-request object

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

  If true -- it's the main request, otherwise it's a sub-request

- **since: 2.0.00**

18.3.13 is_perl_option_enabled

check whether a directory level `PerlOptions` flag is enabled or not.

```
$result = $r->is_perl_option_enabled($flag);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$flag` (string)**
- **ret: `$result` (boolean)**
- **since: 2.0.00**

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.

18.3.14 location

Get the path of the <Location> section from which the current `Perl*Handler` is being called.

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$location` (string)**
- **since: 2.0.00**

18.3.15 `location_merge`

Merge a given `<Location>` container into the current request object:

```
$ret = $r->location_merge($location);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$location` (string)**

  The argument in a `<Location>` section. For example to merge a container:

  ```
  <Location /foo>
      ...
  </Location>
  ```

  that argument will be */foo*

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

  a true value if the merge was successful (i.e. the request `$location` match was found), otherwise false.

- **since: 2.0.00**

Useful for insertion of a configuration section into a custom `Apache2::RequestRec` object, created via the `Apache2::RequestRec->new()` method. See for example the Command Server protocol example.

18.3.16 `new`

Create a new `Apache2::RequestRec` object.

```
$r = Apache2::RequestRec->new($c);
$r = Apache2::RequestRec->new($c, $pool);
```

- **obj: `Apache2::RequestRec` (`Apache2::RequestRec class name`)**
- **arg1: `$c` (`Apache2::Connection object`)**
- **opt arg2: `$pool`**

  If no `$pool` argument is passed, `$c->pool` is used. That means that the created `Apache2::RequestRec` object will be valid as long as the connection object is valid.

- **ret: `$r` (`Apache2::RequestRec object`)**
- **since: 2.0.00**

It’s possible to reuse the HTTP framework features outside the familiar HTTP request cycle. It’s possible to write your own full or partial HTTP implementation without needing a running Apache server. You will need the `Apache2::RequestRec` object in order to be able to reuse the rich functionality supplied via this object.

See for example the Command Server protocol example which reuses HTTP AAA model under non-HTTP protocol.

18.3.17 no_cache

Add/remove cache control headers:

```
$prev_no_cache = $r->no_cache($boolean);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$boolean` (boolean)**

  A true value sets the `no_cache` request record member to a true value and inserts:

  ```
  Pragma: no-cache
  Cache-control: no-cache
  ```

  into the response headers, indicating that the data being returned is volatile and the client should not cache it.

  A false value unsets the `no_cache` request record member and the mentioned headers if they were previously set.

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

  Should you care, the `no_cache` request record member value prior to the change is returned.

- **since: 2.0.00**

This method should be invoked before any response data has been sent out.

18.3.18 pnotes

Share Perl variables between Perl HTTP handlers

```
# to share variables by value and not reference, $val should be a lexical.
$old_val  = $r->pnotes($key => $val);
$val      = $r->pnotes($key);
$hash_ref = $r->pnotes();
```

Note: sharing variables really means it. The variable is not copied. Only its reference count is incremented. If it is changed after being put in pnotes that change also affects the stored value. The following example illustrates the effect:

```
my $v=1;                    my $v=1;
$r->pnotes( 'v'=>$v );      $r->pnotes->{v}=$v;
$v++;                       $v++;
my $x=$r->pnotes('v');      my $x=$r->pnotes->{v};
```

In both cases `$x` is `2` not `1`. See also `Apache2::SafePnotes` on CPAN.

There has been a lot of discussion advocating for pnotes sharing variables by value and not reference. Sharing by reference can create 'spooky action at a distance' effects when the sharing is assumed to share a copy of the value. Tim Bunce offers the following summary and suggestion for sharing by value.

What's wrong with this code:

```
sub foo {
    my ($r, $status, $why) = @_;
    $r->pnotes('foo', ($why) ? "$status:$why" : $status);
    return;
}
```

Nothing, except it doesn't work as expected due to this pnotes bug: If the same code is called in a sub-request then the pnote of $r->prev is magically updated at a distance to the same value!

Try explain why that is to anyone not deeply familar with perl internals!

The fix is to avoid pnotes taking a ref to the invisible op_targ embededed in the code by passing a simple lexical variable as the actual argument. That can be done in-line like this:

```
sub mark_as_internally_redirected {
    my ($r, $status, $why) = @_;
    $r->pnotes('foo', my $tmp = (($why) ? "$status:$why" : $status));
    return;
}
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$key` (string)**

  A key value

- **opt arg2: `$val` (SCALAR)**

  Any scalar value (e.g. a reference to an array)

- **ret: (3 different possible values)**

  if both, `$key` and `$val` are passed the previous value for `$key` is returned if such existed, otherwise `undef` is returned.

  if only `$key` is passed, the current value for the given key is returned.

if no arguments are passed, a hash reference is returned, which can then be directly accessed without going through the `pnotes()` interface.

- **since: 2.0.00**

This method provides functionality similar to (`Apache2::RequestRec::notes`), but values can be any Perl variables. That also means that it can be used only between Perl modules.

The values get reset automatically at the end of each HTTP request.

Examples:

Set a key/value pair:

```
$r->pnotes(foo => [1..5]);
```

Get the value:

```
$val = $r->pnotes("foo");
```

`$val` now contains an array ref containing 5 elements (`1..5`).

Now change the existing value:

```
$old_val = $r->pnotes(foo => ['a'..'c']);
$val = $r->pnotes("foo");
```

`$old_val` now contains an array ref with 5 elements (`1..5`) and `$val` contains an array ref with 3 elements `'a'`, `'b'`, `'c'`.

Alternatively you can access the hash reference with all pnotes values:

```
$pnotes = $r->pnotes;
```

Now we can read what's in there for the key *foo*:

```
$val = $pnotes->{foo};
```

and as before `$val` still gives us an array ref with 3 elements `'a'`, `'b'`, `'c'`.

Now we can add elements to it:

```
push @{ $pnotes{foo} }, 'd'..'f';
```

and we can try to retrieve them using the hash and non-hash API:

```
$val1 = $pnotes{foo};
$val2 = $r->pnotes("foo");
```

Both `$val1` and `$val2` contain an array ref with 6 elements (letters 'a' to 'f').

Finally to reset an entry you could just assign `undef` as a value:

```
$r->pnotes(foo => undef);
```

but the entry for the key *foo* still remains with the value `undef`. If you really want to completely remove it, use the hash interface:

```
delete $r->pnotes->{foo};
```

18.3.19 psignature

Get HTML describing the address and (optionally) admin of the server.

```
$sig = $r->psignature($prefix);
```

- **obj: `$r` (`Apache2::RequestRec`)**
- **arg1: `$prefix` (string)**

  Text which is prepended to the return value

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

  HTML text describing the server. Note that depending on the value of the `ServerSignature` directive, the function may return the address, including the admin information or nothing at all.

- **since: 2.0.00**

18.3.20 request

Get/set the (`Apache2::RequestRec object`) object for the current request.

```
$r = Apache2::RequestUtil->request;
     Apache2::RequestUtil->request($new_r);
```

- **obj: `Apache2` (class name)**

  The Apache class name

- **opt arg1: `$new_r` (`Apache2::RequestRec object`)**
- **ret: `$r` (`Apache2::RequestRec object`)**
- **since: 2.0.00**

The get-able part of this method is only available if `PerlOptions +GlobalRequest` is in effect or if `Apache2->request($new_r)` was called earlier. So instead of setting `PerlOptions +Global-Request`, one can set the global request from within the handler.

18.3.21 push_handlers

Add one or more handlers to a list of handlers to be called for a given phase.

```
$ok = $r->push_handlers($hook_name => \&handler);
$ok = $r->push_handlers($hook_name => ['Foo::Bar::handler', \&handler2]);
```

- **obj: $r (Apache2::RequestRec object)**
- **arg1: $hook_name (string)**

  the phase to add the handlers to

- **arg2: $handlers (CODE ref or SUB name or an ARRAY ref)**

  a single handler CODE reference or just a name of the subroutine (fully qualified unless defined in the current package).

  if more than one passed, use a reference to an array of CODE refs and/or subroutine names.

- **ret: $ok (boolean)**

  returns a true value on success, otherwise a false value

- **since: 2.0.00**

  See also: `$s->add_config`

  Note that to push input/output filters you have to use `Apache2::Filter` methods: `add_input_filter` and `add_output_filter`.

Examples:

A single handler:

```
$r->push_handlers(PerlResponseHandler => \&handler);
```

Multiple handlers:

```
$r->push_handlers(PerlFixupHandler => ['Foo::Bar::handler', \&handler2]);
```

Anonymous functions:

```
$r->push_handlers(PerlLogHandler => sub { return Apache2::Const::OK });
```

18.3.22 set_basic_credentials

Populate the incoming request headers table (`headers_in`) with authentication headers for Basic Authorization as if the client has submitted those in first place:

```
$r->set_basic_credentials($username, $password);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$username` (string)**
- **arg2: `$password` (string)**
- **ret: no return value**
- **since: 2.0.00**

See for example the Command Server protocol example which reuses HTTP AAA model under non-HTTP protocol.

18.3.23 `set_handlers`

Set a list of handlers to be called for a given phase. Any previously set handlers are forgotten.

```
$ok = $r->set_handlers($hook_name => \&handler);
$ok = $r->set_handlers($hook_name => ['Foo::Bar::handler', \&handler2]);
$ok = $r->set_handlers($hook_name => []);
$ok = $r->set_handlers($hook_name => undef);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$hook_name` (string)**

  the phase to set the handlers in

- **arg2: `$handlers` (CODE ref or SUB name or an ARRAY ref)**

  a reference to a single handler CODE reference or just a name of the subroutine (fully qualified unless defined in the current package).

  if more than one passed, use a reference to an array of CODE refs and/or subroutine names.

  if the argument is `undef` or `[]` the list of handlers is reset to zero.

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

  returns a true value on success, otherwise a false value

- **since: 2.0.00**

See also: `$s->add_config`

Examples:

A single handler:

```
$r->set_handlers(PerlResponseHandler => \&handler);
```

Multiple handlers:

```
$r->set_handlers(PerlFixupHandler => ['Foo::Bar::handler', \&handler2]);
```

Anonymous functions:

```
$r->set_handlers(PerlLogHandler => sub { return Apache2::Const::OK });
```

Reset any previously set handlers:

```
$r->set_handlers(PerlCleanupHandler => []);
```

or

```
$r->set_handlers(PerlCleanupHandler => undef);
```

18.3.24 slurp_filename

Slurp the contents of `$r->filename`:

```
$content_ref = $r->slurp_filename($tainted);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$tainted` (number)**

  If the server is run under the tainting mode (`-T`) which we hope you do, by default the returned data is tainted. 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_ref` (SCALAR ref)**

  A reference to a string with the contents

- **excpt: `APR::Error`**

  Possible error codes could be: `APR::Const::EACCES` (permission problems), `APR::Const::ENOENT` (file not found), and others. For checking such error codes, see the documentation for, for example, `APR::Status::is_EACCES` and `APR::Status::is_ENOENT`.

- **since: 2.0.00**

Note that if you assign to `$r->filename` you need to update its stat record.

18.4 See Also

mod_perl 2.0 documentation.

18.5 Copyright

18.6 Authors

The mod_perl development team and numerous contributors.

19 Apache2::Response - Perl API for Apache HTTP request response methods

19.1 Synopsis

```
use Apache2::Response ();

$r->custom_response(Apache2::Const::FORBIDDEN, "No Entry today");

$etag = $r->make_etag($force_weak);
$r->set_etag();
$status = $r->meets_conditions();

$mtime_rat = $r->rationalize_mtime($mtime);
$r->set_last_modified($mtime);
$r->update_mtime($mtime);

$r->send_cgi_header($buffer);

$r->set_content_length($length);

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

19.2 Description

`Apache2::Response` provides the Apache request object utilities API for dealing with HTTP response generation process.

19.3 API

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

19.3.1 `custom_response`

Install a custom response handler for a given status

```
$r->custom_response($status, $string);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$status` (`Apache2::Const constant`)**

  The status for which the custom response should be used (e.g. `Apache2::Const::AUTH_REQUIRED`)

- **arg2: `$string` (string)**

  The custom response to use. This can be a static string, or a URL, full or just the uri path (*/foo/bar.txt*).

- **ret: no return value**
- **since: 2.0.00**

`custom_response()` doesn't alter the response code, but is used to replace the standard response body. For example, here is how to change the response body for the access handler failure:

```
package MyApache2::MyShop;
use Apache2::Response ();
use Apache2::Const -compile => qw(FORBIDDEN OK);
sub access {
    my $r = shift;

    if (MyApache2::MyShop::tired_squirrels()) {
        $r->custom_response(Apache2::Const::FORBIDDEN,
            "It's siesta time, please try later");
        return Apache2::Const::FORBIDDEN;
    }

    return Apache2::Const::OK;
}
...

# httpd.conf
PerlModule MyApache2::MyShop
<Location /TestAPI__custom_response>
    AuthName dummy
    AuthType none
    PerlAccessHandler   MyApache2::MyShop::access
    PerlResponseHandler MyApache2::MyShop::response
</Location>
```

When squirrels can't run any more, the handler will return 403, with the custom message:

```
It's siesta time, please try later
```

19.3.2 `make_etag`

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);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$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

- **since: 2.0.00**

19.3.3 meets_conditions

Implements condition `GET` rules for HTTP/1.1 specification. This function inspects the client headers and determines if the response fulfills the specified requirements.

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **ret: `$status` (`Apache2::Const status constant`)**

  `Apache2::Const::OK` if the response fulfills the condition GET rules. Otherwise some other status code (which should be returned to Apache).

- **since: 2.0.00**

Refer to the Generating Correct HTTP Headers document for an indepth discussion of this method.

19.3.4 rationalize_mtime

Return the latest rational time from a request/mtime pair.

```
$mtime_rat = $r->rationalize_mtime($mtime);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$mtime` (time in seconds)**

  The last modified time

- **ret: `$mtime_rat` (time in seconds)**

  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.

- **since: 2.0.00**

19.3.5 send_cgi_header

Parse the header

```
$r->send_cgi_header($buffer);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **arg1: `$buffer` (string)**

  headers and optionally a response body

- **ret: no return value**
- **since: 2.0.00**

This method is really for back-compatibility with mod_perl 1.0. It's very inefficient to send headers this way, because of the parsing overhead.

If there is a response body following the headers it'll be handled too (as if it was sent via `print()`).

Notice that if only HTTP headers are included they won't be sent until some body is sent (again the "send" part is retained from the mod_perl 1.0 method).

19.3.6 set_content_length

Set the content length for this request.

```
$r->set_content_length($length);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$length` (integer)**

  The new content length

- **ret: no return value**
- **since: 2.0.00**

19.3.7 set_etag

Set the E-tag outgoing header

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

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: no return value**
- **since: 2.0.00**

19.3.8 set_keepalive

Set the keepalive status for this request

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

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

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

  true if keepalive can be set, false otherwise

- **since: 2.0.00**

It's called by `ap_http_header_filter()`. For the complete complicated logic implemented by this method see *httpd-2.0/server/http_protocol.c*.

19.3.9 set_last_modified

sets the `Last-Modified` response header field to the value of the mtime field in the request structure -- rationalized to keep it from being in the future.

```
$r->set_last_modified($mtime);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$mtime` (time in seconds)**

  if the `$mtime` argument is passed, $r->update_mtime will be first run with that argument.

- **ret: no return value**
- **since: 2.0.00**

19.3.10 update_mtime

Set the `$r->mtime` field to the specified value if it's later than what's already there.

```
$r->update_mtime($mtime);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$mtime` (time in seconds)**
- **ret: no return value**
- **since: 2.0.00**

See also: $r->set_last_modified.

19.4 Unsupported API

`Apache2::Response` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

19.4.1 `send_error_response`

Send an "error" response back to client. It is used for any response that can be generated by the server from the request record. This includes all 204 (no content), 3xx (redirect), 4xx (client error), and 5xx (server error) messages that have not been redirected to another handler via the ErrorDocument feature.

```
$r->send_error_response($recursive_error);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$recursive_error` (boolean)**

  the 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**
- **since: 2.0.00**

META: it's really an internal Apache method, I'm not quite sure how can it be used externally.

19.4.2 `send_mmap`

META: Autogenerated - needs to be reviewed/completed

Send an MMAP'ed file to the client

```
$ret = $r->send_mmap($mm, $offset, $length);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$mm` (*`APR::Mmap`*)**

The MMAP'ed file to send

- **arg2: `$offset` (number)**

  The offset into the MMAP to start sending

- **arg3: `$length` (integer)**

  The amount of data to send

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

  The number of bytes sent

- **since: 2.0.00**

META: requires a working APR::Mmap, which is not supported at the moment.

19.5 See Also

mod_perl 2.0 documentation.

19.6 Copyright

19.7 Authors

The mod_perl development team and numerous contributors.

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

20.1 Synopsis

```
use Apache2::ServerRec ();

$error_fname = $s->error_fname();

$is_virtual = $s->is_virtual();

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

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

$path = $s->path();

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

$server_admin = $s->server_admin();

$proc = $s->process();

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

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

20.2 Description

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

`Apache2::ServerUtil` provides an extra functionality.

20.3 API

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

20.3.1 `error_fname`

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

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

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

  If passed, sets the new value for `ErrorLog`

  Note the limited functionality under threaded MPMs.

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

  Returns the `ErrorLog` value setting.

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

- **since: 2.0.00**

20.3.2 `is_virtual`

Test whether `$s` is a virtual host object

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

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

  Returns the is_virtual setting.

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

- **since: 2.0.00**

Example:

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

20.3.3 `keep_alive`

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

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

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

If passed, sets the new keep_alive.

Note the limited functionality under threaded MPMs.

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

  Returns the `KeepAlive` setting.

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

- **since: 2.0.00**

20.3.4 keep_alive_max

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

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

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

  If passed, sets the new keep_alive_max.

  Note the limited functionality under threaded MPMs.

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

  Returns the keep_alive_max setting.

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

- **since: 2.0.00**

20.3.5 keep_alive_timeout

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

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

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

  The expected value is in microsecs.

If passed, sets the new `KeepAlive` timeout.

Note the limited functionality under threaded MPMs.

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

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

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

- **since: 2.0.00**

20.3.6 `limit_req_fields`

Get/set limit on number of request header fields

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

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

  If passed, sets the new request headers number limit.

  Note the limited functionality under threaded MPMs.

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

  Returns the request headers number limit.

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

- **since: 2.0.00**

20.3.7 `limit_req_fieldsize`

Get/set limit on size of any request header field

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

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

  If passed, sets the new request header size limit.

  Note the limited functionality under threaded MPMs.

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

  Returns the request header size limit.

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

- **since: 2.0.00**

20.3.8 `limit_req_line`

Get/set limit on size of the HTTP request line

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

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

  If passed, sets the new request line limit value.

  Note the limited functionality under threaded MPMs.

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

  Returns the request line limit value

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

- **since: 2.0.00**

20.3.9 `loglevel`

Get/set the `LogLevel` directive value

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

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

  If passed, sets a new `LogLevel` value

  Note the limited functionality under threaded MPMs.

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

  Returns the `LogLevel` value as a constant.

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

- **since: 2.0.00**

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

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

20.3.10 next

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

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

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

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

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

20.3.11 path

Get/set pathname for the `ServerPath` setting

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

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

  If passed, sets the new path.

  Note the limited functionality under threaded MPMs.

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

  Returns the path setting.

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

- **since: 2.0.00**

20.3.12 `port`

Get/set the port value

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

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

  If passed, sets the new port.

  Note the limited functionality under threaded MPMs.

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

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

  Returns the port setting.

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

- **since: 2.0.00**

20.3.13 `process`

The process this server is running in

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

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

20.3.14 `server_admin`

Get/set the `ServerAdmin` value

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

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

If passed, sets the new `ServerAdmin` value.

Note the limited functionality under threaded MPMs.

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

  Returns the `ServerAdmin` value.

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

- **since: 2.0.00**

20.3.15 `server_hostname`

Get/set the `ServerName` value

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

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

  If passed, sets the `ServerName` value

  Note the limited functionality under threaded MPMs.

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

  Returns the `ServerName` value

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

- **since: 2.0.00**

20.3.16 `timeout`

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

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

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

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

Note the limited functionality under threaded MPMs.

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

  Returns the timeout setting in microseconds.

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

- **since: 2.0.00**

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

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

20.4 Notes

20.4.1 Limited Functionality under Threaded MPMs

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

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

20.5 Unsupported API

`Apache2::ServerRec` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

20.5.1 `addrs`

Get the addrs value

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

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

  Returns the addrs setting.

- **since: subject to change**

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

20.5.2 lookup_defaults

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

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

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

  Returns the lookup_defaults setting.

- **since: subject to change**

20.5.3 module_config

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

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

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

  Returns the module_config setting.

- **since: subject to change**

20.5.4 names

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

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

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

  If passed, sets the new names.

  Note the limited functionality under threaded MPMs.

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

  Returns the names setting.

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

- **since: 2.0.00**

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

20.5.5 `wild_names`

Wildcarded names for ServerAlias servers

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

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

  If passed, sets the new wild_names.

  Note the limited functionality under threaded MPMs.

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

  Returns the wild_names setting.

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

- **since: 2.0.00**

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

20.6 See Also

mod_perl 2.0 documentation.

20.7 Copyright

20.8 Authors

The mod_perl development team and numerous contributors.

21 Apache2::ServerUtil - Perl API for Apache server record utils

21.1 Synopsis

```
use Apache2::ServerUtil ();
$s = Apache2::ServerUtil->server;

# push config
$s->add_config([’ServerTokens off’]);

# add components to the Server signature
$s->add_version_component("MyModule/1.234");

# access PerlSetVar/PerlAddVar values
my $srv_cfg = $s->dir_config;

# check command line defines
print "this is mp2"
    if Apache2::ServerUtil::exists_config_define(’MODPERL2’);

# get PerlChildExitHandler configured handlers
@handlers = @{ $s->get_handlers(’PerlChildExitHandler’) || []};

# server build and version info:
$when_built = Apache2::ServerUtil::get_server_built();
$description = Apache2::ServerUtil::get_server_description();
$version = Apache2::ServerUtil::get_server_version();
$banner = Apache2::ServerUtil::get_server_banner();

# ServerRoot value
$server_root = Apache2::ServerUtil::server_root();

# get ’conf/’ dir path (avoid using this function!)
my $dir = Apache2::ServerUtil::server_root_relative($r->pool, ’conf’);

# set child_exit handlers
$r->set_handlers(PerlChildExitHandler => \&handler);

# server level PerlOptions flags lookup
$s->push_handlers(ChildExit => \&child_exit)
    if $s->is_perl_option_enabled(’ChildExit’);

# extend HTTP to support a new method
$s->method_register(’NEWGET’);


# register server shutdown callback
Apache2::ServerUtil::server_shutdown_register_cleanup(sub { Apache2::Const::OK });

# do something only when the server restarts
my $cnt = Apache2::ServerUtil::restart_count();
do_something_once() if $cnt > 1;

# get the resolved ids from Group and User entries
my $user_id  = Apache2::ServerUtil->user_id;
my $group_id = Apache2::ServerUtil->group_id;
```

21.2 Description

`Apache2::ServerUtil` provides the Apache server object utilities API.

21.3 Methods API

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

21.3.1 `add_config`

Dynamically add Apache configuration:

```
$s->add_config($lines);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **arg1: `$lines` (ARRAY ref)**

  An ARRAY reference containing configuration lines per element, without the new line terminators.

- **ret: no return value**
- **since: 2.0.00**

See also: `$r->add_config`

For example:

Add a configuration section at the server startup (e.g. from *startup.pl*):

```
use Apache2::ServerUtil ();
my $conf = <<'EOC';
PerlModule Apache2::MyExample
<Location /perl>
  SetHandler perl-script
  PerlResponseHandler Apache2::MyExample
</Location>
EOC
Apache2::ServerUtil->server->add_config([split /\n/, $conf]);
```

21.3.2 `add_version_component`

Add a component to the version string

```
$s->add_version_component($component);
```

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

The string component to add

- **ret: no return value**
- **since: 2.0.00**

This function is usually used by modules to advertise themselves to the world. It's picked up by such statistics collectors, like netcraft.com, which accomplish that by connecting to various servers and grabbing the server version response header (`Server`). Some servers choose to fully or partially conceal that header.

This method should be invoked in the `PerlPostConfigHandler` phase, which will ensure that the Apache core version number will appear first.

For example let's add a component *"Hikers, Inc/0.99999"* to the server string at the server startup:

```
use Apache2::ServerUtil ();
use Apache2::Const -compile => 'OK';

Apache2::ServerUtil->server->push_handlers(
    PerlPostConfigHandler => \&add_my_version);

sub add_my_version {
    my ($conf_pool, $log_pool, $temp_pool, $s) = @_;
    $s->add_version_component("Hikers, Inc/0.99999");
    return Apache2::Const::OK;
}
```

or of course you could register the `PerlPostConfigHandler` handler directly in *httpd.conf*

Now when the server starts, you will something like:

```
[Thu Jul 15 12:15:28 2004] [notice] Apache/2.0.51-dev (Unix)
mod_perl/1.99_15-dev Perl/v5.8.5 Hikers, Inc/0.99999
configured -- resuming normal operations
```

Also remember that the `ServerTokens` directive value controls whether the component information is displayed or not.

21.3.3 dir_config

`$s->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 = $s->dir_config->get($key);
$s->dir_config($key, $val);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg2: `$key` (string)**

Key string

- **opt arg3: `$val` (string)**

  Value string

- **ret: ...**

  Depends on the passed arguments, see further discussion

- **since: 2.0.00**

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 = $s->dir_config->get($key);
```

To receive a list of values you must use `get()` method from the `APR::Table` class.

```
$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 assosiated 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.

21.3.4 `exists_config_define`

Check for a definition from the server startup command line (e.g. `-DMODPERL2`)

```
$result = Apache2::ServerUtil::exists_config_define($name);
```

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

The define string to check for

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

  true if defined, false otherwise

- **since: 2.0.00**

For example:

```
print "this is mp2"
    if Apache2::ServerUtil::exists_config_define('MODPERL2');
```

21.3.5 `get_handlers`

Returns a reference to a list of handlers enabled for a given phase.

```
$handlers_list = $s->get_handlers($hook_name);
```

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

  a string representing the phase to handle.

- **ret: `$handlers_list` (ref to an ARRAY of CODE refs)**

  a list of references to the handler subroutines

- **since: 2.0.00**

See also: `$r->add_config`

For example:

A list of handlers configured to run at the *child_exit* phase:

```
@handlers = @{ $s->get_handlers('PerlChildExitHandler') || []};
```

21.3.6 `get_server_built`

Get the date and time that the server was built

```
$when_built = Apache2::ServerUtil::get_server_built();
```

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

  The server build time string

- **since: 2.0.00**

21.3.7 get_server_version

Get the server version string

```
$version = Apache2::ServerUtil::get_server_version();
```

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

  The server version string

- **since: 2.0.00**

21.3.8 get_server_banner

Get the server banner

```
$banner = Apache2::ServerUtil::get_server_banner();
```

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

  The server banner

- **since: 2.0.4**

21.3.9 get_server_description

Get the server description

```
$description = Apache2::ServerUtil::get_server_description();
```

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

  The server description

- **since: 2.0.4**

21.3.10 group_id

Get the group id corresponding to the `Group` directive in *httpd.conf*:

```
$gid = Apache2::ServerUtil->group_id;
```

- **obj: `Apache2::ServerUtil` (class name)**
- **ret: `$gid` (integer)**

  On Unix platforms returns the gid corresponding to the value used in the `Group` directive in *httpd.conf*. On other platforms returns 0.

- **since: 2.0.03**

21.3.11 is_perl_option_enabled

check whether a server level PerlOptions flag is enabled or not.

```
$result = $s->is_perl_option_enabled($flag);
```

- **obj: $s (Apache2::ServerRec object)**
- **arg1: $flag (string)**
- **ret: $result (boolean)**
- **since: 2.0.00**

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.

21.3.12 method_register

Register a new request method, and return the offset that will be associated with that method.

```
$offset = $s->method_register($methname);
```

- **obj: $s (Apache2::ServerRec object)**
- **arg1: $methname (string)**

  The name of the new method to register (in addition to the already supported GET, HEAD, etc.)

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

  An int value representing an offset into a bitmask. You can probably ignore it.

- **since: 2.0.00**

This method allows you to extend the HTTP protocol to support new methods, which fit the HTTP paradigm. Of course you will need to write a client that understands that protocol extension. For a good example, refer to the MyApache2::SendEmail example presented in the PerlHeaderParserHandler section, which demonstrates how a new method EMAIL is registered and used.

21.3.13 push_handlers

Add one or more handlers to a list of handlers to be called for a given phase.

```
$ok = $s->push_handlers($hook_name => \&handler);
$ok = $s->push_handlers($hook_name => [\&handler, \&handler2]);
```

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

  the phase to add the handlers to

- **arg2: `$handlers` (CODE ref or SUB name or an ARRAY ref)**

  a single handler CODE reference or just a name of the subroutine (fully qualified unless defined in the current package).

  if more than one passed, use a reference to an array of CODE refs and/or subroutine names.

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

  returns a true value on success, otherwise a false value

- **since: 2.0.00**

See also: `$r->add_config`

Examples:

A single handler:

```
$s->push_handlers(PerlChildExitHandler => \&handler);
```

Multiple handlers:

```
$s->push_handlers(PerlChildExitHandler => ['Foo::Bar::handler', \&handler2]);
```

Anonymous functions:

```
$s->push_handlers(PerlLogHandler => sub { return Apache2::Const::OK });
```

21.3.14 `restart_count`

How many times the server was restarted.

```
$restart_count = Apache2::ServerUtil::restart_count();
```

- **ret: `restart_count` (number)**
- **since: 2.0.00**

The following demonstration should make it clear what values to expect from this function. Let's add the following code to *startup.pl*, so it's run every time *httpd.conf* is parsed:

```
use Apache2::ServerUtil ();
my $cnt = Apache2::ServerUtil::restart_count();
open my $fh, ">>/tmp/out" or die "$!";
print $fh "cnt: $cnt\n";
close $fh;
```

Now let's run a series of server starts and restarts and look at what is logged into */tmp/out*:

```
% httpd -k start
cnt: 1
cnt: 2

% httpd -k graceful
cnt: 1
cnt: 3

% httpd -k graceful
cnt: 1
cnt: 4

% httpd -k stop
cnt: 1
```

Remembering that Apache restarts itself immediately after starting, we can see that the `restart_count` goes from 1 to 2 during the server start. Moreover we can see that every operation forces the parsing of *httpd.conf* and therefore reinitialization of mod_perl (and running all the code found in *httpd.conf*). This happens even when the server is shutdown via `httpd -k stop`.

What conclusions can be drawn from this demonstration:

- `Apache2::ServerUtil::restart_count()` returns 1 every time some `-k` command is passed to Apache (or `kill -USR1` or some alternative signal is received).
- At all other times the count will be 2 or higher. So for example on graceful restart the count will be 3 or higher.

For example if you want to run something every time `httpd -k` is run you just need to check whether `restart_count()` returns 1:

```
my $cnt = Apache2::ServerUtil::restart_count();
do_something() if $cnt == 1;
```

To do something only when server restarts (`httpd -k start` or `httpd -k graceful`), check whether `restart_count()` is bigger than 1:

```
my $cnt = Apache2::ServerUtil::restart_count();
do_something() if $cnt > 1;
```

21.3.15 server

Get the main server's object

```
$main_s = Apache2::ServerUtil->server();
```

- **obj: `Apache2::ServerUtil` (class name)**
- **ret: `$main_s` (`Apache2::ServerRec object`)**
- **since: 2.0.00**

21.3.16 server_root

returns the value set by the top-level `ServerRoot` directive.

```
$server_root = Apache2::ServerUtil::server_root();
```

- **ret: `$server_root` (string)**
- **since: 2.0.00**

21.3.17 server_root_relative

Returns the canonical form of the filename made absolute to `ServerRoot`:

```
$path = Apache2::ServerUtil::server_root_relative($pool, $fname);
```

- **arg1: `$pool` (`APR::Pool object`)**

  Make sure that you read the following explanation and understand well which pool object you need to pass before using this function.

- **opt arg2: `$fname` (string)**
- **ret: `$path` (string)**

  The concatenation of `ServerRoot` and the `$fname`.

  If `$fname` is not specified, the value of `ServerRoot` is returned with a trailing `/`. (it's the same as using `''` as `$fname`'s value).

- **since: 2.0.00**

`$fname` is appended to the value of `ServerRoot` and returned. For example:

```
my $dir = Apache2::ServerUtil::server_root_relative($r->pool, 'logs');
```

You must be extra-careful when using this function. If you aren't sure what you are doing don't use it.

It's much safer to build the path by yourself using use `Apache2::ServerUtil::server_root()`, For example:

```
use File::Spec::Functions qw(catfile);
my $path = catfile Apache2::ServerUtil::server_root, qw(t logs);
```

In this example, no memory allocation happens on the Apache-side and you aren't risking to get a memory leak.

The problem with `server_root_relative` is that Apache allocates memory to concatenate the path string. The memory is allocated from the pool object. If you call this method on the server pool object it'll allocate the memory from it. If you do that at the server startup, it's perfectly right, since you will do that only once. However if you do that from within a request or a connection handler, you create a memory leak every time it is called -- as the memory gets allocated from the server pool, it will be freed only when the server is shutdown. Therefore if you need to build a relative to the root server path for the duration of the request, use the request pool:

```
use Apache2::RequestRec ();
Apache2::ServerUtil::server_root_relative($r->pool, $fname);
```

If you need to have the path for the duration of a connection (e.g. inside a protocol handler), you should use:

```
use Apache2::Connection ();
Apache2::ServerUtil::server_root_relative($c->pool, $fname);
```

And if you want it for the scope of the server file:

```
use Apache2::Process ();
use Apache2::ServerUtil ();
Apache2::ServerUtil::server_root_relative($s->process->pool, $fname);
```

Moreover, you could have encountered the opposite problem, where you have used a short-lived pool object to construct the path, but tried to use the resulting path variable, when that pool has been destructed already. In order to avoid mysterious segmentation faults, mod_perl does a wasteful copy of the path string when returning it to you -- another reason to avoid using this function.

21.3.18 server_shutdown_cleanup_register

Register server shutdown cleanup callback:

```
Apache2::ServerUtil::server_shutdown_cleanup_register($sub);
```

- **arg1: `$sub` (CODE ref or SUB name)**
- **ret: no return value**
- **since: 2.0.00**

This function can be used to register a callback to be run once at the server shutdown (compared to `PerlChildExitHandler` which will execute the callback for each exiting child process).

For example in order to arrange the function `do_my_cleanups()` to be run every time the server shuts down (or restarts), run the following code at the server startup:

```
Apache2::ServerUtil::server_shutdown_cleanup_register(\&do_my_cleanups);
```

It's necessary to run this code at the server startup (normally *startup.pl*). The function will croak if run after the `PerlPostConfigHandler` phase.

Values returned from cleanup functions are ignored. If a cleanup dies the exception is stringified and passed to `warn()`. Usually, this results in printing it to the *error_log*.

21.3.19 set_handlers

Set a list of handlers to be called for a given phase. Any previously set handlers are forgotten.

```
$ok = $s->set_handlers($hook_name => \&handler);
$ok = $s->set_handlers($hook_name => [\&handler, \&handler2]);
$ok = $s->set_handlers($hook_name => []);
$ok = $s->set_handlers($hook_name => undef);
```

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

  the phase to set the handlers in

- **arg2: `$handlers` (CODE ref or SUB name or an ARRAY ref)**

  a reference to a single handler CODE reference or just a name of the subroutine (fully qualified unless defined in the current package).

  if more than one passed, use a reference to an array of CODE refs and/or subroutine names.

  if the argument is `undef` or `[]` the list of handlers is reset to zero.

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

  returns a true value on success, otherwise a false value

- **since: 2.0.00**

See also: `$r->add_config`

Examples:

A single handler:

```
$r->set_handlers(PerlChildExitHandler => \&handler);
```

Multiple handlers:

```
$r->set_handlers(PerlFixupHandler => ['Foo::Bar::handler', \&handler2]);
```

Anonymous functions:

```
$r->set_handlers(PerlLogHandler => sub { return Apache2::Const::OK });
```

Reset any previously set handlers:

```
$r->set_handlers(PerlCleanupHandler => []);
```

or

```
$r->set_handlers(PerlCleanupHandler => undef);
```

21.3.20 user_id

Get the user id corresponding to the `User` directive in *httpd.conf*:

```
$uid = Apache2::ServerUtil->user_id;
```

- **obj: `Apache2::ServerUtil` (class name)**
- **ret: `$uid` (integer)**

  On Unix platforms returns the uid corresponding to the value used in the `User` directive in *httpd.conf*. On other platforms returns 0.

- **since: 2.0.03**

21.4 Unsupported API

`Apache2::ServerUtil` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

21.4.1 error_log2stderr

Start sending STDERR to the error_log file

```
$s->error_log2stderr();
```

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

  The current server

- **ret: no return value**
- **since: 2.0.00**

This method may prove useful if you want to start redirecting STDERR to the error_log file before Apache does that on the startup.

21.5 See Also

mod_perl 2.0 documentation.

21.6 Copyright

21.7 Authors

The mod_perl development team and numerous contributors.

22 Apache2::SubProcess -- Executing SubProcesses under mod_perl

22.1 Synopsis

```
use Apache2::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 = $r->spawn_proc_prog($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 = $r->spawn_proc_prog($command);
$output = read_data($out_fh);

# write to/read from the process
$command = "/tmp/in_out_err.pl";
($in_fh, $out_fh, $err_fh) = $r->spawn_proc_prog($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 : '';
}

# pass @ARGV but don't ask for any communication channels
$command = "/tmp/argv.pl";
@argv = qw(foo bar);
$r->spawn_proc_prog($command, \@argv);
```

22.2 Description

`Apache2::SubProcess` provides the Perl API for running and communicating with processes spawned from mod_perl handlers.

At the moment it's possible to spawn only external program in a new process. It's possible to provide other interfaces, e.g. executing a sub-routine reference (via `B::Deparse`) and may be spawn a new program in a thread (since the APR api includes API for spawning threads, e.g. that's how it's running mod_cgi on win32).

22.3 API

22.3.1 spawn_proc_prog

Spawn a sub-process and return STD communication pipes:

```
                              $r->spawn_proc_prog($command);
                              $r->spawn_proc_prog($command, \@argv);
$out_fh                     = $r->spawn_proc_prog($command);
$out_fh                     = $r->spawn_proc_prog($command, \@argv);
($in_fh, $out_fh, $err_fh) = $r->spawn_proc_prog($command);
($in_fh, $out_fh, $err_fh) = $r->spawn_proc_prog($command, \@argv);
```

- **obj: $r (Apache2::RequestRec object)**
- **arg1: $command (string)**

  The command to be $exec()'ed.

- **opt arg2: \@argv (ARRAY ref)**

  A reference to an array of arguments to be passed to the process as the process' ARGV.

- **ret: ...**

  In VOID context returns no filehandles (all std streams to the spawned process are closed).

  In SCALAR context returns the output filehandle of the spawned process (the in and err std streams to the spawned process are closed).

  In LIST context returns the input, outpur and error filehandles of the spawned process.

- **since: 2.0.00**

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.

Several examples appear in the Synopsis section.

`spawn_proc_prog()` is similar to `fork()`, but provides you a better framework to communicate with that process and handles the cleanups for you. But that means that just like `fork()` it gives you a different process, so you don't use the current Perl interpreter in that new process. If you try to use that method or fork to run a high-performance parallel processing you should look elsewhere. You could try Perl threads, but they are **very** expensive to start if you have a lot of things loaded into memory (since `perl_clone()` dups almost everything in the perl land, but the opcode tree). In the mod_perl "paradigm" this is much more expensive than fork, since normally most of the time we have lots of perl things loaded into memory. Most likely the best solution here is to offload the job to PPerl or some other daemon, with the only added complexity of communication.

To spawn a completely independent process, which will be able to run after Apache has been shutdown and which won't prevent Apache from restarting (releasing the ports Apache is listening to) call spawn_proc_prog() in a void context and make the script detach and close/reopen its communication streams. For example, spawn a process as:

```
use Apache2::SubProcess ();
$r->spawn_proc_prog ('/path/to/detach_script.pl', $args);
```

and the */path/to/detach_script.pl* contents are:

```
# file:detach_script.pl
#!/usr/bin/perl -w
use strict;
use warnings;

use POSIX 'setsid';

chdir '/'                 or die "Can't chdir to /: $!";
open STDIN, '/dev/null'   or die "Can't read /dev/null: $!";
open STDOUT, '+>>', '/path/to/apache/error_log'
    or die "Can't write to /dev/null: $!";
open STDERR, '>&STDOUT'   or die "Can't dup stdout: $!";
setsid or die "Can't start a new session: $!";

# run your code here or call exec to another program
```

reopening (or closing) the STD streams and called `setsid()` makes sure that the process is now fully detached from Apache and has a life of its own. `chdir()` ensures that no partition is tied, in case you need to remount it.

22.4 See Also

mod_perl 2.0 documentation.

22.5 Copyright

22.6 Authors

The mod_perl development team and numerous contributors.

23 Apache2::SubRequest - Perl API for Apache subrequests

23.1 Synopsis

```
use Apache2::SubRequest ();

# run internal redirects at once
$r->internal_redirect($new_uri);
$r->internal_redirect_handler($new_uri);

# create internal redirect objects
$subr = $r->lookup_uri("/foo");
$subr = $r->lookup_method_uri("GET", "/tmp/bar")
$subr = $r->lookup_file("/tmp/bar");
# optionally manipulate the output through main request filters
$subr = $r->lookup_uri("/foo", $r->output_filters);
# now run them
my $rc = $subr->run;
```

23.2 Description

`Apache2::SubRequest` contains API for creating and running of Apache sub-requests.

`Apache2::SubRequest` is a sub-class of `Apache2::RequestRec object`.

23.3 API

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

23.3.1 DESTROY

Free the memory associated with a sub request:

```
undef $subr; # but normally don't do that
```

- **obj: `$subr` (`Apache2::SubRequest object`)**

  The sub request to finish

- **ret: no return value**
- **since: 2.0.00**

`DESTROY` is called automatically when `$subr` goes out of scope.

If you want to free the memory earlier than that (for example if you run several subrequests), you can `undef` the object as:

```
undef $subr;
```

but never call `DESTROY` explicitly, since it'll result in `ap_destroy_sub_req` being called more than once, resulting in multiple brain injuries and certain hair loss.

23.3.2 internal_redirect

Redirect the current request to some other uri internally

```
$r->internal_redirect($new_uri);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$new_uri` (string)**

  The URI to replace the current request with

- **ret: no return value**
- **since: 2.0.00**

In case that you want some other request to be served as the top-level request instead of what the client requested directly, call this method from a handler, and then immediately return `Apache2::Const::OK`. The client will be unaware the a different request was served to her behind the scenes.

23.3.3 internal_redirect_handler

Identical to `internal_redirect`, plus automatically sets `$r->content_type` is of the sub-request to be the same as of the main request, if `$r->handler` is true.

```
$r->internal_redirect_handler($new_uri);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$new_uri` (string)**

  The URI to replace the current request with.

- **ret: no return value**
- **since: 2.0.00**

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.

23.3.4 lookup_file

Create a subrequest for the given file. This sub request can be inspected to find information about the requested file

```
$ret = $r->lookup_file($new_file);
$ret = $r->lookup_file($new_file, $next_filter);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$new_file` (string)**

  The file to lookup

- **opt arg2: `$next_filter` (`Apache2::Filter`)**

  See `$r->lookup_uri` for details.

- **ret: `$ret` (`Apache2::SubRequest object`)**

  The sub request record.

- **since: 2.0.00**

See `$r->lookup_uri` for further discussion.

23.3.5 lookup_method_uri

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_uri);
$ret = $r->lookup_method_uri($method, $new_uri, $next_filter);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$method` (string)**

  The method to use in the new sub request (e.g. `"GET"`)

- **arg2: `$new_uri` (string)**

  The URI to lookup

- **opt arg3: `$next_filter` (`Apache2::Filter object`)**

  See `$r->lookup_uri` for details.

- **ret: `$ret` (`Apache2::SubRequest object`)**

  The sub request record.

- **since: 2.0.00**

See `$r->lookup_uri` for further discussion.

23.3.6 `lookup_uri`

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_uri);
$ret = $r->lookup_uri($new_uri, $next_filter);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$new_uri` (string)**

  The URI to lookup

- **opt arg2: `$next_filter` (`Apache2::Filter object`)**

  The first filter the subrequest should pass the data through. If not specified it defaults to the first connection output filter for the main request `$r->proto_output_filters`. So if the subrequest sends any output it will be filtered only once. If for example you desire to apply the main request's output filters to the sub-request output as well pass `$r->output_filters` as an argument.

- **ret: `$ret` (`Apache2::SubRequest object`)**

  The sub request record

- **since: 2.0.00**

Here is an example of a simple subrequest which serves uri */new_uri*:

```
sub handler {
    my $r = shift;

    my $subr = $r->lookup_uri("/new_uri");
    $subr->run;

    return Apache2::Const::OK;
}
```

If let’s say you have three request output filters registered to run for the main request:

```
PerlOutputFilterHandler MyApache2::SubReqExample::filterA
PerlOutputFilterHandler MyApache2::SubReqExample::filterB
PerlOutputFilterHandler MyApache2::SubReqExample::filterC
```

and you wish to run them all, the code needs to become:

```
my $subr = $r->lookup_uri("/new_uri", $r->output_filters);
```

and if you wish to run them all, but the first one (`filterA`), the code needs to be adjusted to be:

```
my $subr = $r->lookup_uri("/new_uri", $r->output_filters->next);
```

23.3.7 `run`

Run a sub-request

```
$rc = $subr->run();
```

- **obj: `$subr` (`Apache2::RequestRec object`)**

  The sub-request (e.g. returned by `lookup_uri`)

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

  The return code of the handler (`Apache2::Const::OK`, `Apache2::Const::DECLINED`, etc.)

- **since: 2.0.00**

23.4 Unsupported API

`Apache2::SubRequest` also provides auto-generated Perl interface for a few other methods which aren’t tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

23.4.1 `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);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

The current request

- **arg1: `$sub_req` (string)**

  A subrequest created from this request

- **ret: no return value**
- **since: 2.0.00**

META: httpd-2.0/modules/http/http_request.c declares this function as:

```
/* XXX: Is this function is so bogus and fragile that we deep-6 it? */
```

do we really want to expose it to mod_perl users?

23.4.2 `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);
$lr = $r->lookup_dirent($finfo, $subtype);
$lr = $r->lookup_dirent($finfo, $subtype, $next_filter);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request

- **arg1: `$finfo` (`APR::Finfo object`)**

  The apr_dir_read result to lookup

- **arg2: `$subtype` (integer)**

  What type of subrequest to perform, one of;

  ```
  Apache2::SUBREQ_NO_ARGS     ignore r->args and r->path_info
  Apache2::SUBREQ_MERGE_ARGS  merge  r->args and r->path_info
  ```

- **arg3: `$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` (`Apache2::RequestRec object`)**

  The new request record

- **since: 2.0.00**

META: where do we take the apr_dir_read result from?

23.5 See Also

mod_perl 2.0 documentation.

23.6 Copyright

23.7 Authors

The mod_perl development team and numerous contributors.

24 Apache2::URI - Perl API for manipulating URIs

24.1 Synopsis

```
use Apache2::URI ();

$hostport = $r->construct_server();
$hostport = $r->construct_server($hostname);
$hostport = $r->construct_server($hostname, $port);
$hostport = $r->construct_server($hostname, $port, $pool);

$url = $r->construct_url();
$url = $r->construct_url($rel_uri);
$url = $r->construct_url($rel_uri, $pool);

$parsed_uri = $r->parse_uri($uri);

$parsed_uri = $r->parsed_uri();

$url = join '%20', qw(one two three);
Apache2::URI::unescape_url($url);
```

24.2 Description

While `APR::URI` provides a generic API to dissect, adjust and put together any given URI string, `Apache2::URI` provides an API specific to Apache, by taking the information directly from the `$r` object. Therefore when manipulating the URI of the current HTTP request usually methods from both classes are used.

24.3 API

`Apache2::URI` provides the following functions and methods:

24.3.1 `construct_server`

Construct a string made of hostname and port

```
$hostport = $r->construct_server();
$hostport = $r->construct_server($hostname);
$hostport = $r->construct_server($hostname, $port);
$hostport = $r->construct_server($hostname, $port, $pool);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request object

- **opt arg1: `$hostname` (string)**

  The hostname of the server.

If that argument is not passed, `$r->get_server_name` is used.

- **opt arg2: `$port` (string)**

  The port the server is running on.

  If that argument is not passed, `$r->get_server_port` is used.

- **opt arg3: `$pool` (`APR::Pool object`)**

  The pool to allocate the string from.

  If that argument is not passed, `$r->pool` is used.

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

  The server's hostport string

- **since: 2.0.00**

Examples:

- Assuming that:

  ```
  $r->get_server_name == "localhost";
  $r->get_server_port == 8001;
  ```

  The code:

  ```
  $hostport = $r->construct_server();
  ```

  returns a string:

  ```
  localhost:8001
  ```

- The following code sets the values explicitly:

  ```
  $hostport = $r->construct_server("my.example.com", 8888);
  ```

  and it returns a string:

  ```
  my.example.com:8888
  ```

24.3.2 `construct_url`

Build a fully qualified URL from the uri and information in the request rec:

```
$url = $r->construct_url();
$url = $r->construct_url($rel_uri);
$url = $r->construct_url($rel_uri, $pool);
```

- **obj: `$r` (`Apache2::RequestRec object`)**

  The current request object

- **opt arg1: `$rel_uri` (string)**

  The path to the requested file (it may include a concatenation of *path*, *query* and *fragment* components).

  If that argument is not passed, `$r->uri` is used.

- **opt arg2: `$pool` (`APR::Pool object`)**

  The pool to allocate the URL from

  If that argument is not passed, `$r->pool` is used.

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

  A fully qualified URL

- **since: 2.0.00**

Examples:

- Assuming that the request was

  ```
  http://localhost.localdomain:8529/test?args
  ```

  The code:

  ```
  my $url = $r->construct_url;
  ```

  returns the string:

  ```
  http://localhost.localdomain:8529/test
  ```

  notice that the query (args) component is not in the string. You need to append it manually if it's needed.

- Assuming that the request was

  ```
  http://localhost.localdomain:8529/test?args
  ```

  The code:

  ```
  my $rel_uri = "/foo/bar?tar";
  my $url = $r->construct_url($rel_uri);
  ```

returns the string:

```
http://localhost.localdomain:8529/foo/bar?tar
```

24.3.3 parse_uri

Break apart URI (affecting the current request's uri components)

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

- **obj: $r (Apache2::RequestRec object)**

  The current request object

- **arg1: $uri (string)**

  The uri to break apart

- **ret: no return value**
- **warning:**

  This method has several side-effects explained below

- **since: 2.0.00**

This method call has the following side-effects:

1. sets `$r->args` to the rest after `'?'` if such exists in the passed `$uri`, otherwise sets it to `undef`.
2. sets `$r->uri` to the passed `$uri` without the `$r->args` part.
3. sets `$r->hostname` (if not set already) using the (`scheme://host:port`) parts of the passed `$uri`.

24.3.4 parsed_uri

Get the current request's parsed uri object

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

- **obj: $r (Apache2::RequestRec object)**

  The current request object

- **ret: $uri (APR::URI object)**

  The parsed uri

- **since: 2.0.00**

  This object is suitable for using with `APR::URI::rpath`

24.3.5 `unescape_url`

Unescape URLs

```
Apache2::URI::unescape_url($url);
```

- **obj: `$url` (string)**

  The URL to unescape

- **ret: no return value**

  The argument `$url` is now unescaped

- **since: 2.0.00**

Example:

```
my $url = join '%20', qw(one two three);
Apache2::URI::unescape_url($url);
```

`$url` now contains the string:

```
"one two three";
```

24.4 See Also

`APR::URI`, mod_perl 2.0 documentation.

24.5 Copyright

24.6 Authors

The mod_perl development team and numerous contributors.

25 Apache2::Util - Perl API for Misc Apache Utility functions

25.1 Synopsis

```
use Apache2::Util ();

# OS escape path
$escaped_path = Apache2::Util::escape_path($path, "a 'long' file.html");

# format time as a string
my $fmt = "%a, %D %H:%M:%S %Z";
$fmtdate = Apache2::Util::ht_time($r->pool, $r->request_time, $fmt, 0);
```

25.2 Description

Various Apache utilities that don't fit into any other group.

25.3 Functions API

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

25.3.1 `escape_path`

convert an OS path to a URL in an OS dependant way.

```
$escaped_path = Apache2::Util::escape_path($path, $p);
$escaped_path = Apache2::Util::escape_path($path, $p, $partial);
```

- **arg1: `$path` (string)**

  The path to convert

- **arg2: `$p` (`APR::Pool`)**

  The pool to allocate from

- **opt arg3: `$partial` (boolean)**

  if TRUE, assume that the path will be appended to something with a '/' in it (and thus does not prefix "./")

  if FALSE it prepends `"./"` unless `$path` contains `:` optionally followed by `/`.

  the default is TRUE

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

  The escaped path

- **since: 2.0.00**

25.3.2 `ht_time`

Convert time from an integer value into a string in a specified format

```
$time_str = Apache2::Util::ht_time($p);
$time_str = Apache2::Util::ht_time($p, $time);
$time_str = Apache2::Util::ht_time($p, $time, $fmt);
$time_str = Apache2::Util::ht_time($p, $time, $fmt, $gmt);
```

- **arg1: `$p` (`APR::Pool object`)**

  The pool to allocate memory from

- **opt arg2: `$time` (number)**

  The time to convert (e.g., `time()` or `$r->request_time`).

  If the value is not passed the current time will be used.

- **opt arg3: `$fmt` (string)**

  The format to use for the conversion, using strftime(3) tokens.

  If the value is not passed the default format used is:

  ```
  "%a, %d %b %Y %H:%M:%S %Z"
  ```

- **opt arg4: `$gmt` (boolean)**

  The time will be not converted to GMT if FALSE is passed.

  If the value is not passed TRUE (do convert) is used as a default.

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

  The string that represents the specified time

- **since: 2.0.00**

Examples:

Use current time, the default format and convert to GMT:

```
$fmtdate = Apache2::Util::ht_time($r->pool);
```

Use my time, the default format and convert to GMT:

```
my $time = time+100;
$fmtdate = Apache2::Util::ht_time($r->pool, $time);
```

Use the time the request has started, custom format and don't convert to GMT:

```
my $fmt = "%a, %D %H:%M:%S %Z";
$fmtdate = Apache2::Util::ht_time($r->pool, $r->request_time, $fmt, 0);
```

25.4 See Also

mod_perl 2.0 documentation.

25.5 Copyright

25.6 Authors

The mod_perl development team and numerous contributors.

26 APR - Perl Interface for Apache Portable Runtime (libapr and libaprutil Libraries)

26.1 Synopsis

```
use APR ();
```

26.2 Description

On load this modules prepares the APR enviroment (initializes memory pools, data structures, etc.)

You don’t need to use this module explicitly, since it’s already loaded internally by all `APR::*` modules.

26.3 Using APR modules outside mod_perl 2.0

You’d use the `APR::*` modules outside mod_perl 2.0, just like you’d use it with mod_perl 2.0. For example to get a random unique string you could call:

```
% perl -MAPR::UUID -le 'print APR::UUID->new->format'
```

26.4 See Also

mod_perl 2.0 documentation.

26.5 Copyright

26.6 Authors

The mod_perl development team and numerous contributors.

27 APR::Base64 - Perl API for APR base64 encoding/decoding functionality

27.1 Synopsis

```
use APR::Base64 ();

my $clear   = "foo"
my $encoded = APR::Base64::encode($clear);
my $decoded = APR::Base64::decode($encoded);
my $len_enc = APR::Base64::encode_len(length $clear);
```

27.2 Description

APR::Base64 provides the access to APR's base64 encoding and decoding API.

27.3 API

APR::Base64 provides the following functions and/or methods:

27.3.1 decode

Decode a base64 encoded string

```
$decoded = decode($encoded);
```

- **arg1: $encoded (string)**

  The encoded string.

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

  The decoded string.

- **since: 2.0.00**

27.3.2 encode

Encode a string to base64

```
$encoded = encode($clear);
```

- **arg1: $clear (string)**

  The unencoded string.

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

  The encoded string.

- **since: 2.0.00**

27.3.3 `encode_len`

Given the length of an unencoded string, get the length of the encoded string.

```
$len_enc = encode_len($len_clear);
```

- **arg1: `$len_clear` (integer)**

  the length of an unencoded string.

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

  the length of the string after it is encoded

- **since: 2.0.00**

27.4 See Also

mod_perl 2.0 documentation.

27.5 Copyright

27.6 Authors

The mod_perl development team and numerous contributors.

28 APR::Brigade - Perl API for manipulating APR Bucket Brigades

28.1 Synopsis

```
use APR::Brigade ();

$bb = APR::Brigade->new($r->pool, $c->bucket_alloc);
$ba = $bb->bucket_alloc();
$pool = $bb->pool;

$bb->insert_head($b);
$bb->insert_tail($b);

$b_first = $bb->first;
$b_last  = $bb->last;

$b_prev = $bb->prev($b_last);
$b_next = $bb->next($b);

$bb2 = APR::Brigade->new($r->pool, $c->bucket_alloc);
$bb1->concat($bb2);

$len = $bb->flatten($data);
$len = $bb2->flatten($data, $wanted);

$len = $bb->length;
$bb3 = $bb->split($b_last);

last if $bb->is_empty();
$bb->cleanup();
$bb->destroy();
```

28.2 Description

`APR::Brigade` allows you to create, manipulate and delete APR bucket brigades.

28.3 API

`APR::Brigade` provides the following functions and/or methods:

28.3.1 `cleanup`

Empty out an entire bucket brigade:

```
$bb->cleanup;
```

- **obj: `$bb` (`APR::Brigade object`)**

  The brigade to cleanup

- **ret: no return value**
- **since: 2.0.00**

This method destroys all of the buckets within the bucket brigade's bucket list. This is similar to `destroy()`, except that it does not deregister the brigade's `pool()` cleanup function.

Generally, you should use `destroy()`. This function can be useful in situations where you have a single brigade that you wish to reuse many times by destroying all of the buckets in the brigade and putting new buckets into it later.

28.3.2 concat

Concatenate brigade `$bb2` onto the end of brigade `$bb1`, leaving brigade `$bb2` empty:

```
$bb1->concat($bb2);
```

- **obj: `$bb1` (`APR::Brigade object`)**

  The brigade to concatenate to.

- **arg1: `$bb2` (`APR::Brigade object`)**

  The brigade to concatenate and empty afterwards.

- **ret: no return value**
- **since: 2.0.00**

28.3.3 destroy

destroy an entire bucket brigade, includes all of the buckets within the bucket brigade's bucket list.

```
$bb->destroy();
```

- **obj: `$bb` (`APR::Brigade object`)**

  The bucket brigade to destroy.

- **ret: no return value**
- **excpt: `APR::Error`**
- **since: 2.0.00**

28.3.4 is_empty

Test whether the bucket brigade is empty

```
$ret = $bb->is_empty();
```

- **obj: `$bb` (`APR::Brigade object`)**
- **ret: `$ret` (boolean)**
- **since: 2.0.00**

28.3.5 first

Return the first bucket in a brigade

```
$b_first = $bb->first;
```

- **obj: $bb (APR::Brigade object)**
- **ret: $b_first (APR::Bucket object)**

  The first bucket in the bucket brigade $bb.

  undef is returned if there are no buckets in $bb.

- **since: 2.0.00**

28.3.6 flatten

Get the data from buckets in the bucket brigade as one string

```
$len = $bb->flatten($buffer);
$len = $bb->flatten($buffer, $wanted);
```

- **obj: $bb (APR::Brigade object)**
- **arg1: $buffer (SCALAR)**

  The buffer to fill. All previous data will be lost.

- **opt arg2: $wanted (number)**

  If no argument is passed then all data will be returned. If $wanted is specified -- that number or less bytes will be returned.

- **ret: $len (number)**

  How many bytes were actually read.

  $buffer gets populated with the string that is read. It will contain an empty string if there was nothing to read.

- **since: 2.0.00**
- **excpt: APR::Error**

28.3.7 insert_head

Insert a list of buckets at the front of a brigade

```
$bb->insert_head($b);
```

- **obj: `$bb` (`APR::Brigade object`)**

  Brigade to insert into

- **arg1: `$b` (`APR::Bucket object`)**

  the bucket to insert. More buckets could be attached to that bucket.

- **ret: no return value**
- **since: 2.0.00**

28.3.8 `insert_tail`

Insert a list of buckets at the end of a brigade

```
$bb->insert_tail($b);
```

- **obj: `$bb` (`APR::Brigade object`)**

  Brigade to insert into

- **arg1: `$b` (`APR::Bucket object`)**

  the bucket to insert. More buckets could be attached to that bucket.

- **ret: no return value**
- **since: 2.0.00**

28.3.9 `last`

Return the last bucket in the brigade

```
$b_last = $bb->last;
```

- **obj: `$bb` (`APR::Brigade object`)**
- **ret: `$b_last` (`APR::Bucket object`)**

  The last bucket in the bucket brigade `$bb`.

  `undef` is returned if there are no buckets in `$bb`.

- **since: 2.0.00**

28.3.10 length

Return the total length of the data in the brigade (not the number of buckets)

```
$len = $bb->length;
```

- **obj: $bb (APR::Brigade object)**
- **ret: $len (number)**
- **since: 2.0.00**

28.3.11 new

```
my $nbb = APR::Brigade->new($p, $bucket_alloc);
my $nbb =          $bb->new($p, $bucket_alloc);
```

- **obj: $bb (APR::Brigade object or class)**
- **arg1: $p (APR::Pool object)**
- **arg2: $bucket_alloc (APR::BucketAlloc object)**
- **ret: $nbb (APR::Brigade object)**

  a newly created bucket brigade object

- **since: 2.0.00**

Example:

Create a new bucket brigade, using the request object's pool:

```
use Apache2::Connection ();
use Apache2::RequestRec ();
use APR::Brigade ();
my $bb = APR::Brigade->new($r->pool, $r->connection->bucket_alloc);
```

28.3.12 bucket_alloc

Get the bucket allocator associated with this brigade.

```
my $ba = $bb->bucket_alloc();
```

- **obj: $bb (APR::Brigade object or class)**
- **ret: $ba (APR::BucketAlloc object)**
- **since: 2.0.00**

28.3.13 next

Return the next bucket in a brigade

```
$b_next = $bb->next($b);
```

- **obj: $bb (APR::Brigade object)**
- **arg1: $b (APR::Bucket object)**

  The bucket after which the next bucket $b_next is located

- **ret: $b_next (APR::Bucket object)**

  The next bucket after bucket $b.

  undef is returned if there is no next bucket (i.e. $b is the last bucket).

- **since: 2.0.00**

28.3.14 pool

The pool the brigade is associated with.

```
$pool = $bb->pool;
```

- **obj: $bb (APR::Brigade object)**
- **ret: $pool (APR::Pool object)**
- **since: 2.0.00**

The data is not allocated out of the pool, but a cleanup is registered with this pool. If the brigade is destroyed by some mechanism other than pool destruction, the destroying function is responsible for killing the registered cleanup.

28.3.15 prev

Return the previous bucket in the brigade

```
$b_prev = $bb->prev($b);
```

- **obj: $bb (APR::Brigade object)**
- **arg1: $b (APR::Bucket object)**

  The bucket located after bucket $b_prev

- **ret: $b_prev (APR::Bucket object)**

  The bucket located before bucket $b.

  undef is returned if there is no previous bucket (i.e. $b is the first bucket).

- **since: 2.0.00**

28.3.16 `split`

Split a bucket brigade into two, such that the given bucket is the first in the new bucket brigade.

```
$bb2 = $bb->split($b);
```

- **obj: `$bb` (`APR::Brigade object`)**

  The brigade to split

- **arg1: `$b` (`APR::Bucket object`)**

  The first bucket of the new brigade

- **ret: `$bb2` (`APR::Brigade object`)**

  The new brigade.

- **since: 2.0.00**

This function is useful when a filter wants to pass only the initial part of a brigade to the next filter.

Example:

Create a bucket brigade with three buckets, and split it into two brigade such that the second brigade will have the last two buckets.

```
my $bb1 = APR::Brigade->new($r->pool, $c->bucket_alloc);
my $ba  = $c->bucket_alloc();
$bb1->insert_tail(APR::Bucket->new($ba, "1"));
$bb1->insert_tail(APR::Bucket->new($ba, "2"));
$bb1->insert_tail(APR::Bucket->new($ba, "3"));
```

`$bb1` now contains buckets "1", "2", "3". Now do the split at the second bucket:

```
my $b = $bb1->first; # 1
$b = $bb1->next($b); # 2
my $bb2 = $bb1->split($b);
```

Now `$bb1` contains bucket "1". `$bb2` contains buckets: "2", "3"

28.4 See Also

mod_perl 2.0 documentation.

28.5 Copyright

28.6 Authors

The mod_perl development team and numerous contributors.

29 APR::Bucket - Perl API for manipulating APR Buckets

29.1 Synopsis

```
use APR::Bucket ();
my $ba = $c->bucket_alloc;

$b1 = APR::Bucket->new($ba, "aaa");
$b2 = APR::Bucket::eos_create($ba);
$b3 = APR::Bucket::flush_create($ba);

$b2->is_eos;
$b3->is_flush;

$len = $b1->length;
$len = $b1->read($data);
$type = $b1->type;

$b1->insert_after($b2);
$b1->insert_before($b3);
$b1->remove;
$b1->destroy;

$b2->delete; # remove+destroy

$b4 = APR::Bucket->new($ba, "to be setaside");
$b4->setaside($pool);
```

29.2 Description

`APR::Bucket` allows you to create, manipulate and delete APR buckets.

You will probably find the various insert methods confusing, the tip is to read the function right to left. The following code sample helps to visualize the operations:

```
my $bb = APR::Brigade->new($r->pool, $ba);
my $d1 = APR::Bucket->new($ba, "d1");
my $d2 = APR::Bucket->new($ba, "d2");
my $f1 = APR::Bucket::flush_create($ba);
my $f2 = APR::Bucket::flush_create($ba);
my $e1 = APR::Bucket::eos_create($ba);
                        # head->tail
$bb->insert_head(  $d1); # head->d1->tail
$d1->insert_after( $d2); # head->d1->d2->tail
$d2->insert_before($f1); # head->d1->f1->d2->tail
$d2->insert_after( $f2); # head->d1->f1->d2->f2->tail
$bb->insert_tail(  $e1); # head->d1->f1->d2->f2->e1->tail
```

29.3 API

`APR::Bucket` provides the following functions and/or methods:

29.3.1 delete

Tell the bucket to remove itself from the bucket brigade it belongs to, and destroy itself.

```
$bucket->delete();
```

- **obj: $bucket (APR::Bucket object)**
- **ret: no return value**
- **since: 2.0.00**

If the bucket is not attached to any bucket brigade then this operation just destroys the bucket.

`delete` is a convenience wrapper, internally doing:

```
$b->remove;
$b->destroy;
```

Examples:

Assuming that `$bb` already exists and filled with buckets, replace the existing data buckets with new buckets with upcased data;

```
for (my $b = $bb->first; $b; $b = $bb->next($b)) {
   if ($b->read(my $data)) {
        my $nb = APR::Bucket->new($bb->bucket_alloc, uc $data);
        $b->insert_before($nb);
        $b->delete;
        $b = $nb;
    }
}
```

29.3.2 destroy

Free the resources used by a bucket. If multiple buckets refer to the same resource it is freed when the last one goes away.

```
$bucket->destroy();
```

- **obj: $bucket (APR::Bucket object)**
- **ret: no return value**
- **since: 2.0.00**

A bucket needs to be destroyed if it was removed from a bucket brigade, to avoid memory leak.

If a bucket is linked to a bucket brigade, it needs to be removed from it, before it can be destroyed.

Usually instead of calling:

```
$b->remove;
$b->destroy;
```

it's better to call `delete` which does exactly that.

29.3.3 `eos_create`

Create an *EndOfStream* bucket.

```
$b = APR::Bucket::eos_create($ba);
```

- **arg1: `$ba` (`APR::BucketAlloc object`)**

  The freelist from which this bucket should be allocated

- **ret: `$b` (`APR::Bucket object`)**

  The new bucket

- **since: 2.0.00**

This bucket type indicates that there is no more data coming from down the filter stack. All filters should flush any buffered data at this point.

Example:

```
use APR::Bucket ();
use Apache2::Connection ();
my $ba = $c->bucket_alloc;
my $eos_b = APR::Bucket::eos_create($ba);
```

29.3.4 `flush_create`

Create a flush bucket.

```
$b = APR::Bucket::flush_create($ba);
```

- **arg1: `$ba` (`APR::BucketAlloc object`)**

  The freelist from which this bucket should be allocated

- **ret: `$b` (`APR::Bucket object`)**

  The new bucket

- **since: 2.0.00**

This bucket type indicates that filters should flush their data. There is no guarantee that they will flush it, but this is the best we can do.

29.3.5 insert_after

Insert a list of buckets after a specified bucket

```
$after_bucket->insert_after($add_bucket);
```

- **obj: $after_bucket (APR::Bucket object)**

  The bucket to insert after

- **arg1: $add_bucket (APR::Bucket object)**

  The buckets to insert. It says buckets, since $add_bucket may have more buckets attached after itself.

- **ret: no return value**
- **since: 2.0.00**

29.3.6 insert_before

Insert a list of buckets before a specified bucket

```
$before_bucket->insert_before($add_bucket);
```

- **obj: $before_bucket (APR::Bucket object)**

  The bucket to insert before

- **arg1: $add_bucket (APR::Bucket object)**

  The buckets to insert. It says buckets, since $add_bucket may have more buckets attached after itself.

- **ret: no return value**
- **since: 2.0.00**

29.3.7 is_eos

Determine if a bucket is an EOS bucket

```
$ret = $bucket->is_eos();
```

- **obj: $bucket (APR::Bucket object)**
- **ret: $ret (boolean)**
- **since: 2.0.00**

29.3.8 is_flush

Determine if a bucket is a FLUSH bucket

```
$ret = $bucket->is_flush();
```

- **obj: $bucket (APR::Bucket object)**
- **ret: $ret (boolean)**
- **since: 2.0.00**

29.3.9 length

Get the length of the data in the bucket.

```
$len = $b->length;
```

- **obj: $b (APR::Bucket object)**
- **ret: $len (integer)**

  If the length is unknown, $len value will be -1.

- **since: 2.0.00**

29.3.10 new

Create a new bucket and initialize it with data:

```
$nb = APR::Bucket->new($ba, $data);
$nb =          $b->new($ba, $data);
$nb = APR::Bucket->new($ba, $data, $offset);
$nb = APR::Bucket->new($ba, $data, $offset, $len);
```

- **obj: $b (APR::Bucket object or class)**
- **arg1: $ba (APR::BucketAlloc object)**
- **arg2: $data (string)**

  The data to initialize with.

  Important: in order to avoid unnecessary data copying the variable is stored in the bucket object. That means that if you modify $data after passing it to new() you will modify the data in the bucket as well. To avoid that pass to new() a copy which you won't modify.

- **opt arg3: $offset (number)**

  Optional offset inside $data. Default: 0.

- **opt arg4: $len (number)**

Optional partial length to read.

If `$offset` is specified, then:

```
length $buffer - $offset;
```

will be used. Otherwise the default is to use:

```
length $buffer;
```

- **ret: `$nb` (`APR::Bucket object`)**

  a newly created bucket object

- **since: 2.0.00**

Examples:

- Create a new bucket using a whole string:

  ```
  use APR::Bucket ();
  my $data = "my data";
  my $b = APR::Bucket->new($ba, $data);
  ```

  now the bucket contains the string *'my data'*.

- Create a new bucket using a sub-string:

  ```
  use APR::Bucket ();
  my $data   = "my data";
  my $offset = 3;
  my $b = APR::Bucket->new($ba, $data, $offset);
  ```

  now the bucket contains the string *'data'*.

- Create a new bucket not using the whole length and starting from an offset:

  ```
  use APR::Bucket ();
  my $data   = "my data";
  my $offset = 3;
  my $len    = 3;
  my $b = APR::Bucket->new($ba, $data, $offset, $len);
  ```

  now the bucket contains the string *'dat'*.

29.3.11 `read`

Read the data from the bucket.

```
$len = $b->read($buffer);
$len = $b->read($buffer, $block);
```

- **obj: `$b` (`APR::Bucket object`)**

  The bucket to read from

- **arg1: `$buffer` (SCALAR)**

  The buffer to fill. All previous data will be lost.

- **opt arg2: `$block` (`APR::Const :read_type constant`)**

  optional reading mode constant.

  By default the read is blocking, via `APR::Const::BLOCK_READ constant`.

- **ret: `$len` (number)**

  How many bytes were actually read

  `$buffer` gets populated with the string that is read. It will contain an empty string if there was nothing to read.

- **since: 2.0.00**
- **excpt: `APR::Error`**

It's important to know that certain bucket types (e.g. file bucket), may perform a split and insert extra buckets following the current one. Therefore never call `$b->remove`, before calling `$b->read`, or you may lose data.

Examples:

Blocking read:

```
my $len = $b->read(my $buffer);
```

Non-blocking read:

```
use APR::Const -compile 'NONBLOCK_READ';
my $len = $b->read(my $buffer, APR::Const::NONBLOCK_READ);
```

29.3.12 `remove`

Tell the bucket to remove itself from the bucket brigade it belongs to.

```
$bucket->remove();
```

- **obj: `$bucket` (`APR::Bucket object`)**
- **ret: no return value**
- **since: 2.0.00**

If the bucket is not attached to any bucket brigade then this operation doesn't do anything.

When the bucket is removed, it's not not destroyed. Usually this is done in order to move the bucket to another bucket brigade. Or to copy the data way before destroying the bucket. If the bucket wasn't moved to another bucket brigade it must be destroyed.

Examples:

Assuming that `$bb1` already exists and filled with buckets, move every odd bucket number to `$bb2` and every even to `$bb3`:

```
my $bb2 = APR::Brigade->new($c->pool, $c->bucket_alloc);
my $bb3 = APR::Brigade->new($c->pool, $c->bucket_alloc);
my $count = 0;
while (my $bucket = $bb->first) {
    $count++;
    $bucket->remove;
    $count % 2
        ? $bb2->insert_tail($bucket)
        : $bb3->insert_tail($bucket);
}
```

29.3.13 `setaside`

Ensure the bucket's data lasts at least as long as the given pool:

```
my $status = $b->setaside($pool);
```

- **obj: `$b` (`APR::Bucket object`)**
- **arg1: `$pool` (`APR::Pool object`)**
- **ret: (`APR::Const status constant`)**

  On success, `APR::Const::SUCCESS` is returned. Otherwise a failure code is returned.

- **excpt: `APR::Error`**

  when your code deals only with mod_perl buckets, you don't have to ask for the return value. If this method is called in the `VOID` context, i.e.:

  ```
  $b->setaside($pool);
  ```

  mod_perl will do the error checking on your behalf, and if the return code is not `APR::Const::SUCCESS`, an `APR::Error exception` will be thrown.

  However if your code doesn't know which bucket types it may need to setaside, you may want to check the return code and deal with any errors. For example one of the possible error codes is `APR::Const::ENOTIMPL`. As of this writing the pipe and socket buckets can't `setaside()`, in which case you may want to look at the `ap_save_brigade()` implementation.

- **since: 2.0.00**

Usually setaside is called by certain output filters, in order to buffer socket writes of smaller buckets into a single write. This method works on all bucket types (not only the mod_perl bucket type), but as explained in the exceptions section, not all bucket types implement this method.

When a mod_perl bucket is setaside, its data is detached from the original perl scalar and copied into a pool bucket. That allows downstream filters to deal with the data originally owned by a Perl interpreter, making it possible for that interpreter to go away and do other things, or be destroyed.

29.3.14 `type`

Get the type of the data in the bucket.

```
$type = $b->type;
```

- **obj: `$b` (`APR::Bucket object`)**
- **ret: `$type` (`APR::BucketType object`)**
- **since: 2.0.00**

You need to invoke `APR::BucketType` methods to access the data.

Example:

Create a flush bucket and read its type's name:

```
use APR::Bucket ();
use APR::BucketType ();
my $b = APR::Bucket::flush_create($ba);
my $type = $b->type;
my $type_name =  $type->name; # FLUSH
```

The type name will be *'FLUSH'* in this example.

29.4 Unsupported API

`APR::Socket` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

29.4.1 `data`

```
$data = $b->data;
```

Gives a C pointer to the address of the data in the bucket. I can’t see what use can be done of it in Perl.

- **obj: `$b` (`APR::Bucket object`)**
- **ret: `$data` (C pointer)**
- **since: subject to change**

29.4.2 `start`

```
$start = $b->start;
```

It gives the offset to when a new bucket is created with a non-zero offset value:

```
my $b = APR::Bucket->new($ba, $data, $offset, $len);
```

So if the offset was 3. `$start` will be 3 too.

I fail to see what it can be useful for to the end user (it’s mainly used internally).

- **obj: `$b` (`APR::Bucket object`)**
- **ret: `$start` (offset number)**
- **since: subject to change**

29.5 See Also

mod_perl 2.0 documentation.

29.6 Copyright

29.7 Authors

The mod_perl development team and numerous contributors.

30 APR::BucketAlloc - Perl API for Bucket Allocation

30.1 Synopsis

```
use APR::BucketAlloc ();
$ba = APR::BucketAlloc->new($pool);
$ba->destroy;
```

30.2 Description

APR::BucketAlloc is used for bucket allocation.

30.2.1 new

Create an APR::BucketAlloc object:

```
$ba = APR::BucketAlloc->new($pool);
```

- **class: APR::BucketAlloc**
- **arg1: $pool (APR::Pool object)**

  The pool used to create this object.

- **ret: $ba (APR::BucketAlloc object)**

  The new object.

- **since: 2.0.00**

This bucket allocation list (freelist) is used to create new buckets (via APR::Bucket->new) and bucket brigades (via APR::Brigade->new).

You only need to use this method if you aren't running under httpd. If you are running under mod_perl, you already have a bucket allocation available via $c->bucket_alloc and $bb->bucket_alloc.

Example:

```
use APR::BucketAlloc ();
use APR::Pool ();
my $ba = APR::BucketAlloc->(APR::Pool->pool);
my $eos_b = APR::Bucket::eos_create($ba);
```

30.2.2 destroy

Destroy an APR::BucketAlloc object:

```
$ba->destroy;
```

- **arg1: `$ba` (`APR::BucketAlloc object`)**

  The freelist to destroy.

- **ret: no return value**
- **since: 2.0.00**

Once destroyed this object may not be used again.

You need to destroy `$ba` **only** if you have created it via `APR::BucketAlloc->new`. If you try to destroy an allocation not created by this method, you will get a segmentation fault.

Moreover normally it is not necessary to destroy allocators, since the pool which created them will destroy them during that pool's cleanup phase.

30.3 See Also

mod_perl 2.0 documentation.

30.4 Copyright

30.5 Authors

The mod_perl development team and numerous contributors.

31 APR::BucketType - Perl API for APR bucket types

31.1 Synopsis

```
use APR::BucketType ();

my $name = $b_type->name;
```

31.2 Description

APR::BucketType allows you to query bucket object type properties.

31.3 API

APR::BucketType provides the following functions and/or methods:

31.3.1 name

Get the name of the bucket type:

```
my $bucket_type_name = $b_type->name;
```

- **arg1: $b_type (APR::BucketType object)**
- **ret: $bucket_type_name (string)**
- **since: 2.0.00**

Example:

```
use APR::Bucket ();
use APR::BucketType ();
my $eos_b = APR::Bucket::eos_create($ba);
my $b_type = $eos_b->type;
my $name = $b_type->name;
```

Now $name contains *'EOS'*.

31.4 See Also

mod_perl 2.0 documentation.

31.5 Copyright

31.6 Authors

The mod_perl development team and numerous contributors.

32 APR::Const - Perl Interface for APR Constants

32.1 Synopsis

```
# make the constants available but don't import them
use APR::Const -compile => qw(constant names ...);

# w/o the => syntax sugar
use APR::Const ("-compile", qw(constant names ...));

# compile and import the constants
use APR::Const qw(constant names ...);
```

32.2 Description

This package contains constants specific to APR features.

Refer to `the Apache2::Const description section` for more information.

32.3 Constants

32.3.1 :common

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

The `:common` group is for XXX constants.

32.3.1.1 APR::Const::SUCCESS

- **since: 2.0.00**

32.3.2 :error

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

The `:error` group is for XXX constants.

32.3.2.1 APR::Const::EABOVEROOT

- **since: 2.0.00**

32.3.2.2 APR::Const::EABSOLUTE

- **since: 2.0.00**

32.3.2.3 APR::Const::EACCES

- **since: 2.0.00**

Due to possible variants in conditions matching EACCES, for checking error codes against this you most likely want to use the APR::Status::is_EACCES function instead.

32.3.2.4 APR::Const::EAGAIN

- **since: 2.0.00**

The error *Resource temporarily unavailable*, may be returned by many different system calls, especially IO calls. Most likely you want to use the APR::Status::is_EAGAIN function instead.

32.3.2.5 APR::Const::EBADDATE

- **since: 2.0.00**

32.3.2.6 APR::Const::EBADF

- **since: 2.0.00**

32.3.2.7 APR::Const::EBADIP

- **since: 2.0.00**

32.3.2.8 APR::Const::EBADMASK

- **since: 2.0.00**

32.3.2.9 APR::Const::EBADPATH

- **since: 2.0.00**

32.3.2.10 APR::Const::EBUSY

- **since: 2.0.00**

32.3.2.11 APR::Const::ECONNABORTED

- **since: 2.0.00**

Due to possible variants in conditions matching ECONNABORTED, for checking error codes against this you most likely want to use the APR::Status::is_ECONNABORTED function instead.

32.3.2.12 APR::Const::ECONNREFUSED

- **since: 2.0.00**

32.3.2.13 APR::Const::ECONNRESET

- **since: 2.0.00**

Due to possible variants in conditions matching `ECONNRESET`, for checking error codes against this you most likely want to use the `APR::Status::is_ECONNRESET` function instead.

32.3.2.14 APR::Const::EDSOOPEN

- **since: 2.0.00**

32.3.2.15 APR::Const::EEXIST

- **since: 2.0.00**

32.3.2.16 APR::Const::EFTYPE

- **since: 2.0.00**

32.3.2.17 APR::Const::EGENERAL

- **since: 2.0.00**

32.3.2.18 APR::Const::EHOSTUNREACH

- **since: 2.0.00**

32.3.2.19 APR::Const::EINCOMPLETE

- **since: 2.0.00**

32.3.2.20 APR::Const::EINIT

- **since: 2.0.00**

32.3.2.21 APR::Const::EINPROGRESS

- **since: 2.0.00**

32.3.2.22 APR::Const::EINTR

- **since: 2.0.00**

32.3.2.23 APR::Const::EINVAL

- **since: 2.0.00**

32.3.2.24 APR::Const::EINVALSOCK

- **since: 2.0.00**

32.3.2.25 APR::Const::EMFILE

- **since: 2.0.00**

32.3.2.26 APR::Const::EMISMATCH

- **since: 2.0.00**

32.3.2.27 APR::Const::ENAMETOOLONG

- **since: 2.0.00**

32.3.2.28 APR::Const::END

- **since: 2.0.00**

32.3.2.29 APR::Const::ENETUNREACH

- **since: 2.0.00**

32.3.2.30 APR::Const::ENFILE

- **since: 2.0.00**

32.3.2.31 APR::Const::ENODIR

- **since: 2.0.00**

32.3.2.32 APR::Const::ENOENT

- **since: 2.0.00**

Due to possible variants in conditions matching `ENOENT`, for checking error codes against this you most likely want to use the `APR::Status::is_ENOENT` function instead.

32.3.2.33 APR::Const::ENOLOCK

- **since: 2.0.00**

32.3.2.34 APR::Const::ENOMEM

- **since: 2.0.00**

32.3.2.35 APR::Const::ENOPOLL

- **since: 2.0.00**

32.3.2.36 APR::Const::ENOPOOL

- **since: 2.0.00**

32.3.2.37 APR::Const::ENOPROC

- **since: 2.0.00**

32.3.2.38 APR::Const::ENOSHMAVAIL

- **since: 2.0.00**

32.3.2.39 APR::Const::ENOSOCKET

- **since: 2.0.00**

32.3.2.40 APR::Const::ENOSPC

- **since: 2.0.00**

32.3.2.41 APR::Const::ENOSTAT

- **since: 2.0.00**

32.3.2.42 APR::Const::ENOTDIR

- **since: 2.0.00**

32.3.2.43 APR::Const::ENOTEMPTY

- **since: 2.0.00**

32.3.2.44 APR::Const::ENOTHDKEY

- **since: 2.0.00**

32.3.2.45 APR::Const::ENOTHREAD

- **since: 2.0.00**

32.3.2.46 APR::Const::ENOTIME

- **since: 2.0.00**

32.3.2.47 APR::Const::ENOTIMPL

Something is not implemented

- **since: 2.0.00**

32.3.2.48 APR::Const::ENOTSOCK

- **since: 2.0.00**

32.3.2.49 APR::Const::EOF

- **since: 2.0.00**

Due to possible variants in conditions matching `EOF`, for checking error codes against this you most likely want to use the `APR::Status::is_EOF` function instead.

32.3.2.50 APR::Const::EPATHWILD

- **since: 2.0.00**

32.3.2.51 APR::Const::EPIPE

- **since: 2.0.00**

32.3.2.52 APR::Const::EPROC_UNKNOWN

- **since: 2.0.00**

32.3.2.53 APR::Const::ERELATIVE

- **since: 2.0.00**

32.3.2.54 APR::Const::ESPIPE

- **since: 2.0.00**

32.3.2.55 APR::Const::ESYMNOTFOUND

- **since: 2.0.00**

32.3.2.56 APR::Const::ETIMEDOUT

- **since: 2.0.00**

32.3.2.57 APR::Const::EXDEV

- **since: 2.0.00**

32.3.3 :fopen

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

The `:fopen` group is for XXX constants.

32.3.3.1 APR::Const::FOPEN_BINARY

- **since: 2.0.00**

32.3.3.2 APR::Const::FOPEN_BUFFERED

- **since: 2.0.00**

32.3.3.3 APR::Const::FOPEN_CREATE

- **since: 2.0.00**

32.3.3.4 APR::Const::FOPEN_DELONCLOSE

- **since: 2.0.00**

32.3.3.5 APR::Const::FOPEN_EXCL

- **since: 2.0.00**

32.3.3.6 APR::Const::FOPEN_PEND

- **since: 2.0.00**

32.3.3.7 APR::Const::FOPEN_READ

- **since: 2.0.00**

32.3.3.8 APR::Const::FOPEN_TRUNCATE

- **since: 2.0.00**

32.3.3.9 APR::Const::FOPEN_WRITE

- **since: 2.0.00**

32.3.4 :filepath

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

The `:filepath` group is for XXX constants.

32.3.4.1 APR::Const::FILEPATH_ENCODING_LOCALE

- **since: 2.0.00**

32.3.4.2 APR::Const::FILEPATH_ENCODING_UNKNOWN

- **since: 2.0.00**

32.3.4.3 APR::Const::FILEPATH_ENCODING_UTF8

- **since: 2.0.00**

32.3.4.4 APR::Const::FILEPATH_NATIVE

- **since: 2.0.00**

32.3.4.5 APR::Const::FILEPATH_NOTABOVEROOT

- **since: 2.0.00**

32.3.4.6 APR::Const::FILEPATH_NOTABSOLUTE

- **since: 2.0.00**

32.3.4.7 APR::Const::FILEPATH_NOTRELATIVE

- **since: 2.0.00**

32.3.4.8 APR::Const::FILEPATH_SECUREROOT

- **since: 2.0.00**

32.3.4.9 APR::Const::FILEPATH_SECUREROOTTEST

- **since: 2.0.00**

32.3.4.10 APR::Const::FILEPATH_TRUENAME

- **since: 2.0.00**

32.3.5 :fprot

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

The `:fprot` group is used by `$finfo->protection`.

32.3.5.1 APR::Const::FPROT_GEXECUTE

Execute by group

- **since: 2.0.00**

32.3.5.2 APR::Const::FPROT_GREAD

Read by group

- **since: 2.0.00**

32.3.5.3 APR::Const::FPROT_GSETID

Set group id

- **since: 2.0.00**

32.3.5.4 APR::Const::FPROT_GWRITE

Write by group

- **since: 2.0.00**

32.3.5.5 APR::Const::FPROT_OS_DEFAULT

use OS’s default permissions

- **since: 2.0.00**

32.3.5.6 APR::Const::FPROT_UEXECUTE

Execute by user

- **since: 2.0.00**

32.3.5.7 APR::Const::FPROT_UREAD

Read by user

- **since: 2.0.00**

32.3.5.8 APR::Const::FPROT_USETID

Set user id

- **since: 2.0.00**

32.3.5.9 APR::Const::FPROT_UWRITE

Write by user

- **since: 2.0.00**

32.3.5.10 APR::Const::FPROT_WEXECUTE

Execute by others

- **since: 2.0.00**

32.3.5.11 APR::Const::FPROT_WREAD

Read by others

- **since: 2.0.00**

32.3.5.12 APR::Const::FPROT_WSTICKY

Sticky bit

- **since: 2.0.00**

32.3.5.13 APR::Const::FPROT_WWRITE

Write by others

- **since: 2.0.00**

32.3.6 :filetype

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

The `:filetype` group is used by `$finfo->filetype`.

32.3.6.1 APR::Const::FILETYPE_BLK

a file is a block device

- **since: 2.0.00**

32.3.6.2 APR::Const::FILETYPE_CHR

a file is a character device

- **since: 2.0.00**

32.3.6.3 APR::Const::FILETYPE_DIR

a file is a directory

- **since: 2.0.00**

32.3.6.4 APR::Const::FILETYPE_LNK

a file is a symbolic link

- **since: 2.0.00**

32.3.6.5 APR::Const::FILETYPE_NOFILE

the file type is undedetermined.

- **since: 2.0.00**

32.3.6.6 APR::Const::FILETYPE_PIPE

a file is a FIFO or a pipe.

- **since: 2.0.00**

32.3.6.7 APR::Const::FILETYPE_REG

a file is a regular file.

- **since: 2.0.00**

32.3.6.8 APR::Const::FILETYPE_SOCK

a file is a [unix domain] socket.

- **since: 2.0.00**

32.3.6.9 APR::Const::FILETYPE_UNKFILE

a file is of some other unknown type or the type cannot be determined.

- **since: 2.0.00**

32.3.7 :finfo

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

The `:finfo` group is used by `stat()` and `$finfo->valid`.

32.3.7.1 APR::Const::FINFO_ATIME

Access Time

- **since: 2.0.00**

32.3.7.2 APR::Const::FINFO_CSIZE

Storage size consumed by the file

- **since: 2.0.00**

32.3.7.3 APR::Const::FINFO_CTIME

Creation Time

- **since: 2.0.00**

32.3.7.4 APR::Const::FINFO_DEV

Device

- **since: 2.0.00**

32.3.7.5 APR::Const::FINFO_DIRENT

an atomic unix apr_dir_read()

- **since: 2.0.00**

32.3.7.6 APR::Const::FINFO_GPROT

Group protection bits

- **since: 2.0.00**

32.3.7.7 APR::Const::FINFO_GROUP

Group id

- **since: 2.0.00**

32.3.7.8 APR::Const::FINFO_ICASE

whether device is case insensitive

- **since: 2.0.00**

32.3.7.9 APR::Const::FINFO_IDENT

device and inode

- **since: 2.0.00**

32.3.7.10 APR::Const::FINFO_INODE

Inode

- **since: 2.0.00**

32.3.7.11 APR::Const::FINFO_LINK

Stat the link not the file itself if it is a link

- **since: 2.0.00**

32.3.7.12 APR::Const::FINFO_MIN

type, mtime, ctime, atime, size

- **since: 2.0.00**

32.3.7.13 APR::Const::FINFO_MTIME

Modification Time

- **since: 2.0.00**

32.3.7.14 APR::Const::FINFO_NAME

name in proper case

- **since: 2.0.00**

32.3.7.15 APR::Const::FINFO_NLINK

Number of links

- **since: 2.0.00**

32.3.7.16 APR::Const::FINFO_NORM

All fields provided by an atomic unix apr_stat()

- **since: 2.0.00**

32.3.7.17 APR::Const::FINFO_OWNER

user and group

- **since: 2.0.00**

32.3.7.18 APR::Const::FINFO_PROT

all protections

- **since: 2.0.00**

32.3.7.19 APR::Const::FINFO_SIZE

Size of the file

- **since: 2.0.00**

32.3.7.20 APR::Const::FINFO_TYPE

Type

- **since: 2.0.00**

32.3.7.21 APR::Const::FINFO_UPROT

User protection bits

- **since: 2.0.00**

32.3.7.22 APR::Const::FINFO_USER

User id

- **since: 2.0.00**

32.3.7.23 APR::Const::FINFO_WPROT

World protection bits

- **since: 2.0.00**

32.3.8 :flock

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

The `:flock` group is for XXX constants.

32.3.8.1 APR::Const::FLOCK_EXCLUSIVE

- **since: 2.0.00**

32.3.8.2 APR::Const::FLOCK_NONBLOCK

- **since: 2.0.00**

32.3.8.3 APR::Const::FLOCK_SHARED

- **since: 2.0.00**

32.3.8.4 APR::Const::FLOCK_TYPEMASK

- **since: 2.0.00**

32.3.9 :hook

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

The `:hook` group is for XXX constants.

32.3.9.1 APR::Const::HOOK_FIRST

- **since: 2.0.00**

32.3.9.2 APR::Const::HOOK_LAST

- **since: 2.0.00**

32.3.9.3 APR::Const::HOOK_MIDDLE

- **since: 2.0.00**

32.3.9.4 APR::Const::HOOK_REALLY_FIRST

- **since: 2.0.00**

32.3.9.5 APR::Const::HOOK_REALLY_LAST

- **since: 2.0.00**

32.3.10 :limit

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

The `:limit` group is for XXX constants.

32.3.10.1 APR::Const::LIMIT_CPU

- **since: 2.0.00**

32.3.10.2 APR::Const::LIMIT_MEM

- **since: 2.0.00**

32.3.10.3 APR::Const::LIMIT_NOFILE

- **since: 2.0.00**

32.3.10.4 APR::Const::LIMIT_NPROC

- **since: 2.0.00**

32.3.11 :lockmech

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

The `:lockmech` group is for XXX constants.

32.3.11.1 APR::Const::LOCK_DEFAULT

- **since: 2.0.00**

32.3.11.2 APR::Const::LOCK_FCNTL

- **since: 2.0.00**

32.3.11.3 APR::Const::LOCK_FLOCK

- **since: 2.0.00**

32.3.11.4 APR::Const::LOCK_POSIXSEM

- **since: 2.0.00**

32.3.11.5 APR::Const::LOCK_PROC_PTHREAD

- **since: 2.0.00**

32.3.11.6 APR::Const::LOCK_SYSVSEM

- **since: 2.0.00**

32.3.12 :poll

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

The `:poll` group is used by `poll`.

32.3.12.1 APR::Const::POLLERR

- **since: 2.0.00**

Pending error

32.3.12.2 APR::Const::POLLHUP

- **since: 2.0.00**

Hangup occurred

32.3.12.3 APR::Const::POLLIN

- **since: 2.0.00**

Can read without blocking

32.3.12.4 APR::Const::POLLNVAL

- **since: 2.0.00**

Descriptior invalid

32.3.12.5 APR::Const::POLLOUT

- **since: 2.0.00**

Can write without blocking

32.3.12.6 APR::Const::POLLPRI

- **since: 2.0.00**

Priority data available

32.3.13 :read_type

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

The `:read_type` group is for IO constants.

32.3.13.1 APR::Const::BLOCK_READ

- **since: 2.0.00**

the read function blocks

32.3.13.2 APR::Const::NONBLOCK_READ

- **since: 2.0.00**

the read function does not block

32.3.14 :shutdown_how

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

The `:shutdown_how` group is for XXX constants.

32.3.14.1 APR::Const::SHUTDOWN_READ

- **since: 2.0.00**

32.3.14.2 APR::Const::SHUTDOWN_READWRITE

- **since: 2.0.00**

32.3.14.3 APR::Const::SHUTDOWN_WRITE

- **since: 2.0.00**

32.3.15 :socket

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

The `:socket` group is for the `APR::Socket` object constants, in methods `opt_get` and `opt_set`.

The following section discusses in detail each of the `:socket` constants.

32.3.15.1 APR::Const::SO_DEBUG

Possible values:

XXX

- **since: 2.0.00**

Turns on debugging information

32.3.15.2 APR::Const::SO_DISCONNECTED

Queries the disconnected state of the socket. (Currently only used on Windows)

Possible values:

XXX

- **since: 2.0.00**

32.3.15.3 APR::Const::SO_KEEPALIVE

Keeps connections active

Possible values:

XXX

- **since: 2.0.00**

32.3.15.4 APR::Const::SO_LINGER

Lingers on close if data is present

- **since: 2.0.00**

32.3.15.5 APR::Const::SO_NONBLOCK

Turns blocking IO mode on/off for socket.

Possible values:

```
1 nonblocking
0 blocking
```

For example, to set a socket to a blocking IO mode:

```
use APR::Socket ();
use APR::Const    -compile => qw(SO_NONBLOCK);
...
if ($socket->opt_get(APR::Const::SO_NONBLOCK)) {
    $socket->opt_set(APR::Const::SO_NONBLOCK => 0);
}
```

You don’t have to query for this option, before setting it. It was done for the demonstration purpose.

- **since: 2.0.00**

32.3.15.6 APR::Const::SO_RCVBUF

Controls the `ReceiveBufferSize` setting

Possible values:

XXX

- **since: 2.0.00**

32.3.15.7 APR::Const::SO_REUSEADDR

The rules used in validating addresses supplied to bind should allow reuse of local addresses.

Possible values:

XXX

- **since: 2.0.00**

32.3.15.8 APR::Const::SO_SNDBUF

Controls the SendBufferSize setting

Possible values:

XXX

- **since: 2.0.00**

32.3.16 :status

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

The :status group is for the API that return status code, or set the error variable XXXXXX.

The following section discusses in detail each of the available :status constants.

32.3.16.1 APR::Const::TIMEUP

The operation did not finish before the timeout.

- **since: 2.0.00**

Due to possible variants in conditions matching TIMEUP, for checking error codes against this you most likely want to use the APR::Status::is_TIMEUP function instead.

32.3.17 :table

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

The :table group is for overlap() and compress() constants. See APR::Table for details.

32.3.17.1 APR::Const::OVERLAP_TABLES_MERGE

- **since: 2.0.00**

See APR::Table::compress and APR::Table::overlap.

32.3.17.2 APR::Const::OVERLAP_TABLES_SET

- **since: 2.0.00**

See APR::Table::compress and APR::Table::overlap.

32.3.18 :uri

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

The :uri group of constants is for manipulating URIs.

32.3.18.1 APR::Const::URI_ACAP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.2 APR::Const::URI_FTP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.3 APR::Const::URI_GOPHER_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.4 APR::Const::URI_HTTPS_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.5 APR::Const::URI_HTTP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.6 APR::Const::URI_IMAP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.7 APR::Const::URI_LDAP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.8 APR::Const::URI_NFS_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.9 APR::Const::URI_NNTP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.10 APR::Const::URI_POP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.11 APR::Const::URI_PROSPERO_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.12 APR::Const::URI_RTSP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.13 APR::Const::URI_SIP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.14 APR::Const::URI_SNEWS_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.15 APR::Const::URI_SSH_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.16 APR::Const::URI_TELNET_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.17 APR::Const::URI_TIP_DEFAULT_PORT

- **since: 2.0.00**

32.3.18.18 APR::Const::URI_UNP_OMITPASSWORD

- **since: 2.0.00**

See `APR::URI::unparse`.

32.3.18.19 APR::Const::URI_UNP_OMITPATHINFO

- **since: 2.0.00**

See `APR::URI::unparse`.

32.3.18.20 APR::Const::URI_UNP_OMITQUERY

- **since: 2.0.00**

See APR::URI::unparse.

32.3.18.21 APR::Const::URI_UNP_OMITSITEPART

- **since: 2.0.00**

See APR::URI::unparse.

32.3.18.22 APR::Const::URI_UNP_OMITUSER

- **since: 2.0.00**

See APR::URI::unparse.

32.3.18.23 APR::Const::URI_UNP_OMITUSERINFO

- **since: 2.0.00**

32.3.18.24 APR::Const::URI_UNP_REVEALPASSWORD

- **since: 2.0.00**

See APR::URI::unparse.

32.3.18.25 APR::Const::URI_WAIS_DEFAULT_PORT

- **since: 2.0.00**

32.3.19 Other Constants

32.3.19.1 APR::PerlIO::PERLIO_LAYERS_ARE_ENABLED

- **since: 2.0.00**

See APR::PerlIO::Constants)

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 2.0.

32.6 Authors

The mod_perl development team and numerous contributors.

33 APR::Date - Perl API for APR date manipulating functions

33.1 Synopsis

```
use APR::Date ();

# parse HTTP-complient date string
$date_string = 'Sun, 06 Nov 1994 08:49:37 GMT';
$date_parsed = APR::Date::parse_http($date_string);

# parse RFC822-complient date string
$date_string = 'Sun, 6 Nov 94 8:49:37 GMT';
$date_parsed = APR::Date::parse_rfc($date_string);
```

33.2 Description

`APR::Socket` provides the Perl interface to APR date manipulating functions.

33.3 API

`APR::Date` provides the following functions and/or methods:

33.3.1 `parse_http`

Parse HTTP date strings

```
$date_parsed = parse_http($date_string);
```

- **arg1: `$date_string` (string)**

  The date string can be in one of the following formats:

  ```
  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
  ```

  refer to RFC2616 for the details (GMT is assumed, regardless of the used timezone).

- **ret: `$date_parsed` (number)**

  the number of microseconds since 1 Jan 1970 GMT, or 0 if out of range or if the date is invalid.

- **since: 2.0.00**

Remember to divide the return value by 1_000_000 if you need it in seconds.

33.3.2 `parse_rfc`

Parse a string resembling an RFC 822 date. It's meant to be lenient in its parsing of dates. Hence, this will parse a wider range of dates than `parse_http()`.

```
$date_parsed = parse_rfc($date_string);
```

- **arg1: `$date_string` (string)**

  The date string can be in one of the following formats:

  ```
  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: `$date_parsed` (number)**

  the number of microseconds since 1 Jan 1970 GMT, or 0 if out of range or if the date is invalid.

- **since: 2.0.00**

Remember to divide the return value by 1_000_000 if you need it in seconds.

33.4 See Also

mod_perl 2.0 documentation.

33.5 Copyright

33.6 Authors

The mod_perl development team and numerous contributors.

34 APR::Error - Perl API for APR/Apache/mod_perl exceptions

34.1 Synopsis

```
eval { $obj->mp_method() };
if ($@ && $ref $@ eq 'APR::Error' && $@ == $some_code) {
    # handle the exception
}
else {
    die $@; # rethrow it
}
```

34.2 Description

`APR::Error` handles APR/Apache/mod_perl exceptions for you, while leaving you in control.

Apache and APR API return a status code for almost all methods, so if you didn't check the return code and handled any possible problems, you may have silent failures which may cause all kind of obscure problems. On the other hand checking the status code after each call is just too much of a kludge and makes quick prototyping/development almost impossible, not talking about the code readability. Having methods return status codes, also complicates the API if you need to return other values.

Therefore to keep things nice and make the API readable we decided to not return status codes, but instead throw exceptions with `APR::Error` objects for each method that fails. If you don't catch those exceptions, everything works transparently - perl will intercept the exception object and `die()` with a proper error message. So you get all the errors logged without doing any work.

Now, in certain cases you don't want to just die, but instead the error needs to be trapped and handled. For example if some IO operation times out, may be it is OK to trap that and try again. If we were to die with an error message, you would have had to match the error message, which is ugly, inefficient and may not work at all if locale error strings are involved. Therefore you need to be able to get the original status code that Apache or APR has generated. And the exception objects give you that if you want to. Moreover the objects contain additional information, such as the function name (in case you were eval'ing several commands in one block), file and line number where that function was invoked from. More attributes could be added in the future.

`APR::Error` uses Perl operator overloading, such that in boolean and numerical contexts, the object returns the status code; in the string context the full error message is returned.

When intercepting exceptions you need to check whether `$@` is an object (reference). If your application uses other exception objects you additionally need to check whether this is a an `APR::Error` object. Therefore most of the time this is enough:

```
eval { $obj->mp_method() };
if ($@ && $ref $@ && $@ == $some_code)
    warn "handled exception: $@";
}
```

But with other, non-mod_perl, exception objects you need to do:

```
eval { $obj->mp_method() };
if ($@ && $ref $@ eq 'APR::Error' && $@ == $some_code)
    warn "handled exception: $@";
}
```

In theory you could even do:

```
eval { $obj->mp_method() };
if ($@ && $@ == $some_code)
    warn "handled exception: $@";
}
```

but it's possible that the method will die with a plain string and not an object, in which case `$@ == $some_code` won't quite work. Remember that mod_perl throws exception objects only when Apache and APR fail, and in a few other special cases of its own (like `exit`).

```
warn "handled exception: $@" if $@ && $ref $@;
```

There are two ways to figure out whether an error fits your case. In most cases you just compare `$@` with an the error constant. For example if a socket has a timeout set and the data wasn't read within the timeout limit a `APR::Const::TIMEUP`)

```
use APR::Const -compile => qw(TIMEUP);
$sock->timeout_set(1_000_000); # 1 sec
my $buff;
eval { $sock->recv($buff, BUFF_LEN) };
if ($@ && ref $@ && $@ == APR::Const::TIMEUP) {

}
```

However there are situations, where on different Operating Systems a different error code will be returned. In which case to simplify the code you should use the special subroutines provided by the `APR::Status` class. One such condition is socket `recv()` timeout, which on Unix throws the `EAGAIN` error, but on other system it throws a different error. In this case `APR::Status::is_EAGAIN` should be used.

Let's look at a complete example. Here is a code that performs a socket read:

```
my $rlen = $sock->recv(my $buff, 1024);
warn "read $rlen bytes\n";
```

and in certain cases it times out. The code will die and log the reason for the failure, which is fine, but later on you may decide that you want to have another attempt to read before dying and add some fine grained sleep time between attempts, which can be achieved with `select`. Which gives us:

```
use APR::Status ();
# ....
my $tries = 0;
my $buffer;
RETRY: my $rlen = eval { $sock->recv($buffer, SIZE) };
if ($@)
    die $@ unless ref $@ && APR::Status::is_EAGAIN($@);
```

```
    if ($tries++ < 3) {
        # sleep 250msec
        select undef, undef, undef, 0.25;
        goto RETRY;
    }
    else {
        # do something else
    }
}
warn "read $rlen bytes\n"
```

Notice that we handle non-object and non-`APR::Error` exceptions as well, by simply re-throwing them.

Finally, the class is called `APR::Error` because it needs to be used outside mod_perl as well, when called from APR applications written in Perl.

34.3 API

34.3.1 `cluck`

`cluck` is an equivalent of `Carp::cluck` that works with `APR::Error` exception objects.

34.3.2 `confess`

`confess` is an equivalent of `Carp::confess` that works with `APR::Error` exception objects.

34.3.3 `strerror`

Convert APR error code to its string representation.

```
$error_str = APR::Error::strerror($rc);
```

- **ret: `$rc` (`APR::Const status constant`)**

  The numerical value for the return (error) code

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

  The string error message corresponding to the numerical value inside `$rc`. (Similar to the C function `strerror(3)`)

- **since: 2.0.00**

Example:

Try to retrieve the bucket brigade, and if the return value doesn’t indicate success or end of file (usually in protocol handlers) die, but give the user the human-readable version of the error and not just the code.

```
my $rc = $c->input_filters->get_brigade($bb_in,
                                        Apache2::Const::MODE_GETLINE);
if ($rc != APR::Const::SUCCESS && $rc != APR::Const::EOF) {
    my $error = APR::Error::strerror($rc);
    die "get_brigade error: $rc: $error\n";
}
```

It’s probably a good idea not to omit the numerical value in the error message, in case the error string is generated with non-English locale.

34.4 See Also

mod_perl 2.0 documentation.

34.5 Copyright

34.6 Authors

The mod_perl development team and numerous contributors.

35 APR::Finfo - Perl API for APR fileinfo structure

35.1 Synopsis

```
use APR::Finfo ();
use APR::Const -compile => qw(FINFO_NORM);
my $finfo = APR::Finfo::stat("/tmp/test", APR::Const::FINFO_NORM, $pool);

$device = $finfo->device;     # (stat $file)[0]
$inode  = $finfo->inode;      # (stat $file)[1]
# stat returns an octal number while protection is hex
$prot   = $finfo->protection; # (stat $file)[2]
$nlink  = $finfo->nlink;      # (stat $file)[3]
$gid    = $finfo->group;      # (stat $file)[4]
$uid    = $finfo->user;       # (stat $file)[5]
$size   = $finfo->size;       # (stat $file)[7]
$atime  = $finfo->atime;      # (stat $file)[8]
$mtime  = $finfo->mtime;      # (stat $file)[9]
$ctime  = $finfo->ctime;      # (stat $file)[10]

$csize = $finfo->csize; # consumed size: not portable!

$filetype = $finfo->filetype; # file/dir/socket/etc

$fname = $finfo->fname;
$name  = $finfo->name;  # in filesystem case:

# valid fields that can be queried
$valid = $finfo->valid;
```

35.2 Description

APR fileinfo structure provides somewhat similar information to Perl's `stat()` call, but you will want to use this module's API to query an already `stat()'ed` filehandle to avoid an extra system call or to query attributes specific to APR file handles.

During the HTTP request handlers coming after `PerlMapToStorageHandler`, `$r->finfo` already contains the cached values from the apr's `stat()` call. So you don't want to perform it again, but instead get the `APR::Finfo` object via:

```
my $finfo = $r->finfo;
```

35.3 API

`APR::Finfo` provides the following functions and/or methods:

35.3.1 `atime`

Get the time the file was last accessed:

```
$atime = $finfo->atime;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$atime` (integer)**

  Last access time in seconds since the epoch

- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[8]
```

Note that this method may not be reliable on all platforms, most notably Win32 -- FAT32 filesystems appear to work properly, but NTFS filesystems do not.

35.3.2 `csize`

Get the storage size consumed by the file

```
$csize = $finfo->csize;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$csize` (integer)**
- **since: 2.0.00**

Chances are that you don't want to use this method, since its functionality is not supported on most platforms (in which case it always returns 0).

35.3.3 `ctime`

Get the time the file was last changed

```
$ctime = $finfo->ctime;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$ctime` (integer)**

  Inode change time in seconds since the epoch

- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[10]
```

The ctime field is non-portable. In particular, you cannot expect it to be a "creation time", see "Files and Filesystems" in the *perlport* manpage for details.

35.3.4 device

Get the id of the device the file is on.

```
$device = $finfo->device;
```

- **obj: $finfo (APR::Finfo object)**
- **return: $device (integer)**
- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[0]
```

Note that this method is non-portable. It doesn't work on all platforms, most notably Win32.

35.3.5 filetype

Get the type of file.

```
$filetype = $finfo->filetype;
```

- **obj: $finfo (APR::Finfo object)**
- **return: $filetype (:filetype constant)**
- **since: 2.0.00**

For example:

```
use APR::Pool;
use APR::Finfo;
use APR::Const -compile => qw(FILETYPE_DIR FILETYPE_REG FINFO_NORM);
my $pool  = APR::Pool->new();
my $finfo = APR::Finfo::stat("/tmp", APR::Const::FINFO_NORM, $pool);
my $finfo = $finfo->filetype;
if ($finfo == APR::Const::FILETYPE_REG) {
    print "regular file";
}
elsif ($finfo == APR::Const::FILETYPE_REG) {
    print "directory";
}
else {
    print "other file";
}
```

Since */tmp* is a directory, this will print:

```
directory
```

35.3.6 fname

Get the pathname of the file (possibly unrooted)

```
$fname = $finfo->fname;
```

- **obj: $finfo (APR::Finfo object)**
- **return: $filetype (string)**
- **since: 2.0.00**

35.3.7 group

Get the group id that owns the file:

```
$gid = $finfo->group;
```

- **obj: $finfo (APR::Finfo object)**
- **return: $gid (number)**
- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[5]
```

Note that this method may not be meaningful on all platforms, most notably Win32. Incorrect results have also been reported on some versions of OSX.

35.3.8 inode

Get the inode of the file.

```
$inode = $finfo->inode;
```

- **obj: $finfo (APR::Finfo object)**
- **return: $inode (integer)**
- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[1]
```

Note that this method may not be meaningful on all platforms, most notably Win32.

35.3.9 mtime

The time the file was last modified

```
$mtime = $finfo->mtime;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$mtime` (integer)**

  Last modify time in seconds since the epoch

- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[9]
```

35.3.10 name

Get the file's name (no path) in filesystem case:

```
$name = $finfo->name;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$device` (string)**
- **since: 2.0.00**

35.3.11 nlink

Get the number of hard links to the file.

```
$nlink = $finfo->nlink;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$nlink` (integer)**
- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[3]
```

35.3.12 protection

Get the access permissions of the file. Mimics Unix access rights.

```
$prot = $finfo->protection;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$prot` (`:fprot constant`)**
- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[2]
```

Note: Perl's stat returns an octal number while mod_perl's `protection` returns a hex number.

See perldoc -f stat and APR's file_io for more information on each.

35.3.13 `size`

Get the size of the file

```
$size = $finfo->size;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$size` (integer)**

  Total size of file, in bytes

- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[7]
```

35.3.14 `stat`

Get the specified file's stats.

```
$finfo = APR::Finfo::stat($fname, $wanted_fields, $p);
```

- **arg1: `$fname` (string)**

  The path to the file to `stat()`.

- **arg2: `$wanted_fields` (`:finfo constant`)**

  The desired fields, as a bitmask flag of `APR::FINFO_*` constants.

  Notice that you can also use the constants that already combine several elements in one. For example `APR::Const::FINFO_PROT` asks for all protection bits, `APR::Const::FINFO_MIN` asks for the following fields: type, mtime, ctime, atime, size and `APR::Const::FINFO_NORM` asks for all atomic unix `apr_stat()` fields (similar to perl's `stat()`).

- **arg3: `$p` (`APR::Pool object`)**

  the pool to use to allocate the file stat structure.

- **ret: `$finfo` (`APR::Finfo object`)**
- **since: 2.0.00**

For example, here is how to get most of the `stat` fields:

```
use APR::Pool ();
use APR::Finfo ();
use APR::Const -compile => qw(FINFO_NORM);
my $pool = APR::Pool->new();
my $finfo = APR::Finfo::stat("/tmp/test", APR::Const::FINFO_NORM, $pool);
```

35.3.15 `user`

Get the user id that owns the file:

```
$uid = $finfo->user;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **return: `$uid` (number)**
- **since: 2.0.00**

This method returns the same value as Perl's:

```
(stat $filename)[4]
```

Note that this method may not be meaningful on all platforms, most notably Win32.

35.3.16 `valid`

The bitmask describing valid fields of this apr_finfo_t structure including all available 'wanted' fields and potentially more

```
$valid = $finfo->valid;
```

- **obj: `$finfo` (`APR::Finfo object`)**
- **arg1: `$valid` (bitmask)**

  This bitmask flag should be bit-OR'ed against `:finfo constant` constants.

- **since: 2.0.00**

35.4 See Also

mod_perl 2.0 documentation.

35.5 Copyright

35.6 Authors

The mod_perl development team and numerous contributors.

36 APR::IpSubnet - Perl API for accessing APRs ip_subnet structures

36.1 Synopsis

```
use APR::IpSubnet ();

my $ipsub = APR::IpSubnet->new($pool, "127.0.0.1");
$ok = $ipsub->test($sock_addr);
```

36.2 Description

`APR::IpSubnet` object represents a range of IP addresses (IPv4/IPv6). A socket connection can be matched against this range to test whether the IP it's coming from is inside or outside of this range.

36.3 API

`APR::IpSubnet` provides the following functions and/or methods:

36.3.1 new

Create an IP subnet represenation object

```
$ipsubnet = APR::IpSubnet->new($pool, $ip);
$ipsubnet = APR::IpSubnet->new($pool, $ip, $mask_or_numbits);
```

- **obj: `APR::IpSubnet` (class name)**
- **arg1: `$pool` (`APR::Pool object`)**
- **arg2: `$ip` (string)**

  IP address in one of the two formats: IPv4 (e.g. *"127.0.0.1"*) or IPv6 (e.g. *"::1"*). IPv6 addresses are accepted only if APR has the IPv6 support enabled.

- **opt arg3: `$mask_or_numbits` (string)**

  An optional IP mask (e.g. *"255.0.0.0"*) or number of bits (e.g. *"15"*).

  If none provided, the default is not to mask off.

- **ret: `$ret` (`APR::IpSubnet object`)**

  The IP-subnet object

- **excpt: `APR::Error`**
- **since: 2.0.00**

36.3.2 `test`

Test the IP address in the socket address object against a pre-built ip-subnet representation.

```
$ret = $ipsub->test($sockaddr);
```

- **obj: `$ipsub` (`APR::IpSubnet object`)**

  The ip-subnet representation

- **arg1: `$sockaddr` (`APR::SockAddr object`)**

  The socket address to test

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

  true if the socket address is within the subnet, false otherwise

- **since: 2.0.00**

This method is used for testing whether or not an address is within a subnet. It's used by module `mod_access` to check whether the client IP fits into the IP range, supplied by `Allow`/`Deny` directives.

Example:

Allow accesses only from the localhost (IPv4):

```
use APR::IpSubnet ();
use Apache2::Connection ();
use Apache2::RequestRec ();
my $ipsub = APR::IpSubnet->new($r->pool, "127.0.0.1");
ok $ipsub->test($r->connection->remote_addr);
```

36.4 See Also

mod_perl 2.0 documentation.

36.5 Copyright

36.6 Authors

The mod_perl development team and numerous contributors.

37 APR::OS - Perl API for Platform-specific APR API

37.1 Synopsis

```
use APR::OS ();

# get current thread id
my $tid = APR::OS::current_thread_id();
```

37.2 Description

APR::OS provides the Perl interface to platform-specific APR API.

You should be extremely careful when relying on any of the API provided by this module, since they are no portable. So if you use those you application will be non-portable as well.

37.3 API

APR::OS provides the following methods:

37.3.1 `current_thread_id`

Get the current thread ID

```
$tid = APR::OS::current_thread_id();
```

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

  under threaded MPMs returns the current thread ID, otherwise 0.

- **since: 2.0.00**

Example:

```
use Apache2::MPM ();
use APR::OS ();
if (Apache2::MPM->is_threaded) {
    my $tid_obj = APR::OS::current_thread_id();
    print "TID: $tid";
}
else {
    print "PID: $$";
}
```

37.4 See Also

mod_perl 2.0 documentation.

37.5 Copyright

37.6 Authors

The mod_perl development team and numerous contributors.

38 APR::PerlIO -- Perl IO layer for APR

38.1 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 Apache2::Const::OK;
}

# outside mod_perl
% perl -MAPR -MAPR::PerlIO -MAPR::Pool -le \
'open my $fh, ">:APR", "/tmp/apr", APR::Pool->new or die "$!"; \
 print $fh "whoah!"; \
 close $fh;'
```

38.2 Description

`APR::PerlIO` implements a Perl IO layer using APR's file manipulation API internally.

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 file-handle which is internally hooked to APR. But you can use APR Perl IO directly if you want.

38.3 Prerequisites

Not every Perl will have full `APR::PerlIO` functionality available.

Before using the Perl IO APR layer one has to check whether it's supported by the used APR/Perl build. Perl 5.8.x or higher with perlio enabled is required. You can check whether your Perl fits the bill by running:

```
% perl -V:useperlio
useperlio='define';
```

It should say *define*.

If you need to do the checking in the code, there is a special constant provided by `APR::PerlIO`, which can be used as follows:

```
use APR::PerlIO ();
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.

38.4 Constants

38.4.1 APR::PerlIO::PERLIO_LAYERS_ARE_ENABLED

See Prerequisites.

38.5 API

Most of the API is as in normal perl IO with a few nuances listed in the following sections.

META: need to rework the exception mechanism here. Current success in using errno ($!) being set (e.g. on open()) is purely accidental and not guaranteed across all platforms and functions. So don't rely on $!. Will use `APR::Error` for that purpose.

38.5.1 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: (integer)**

success or failure value (boolean).

- **since: 2.0.00**

38.5.2 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 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.

- **since: 2.0.00**

38.6 C API

The C API provides functions to convert between Perl IO and APR Perl IO filehandles.

META: document these

38.7 See Also

mod_perl 2.0 documentation. The *perliol(1)*, *perlapio(1)* and *perl(1)* manpages.

38.8 Copyright

38.9 Authors

The mod_perl development team and numerous contributors.

39 APR::Pool - Perl API for APR pools

39.1 Synopsis

```
use APR::Pool ();

my $sp = $r->pool->new;
my $sp2 = APR::Pool->new;

# $sp3 is a subpool of $sp,
# which in turn is a subpool of $r->pool
$sp3 = $sp->new;
print '$r->pool is an ancestor of $sp3'
    if $r->pool->is_ancestor($sp3);
# but sp2 is not a sub-pool of $r->pool
print '$r->pool is not an ancestor of $sp2'
    unless $r->pool->is_ancestor($sp2);

# $sp4 and $sp are the same pool (though you can't
# compare the handle as variables)
my $sp4 = $sp3->parent_get;


# register a dummy cleanup function
# that just prints the passed args
$sp->cleanup_register(sub { print @{ $_[0] || [] } }, [1..3]);

# tag the pool
$sp->tag("My very best pool");

# clear the pool
$sp->clear();

# destroy sub pool
$sp2->destroy;
```

39.2 Description

`APR::Pool` provides an access to APR pools, which are used for an easy memory management.

Different pools have different life scopes and therefore one doesn't need to free allocated memory explicitly, but instead it's done when the pool's life is getting to an end. For example a request pool is created at the beginning of a request and destroyed at the end of it, and all the memory allocated during the request processing using the request pool is freed at once at the end of the request.

Most of the time you will just pass various pool objects to the methods that require them. And you must understand the scoping of the pools, since if you pass a long lived server pool to a method that needs the memory only for a short scoped request, you are going to leak memory. A request pool should be used in such a case. And vice versa, if you need to allocate some memory for a scope longer than a single request, then a request pool is inappropriate, since when the request will be over, the memory will be freed and bad things may happen.

If you need to create a new pool, you can always do that via the `new()` method.

39.3 API

`APR::Pool` provides the following functions and/or methods:

39.3.1 `cleanup_register`

Register cleanup callback to run

```
$pool->cleanup_register($callback);
$pool->cleanup_register($callback, $arg);
```

- **obj: `$pool` (`APR::Pool object`)**

  The pool object to register the cleanup callback for

- **arg1: `$callback` (CODE ref or sub name)**

  a cleanup callback CODE reference or just a name of the subroutine (fully qualified unless defined in the current package).

- **opt arg2: `$arg` (SCALAR)**

  If this optional argument is passed, the `$callback` function will receive it as the first and only argument when executed.

  To pass more than one argument, use an ARRAY or a HASH reference

- **ret: no return value**
- **excpt:**

  If a registered callback dies or throws an exception `$@` is stringified and passed to `warn()`. Usually, this results in printing it to the *error_log*. However, a `$SIG{__WARN__}` handler can be used to catch them.

  ```
  $pool->cleanup_register(sub {die "message1\n"});
  $pool->cleanup_register(sub {die "message2\n"});
  my @warnings;
  {
      local $SIG{__WARN__}=sub {push @warnings, @_};
      $pool->destroy;       # or simply undef $pool
  }
  ```

  Both of the cleanups above are executed at the time `$pool->destroy` is called. `@warnings` contains `message2\n` and `message1\n` afterwards. `$pool->destroy` itself does not throw an exception. Any value of `$@` is preserved.

- **since: 2.0.00**

If there is more than one callback registered (when `cleanup_register` is called more than once on the same pool object), the last registered callback will be executed first (LIFO).

Examples:

No arguments, using anon sub as a cleanup callback:

```
$r->pool->cleanup_register(sub { warn "running cleanup" });
```

One or more arguments using a cleanup code reference:

```
$r->pool->cleanup_register(\&cleanup, $r);
$r->pool->cleanup_register(\&cleanup, [$r, $foo]);
sub cleanup {
    my @args = (@_ && ref $_[0] eq ARRAY) ? @{ +shift } : shift;
    my $r = shift @args;
    warn "cleaning up";
}
```

No arguments, using a function name as a cleanup callback:

```
$r->pool->cleanup_register('foo');
```

39.3.2 `clear`

Clear all memory in the pool and run all the registered cleanups. This also destroys all sub-pools.

```
$pool->clear();
```

- **obj: `$pool(APR::Pool object)`**

  The pool to clear

- **ret: no return value**
- **since: 2.0.00**

This method differs from `destroy()` in that it is not freeing the previously allocated, but allows the pool to re-use it for the future memory allocations.

39.3.3 `DESTROY`

`DESTROY` is an alias to `destroy`. It's there so that custom `APR::Pool` objects will get properly cleaned up, when the pool object goes out of scope. If you ever want to destroy an `APR::Pool` object before it goes out of scope, use `destroy`.

- **since: 2.0.00**

39.3.4 destroy

Destroy the pool.

```
$pool->destroy();
```

- **obj: $pool (APR::Pool object)**

  The pool to destroy

- **ret: no return value**
- **since: 2.0.00**

This method takes a similar action to `clear()` and then frees all the memory.

39.3.5 is_ancestor

Determine if pool a is an ancestor of pool b

```
$ret = $pool_a->is_ancestor($pool_b);
```

- **obj: $pool_a (APR::Pool object)**

  The pool to search

- **arg1: $pool_b (APR::Pool object)**

  The pool to search for

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

  True if `$pool_a` is an ancestor of `$pool_b`.

- **since: 2.0.00**

For example create a sub-pool of a given pool and check that the pool is an ancestor of that sub-pool:

```
use APR::Pool ();
my $pp = $r->pool;
my $sp = $pp->new();
$pp->is_ancestor($sp) or die "Don’t mess with genes!";
```

39.3.6 new

Create a new sub-pool

```
my $pool_child = $pool_parent->new;
my $pool_child = APR::Pool->new;
```

- **obj: `$pool_parent(APR::Pool object)`**

  The parent pool.

  If you don't have a parent pool to create the sub-pool from, you can use this object method as a class method, in which case the sub-pool will be created from the global pool:

  ```
  my $pool_child = APR::Pool->new;
  ```

- **ret: `$pool_child(APR::Pool object)`**

  The child sub-pool

- **since: 2.0.00**

39.3.7 `parent_get`

Get the parent pool

```
$parent_pool = $child_pool->parent_get();
```

- **obj: `$child_pool(APR::Pool object)`**

  the child pool

- **ret: `$parent_pool(APR::Pool object)`**

  the parent pool. `undef` if there is no parent pool (which is the case for the top-most global pool).

- **since: 2.0.00**

Example: Calculate how big is the pool's ancestry:

```
use APR::Pool ();
sub ancestry_count {
    my $child = shift;
    my $gen = 0;
    while (my $parent = $child->parent_get) {
        $gen++;
        $child = $parent;
    }
    return $gen;
}
```

39.3.8 `tag`

Tag a pool (give it a name)

```
$pool->tag($tag);
```

- **obj: `$pool( APR::Pool object )`**

  The pool to tag

- **arg1: `$tag` (string)**

  The tag (some unique string)

- **ret: no return value**
- **since: 2.0.00**

Each pool can be tagged with a unique label. This can prove useful when doing low level apr_pool C tracing (when apr is compiled with `-DAPR_POOL_DEBUG`). It allows you to grep(1) for the tag you have set, to single out the traces relevant to you.

Though there is no way to get read the tag value, since APR doesn't provide such an accessor method.

39.4 Unsupported API

`APR::Pool` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

39.4.1 `cleanup_for_exec`

META: Autogenerated - needs to be reviewed/completed

Preparing for exec() --- close files, etc., but *don't* flush I/O 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**
- **since: subject to change**

39.5 See Also

mod_perl 2.0 documentation.

39.6 Copyright

39.7 Authors

The mod_perl development team and numerous contributors.

40 APR::SockAddr - Perl API for APR socket address structure

40.1 Synopsis

```
use APR::SockAddr ();

my $ip = $sock_addr->ip_get;
my $port = $sock_addr->port;
```

40.2 Description

APR::SockAddr provides an access to a socket address structure fields.

Normally you'd get a socket address object, by calling:

```
use Apache2::Connection ();
my $remote_sock_addr = $c->remote_addr;
my $local_sock_addr  = $c->remote_local;
```

40.3 API

APR::SockAddr provides the following functions and/or methods:

40.3.1 ip_get

Get the IP address of the socket

```
$ip = $sock_addr->ip_get();
```

- **obj: $sock_addr (APR::SockAddr object)**
- **ret: $ip (string)**
- **since: 2.0.00**

If you are familiar with how perl's Socket works:

```
use Socket 'sockaddr_in';
my ($serverport, $serverip) = sockaddr_in(getpeername($local_sock));
my ($remoteport, $remoteip) = sockaddr_in(getpeername($remote_sock));
```

in apr-speak that'd be written as:

```
use APR::SockAddr ();
use Apache2::Connection ();
my $serverport = $c->local_addr->port;
my $serverip   = $c->local_addr->ip_get;
my $remoteport = $c->remote_addr->port;
my $remoteip   = $c->remote_addr->ip_get;
```

40.3.2 port

Get the IP address of the socket

```
$port = $sock_addr->port();
```

- **obj: $sock_addr (APR::SockAddr object)**
- **ret: $port (integer)**
- **since: 2.0.00**

Example: see ip_get()

40.4 Unsupported API

APR::SockAddr also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

40.4.1 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);
```

- **obj: $addr1 (APR::SockAddr object)**

  One of the APR socket addresses.

- **arg1: $addr2 (APR::SockAddr object)**

  The other APR socket address.

- **ret: $ret (integer)**
- **since: subject to change**

The return value will be non-zero if the addresses are equivalent.

40.5 See Also

mod_perl 2.0 documentation.

40.6 Copyright

40.7 Authors

The mod_perl development team and numerous contributors.

41 APR::Socket - Perl API for APR sockets

41.1 Synopsis

```
use APR::Socket ();

### set the socket to the blocking mode if it isn't already
### and read in the loop and echo it back
use APR::Const -compile => qw(SO_NONBLOCK);
if ($sock->opt_get(APR::Const::SO_NONBLOCK)) {
    $sock->opt_set(APR::Const::SO_NONBLOCK => 0);
}
# read from/write to the socket (w/o handling possible failures)
my $wanted = 1024;
while ($sock->recv(my $buff, $wanted)) {
    $sock->send($buff);
}

### get/set IO timeout and try to read some data
use APR::Const -compile => qw(TIMEUP);
# timeout is in usecs!
my $timeout = $sock->timeout_get();
if ($timeout < 10_000_000) {
    $sock->timeout_set(20_000_000); # 20 secs
}
# now read, while handling timeouts
my $wanted = 1024;
my $buff;
my $rlen = eval { $sock->recv($buff, $wanted) };
if ($@ && ref $@ && $@ == APR::Const::TIMEUP) {
    # timeout, do something, e.g.
    warn "timed out, will try again later";
}
else {
    warn "asked for $wanted bytes, read $rlen bytes\n";
    # do something with the data
}

# non-blocking io poll
$sock->opt_set(APR::Const::SO_NONBLOCK => 1);
my $rc = $sock->poll($c->pool, 1_000_000, APR::Const::POLLIN);
if ($rc == APR::Const::SUCCESS) {
    # read the data
}

else {
    # handle the condition
}

# fetch the operating level socket
my $fd=$sock->fileno;
```

41.2 Description

`APR::Socket` provides the Perl interface to APR sockets.

41.3 API

`APR::Socket` provides the following methods:

41.3.1 `fileno`

Get the operating system socket, the file descriptor on UNIX.

```
$fd = $sock->fileno;
```

- **obj: `$sock` (`APR::Socket object`)**

  The socket

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

  The OS-level file descriptor.

- **since: 2.0.5 (not implemented on Windows)**

41.3.2 `opt_get`

Query socket options for the specified socket

```
$val = $sock->opt_get($opt);
```

- **obj: `$sock` (`APR::Socket object`)**

  the socket object to query

- **arg1: `$opt` (`APR::Const constant`)**

  the socket option we would like to configure. Here are the available socket options.

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

  the currently set value for the socket option you've queried for

- **excpt: `APR::Error`**
- **since: 2.0.00**

Examples can be found in the socket options constants section. For example setting the IO to the blocking mode.

41.3.3 opt_set

Setup socket options for the specified socket

```
$sock->opt_set($opt, $val);
```

- **obj: $sock (APR::Socket object object)**

  the socket object to set up.

- **arg1: $opt (APR::Const constant)**

  the socket option we would like to configure. Here are the available socket options.

- **arg2: $val (integer)**

  value for the option. Refer to the socket options section to learn about the expected values.

- **ret: no return value**
- **excpt: APR::Error**
- **since: 2.0.00**

Examples can be found in the socket options constants section. For example setting the IO to the blocking mode.

41.3.4 poll

Poll the socket for events:

```
$rc = $sock->poll($pool, $timeout, $events);
```

- **obj: $sock (APR::Socket object)**

  The socket to poll

- **arg1: $pool (APR::Pool object)**

  usually $c->pool.

- **arg2: $timeout (integer)**

  The amount of time to wait (in milliseconds) for the specified events to occur.

- **arg3: $events (APR::Const :poll constants)**

The events for which to wait.

For example use `APR::Const::POLLIN` to wait for incoming data to be available, `APR::Const::POLLOUT` to wait until it's possible to write data to the socket and `APR::Const::POLLPRI` to wait for priority data to become available.

- **ret: `$rc` (`APR::Const` constant)**

  If `APR::Const::SUCCESS` is received than the polling was successful. If not -- the error code is returned, which can be converted to the error string with help of `APR::Error::strerror`.

- **since: 2.0.00**

For example poll a non-blocking socket up to 1 second when reading data from the client:

```
use APR::Socket ();
use APR::Connection ();
use APR::Error ();

use APR::Const -compile => qw(SO_NONBLOCK POLLIN SUCCESS TIMEUP);

$sock->opt_set(APR::Const::SO_NONBLOCK => 1);

my $rc = $sock->poll($c->pool, 1_000_000, APR::Const::POLLIN);
if ($rc == APR::Const::SUCCESS) {
    # Data is waiting on the socket to be read.
    # $sock->recv(my $buf, BUFF_LEN)
}
elsif ($rc == APR::Const::TIMEUP) {
    # One second elapsed and still there is no data waiting to be
    # read. for example could try again.
}
else {
    die "poll error: " . APR::Error::strerror($rc);
}
```

41.3.5 `recv`

Read incoming data from the socket

```
$len = $sock->recv($buffer, $wanted);
```

- **obj: `$sock` (`APR::SockAddr` object object)**

  The socket to read from

- **arg1: `$buffer` (SCALAR)**

  The buffer to fill. All previous data will be lost.

- **arg2: `$wanted` (int)**

How many bytes to attempt to read.

- **ret: `$len` (number)**

  How many bytes were actually read.

  `$buffer` gets populated with the string that is read. It will contain an empty string if there was nothing to read.

- **excpt: `APR::Error`**

  If you get the `'(11) Resource temporarily unavailable'` error (exception `APR::Const::EAGAIN`) (or another equivalent, which might be different on non-POSIX systems), then you didn't ensure that the socket is in a blocking IO mode before using it. Note that you should use `APR::Status::is_EAGAIN` to perform this check (since different error codes may be returned for the same event on different OSes). For example if the socket is set to the non-blocking mode and there is no data right away, you may get this exception thrown. So here is how to check for it and retry a few times after short delays:

  ```
  use APR::Status ();
  $sock->opt_set(APR::Const::SO_NONBLOCK, 1);
  # ....
  my $tries = 0;
  my $buffer;
  RETRY: my $rlen = eval { $socket->recv($buffer, SIZE) };
  if ($@)
      die $@ unless ref $@ && APR::Status::is_EAGAIN($@);
      if ($tries++ < 3) {
          # sleep 250msec
          select undef, undef, undef, 0.25;
          goto RETRY;
      }
      else {
          # do something else
      }
  }
  warn "read $rlen bytes\n"
  ```

  If timeout was set via `timeout_set|/C_timeout_set_`, you may need to catch the `APR::Const::TIMEUP` exception. For example:

  ```
  use APR::Const -compile => qw(TIMEUP);
  $sock->timeout_set(1_000_000); # 1 sec
  my $buffer;
  eval { $sock->recv($buffer, $wanted) };
  if ($@ && $@ == APR::Const::TIMEUP) {
      # timeout, do something, e.g.
  }
  ```

  If not handled -- you may get the error `'70007: The timeout specified has expired'`.

Another error condition that may occur is the `'(104) Connection reset by peer'` error, which is up to your application logic to decide whether it's an error or not. This error usually happens when the client aborts the connection.

```
use APR::Const -compile => qw(ECONNABORTED);
my $buffer;
eval { $sock->recv($buffer, $wanted) };
if ($@ == APR::Const::ECONNABORTED) {
    # ignore it or deal with it
}
```

- **since: 2.0.00**

Here is the quick prototype example, which doesn't handle any errors (mod_perl will do that for you):

```
use APR::Socket ();

# set the socket to the blocking mode if it isn't already
use APR::Const -compile => qw(SO_NONBLOCK);
if ($sock->opt_get(APR::Const::SO_NONBLOCK)) {
    $sock->opt_set(APR::Const::SO_NONBLOCK => 0);
}
# read from/write to the socket (w/o handling possible failures)
my $wanted = 1024;
while ($sock->recv(my $buffer, $wanted)) {
    $sock->send($buffer);
}
```

If you want to handle errors by yourself, the loop may look like:

```
use APR::Const -compile => qw(ECONNABORTED);
# ...
while (1) {
    my $buf;
    my $len = eval { $sock->recv($buf, $wanted) };
    if ($@) {
        # handle the error, e.g. to ignore aborted connections but
        # rethrow any other errors:
        if ($@ == APR::Const::ECONNABORTED) {
            # ignore
            last;
        }
        else {
            die $@; # retrow
        }
    }

    if ($len) {
        $sock->send($buffer);
    }
    else {
        last;
    }
}
```

41.3.6 send

Write data to the socket

```
$wlen = $sock->send($buf, $opt_len);
```

- **obj: $sock (APR::Socket object)**

  The socket to write to

- **arg1: $buf (scalar)**

  The data to send

- **opt arg2: $opt_len (int)**

  There is no need to pass this argument, unless you want to send less data than contained in $buf.

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

  How many bytes were sent

- **since: 2.0.00**

For examples see the recv item.

41.3.7 timeout_get

Get socket timeout settings

```
$usecs = $sock->timeout_get();
```

- **obj: $sock (APR::Socket object)**

  The socket to set up.

- **ret: $usecs (number)**

  Currently set timeout in microseconds (and also the blocking IO behavior). See (APR::timeout_set) for possible values and their meaning.

- **excpt: APR::Error**
- **since: 2.0.00**

41.3.8 `timeout_set`

Setup socket timeout.

```
$sock->timeout_set($usecs);
```

- **obj: `$sock` (`APR::Socket object`)**

  The socket to set up.

- **arg1: `$usecs` (number)**

  Value for the timeout in microseconds and also the blocking IO behavior.

  The possible values are:

  - **t > 0**

    `send()` and `recv()` throw (`APR::Const::TIMEUP` exception) if specified time elapses with no data sent or received.

    Notice that the positive value is in micro seconds. So if you want to set the timeout for 5 seconds, the value should be: 5_000_000.

    This mode sets the socket into a non-blocking IO mode.

  - **t == 0**

    `send()` and `recv()` calls never block.

  - **t < 0**

    `send()` and `recv()` calls block.

    Usually just -1 is used for this case, but any negative value will do.

    This mode sets the socket into a blocking IO mode.

  - **ret: no return value**

- **excpt: `APR::Error`**
- **since: 2.0.00**

41.4 Unsupported API

`APR::Socket` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

41.4.1 bind

META: Autogenerated - needs to be reviewed/completed

Bind the socket to its associated port

```
$ret = $sock->bind($sa);
```

- **obj: $sock (APR::Socket object)**

  The socket to bind

- **arg1: $sa (APR::SockAddr object)**

  The socket address to bind to

- **ret: $ret (integer)**
- **since: subject to change**

This may be where we will find out if there is any other process using the selected port.

41.4.2 close

META: Autogenerated - needs to be reviewed/completed

Close a socket.

```
$ret = $sock->close();
```

- **obj: $sock (APR::Socket object)**

  The socket to close

- **ret: $ret (integer)**
- **since: subject to change**

41.4.3 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);
```

- **obj: $sock (APR::Socket object)**

  The socket we wish to use for our side of the connection

- **arg1: `$sa` (`APR::SockAddr object`)**

  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)**
- **since: subject to change**

41.4.4 `listen`

META: Autogenerated - needs to be reviewed/completed

Listen to a bound socket for connections.

```
$ret = $sock->listen($backlog);
```

- **obj: `$sock` (`APR::Socket object`)**

  The socket to listen on

- **arg1: `$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)**
- **since: subject to change**

41.4.5 `recvfrom`

META: Autogenerated - needs to be reviewed/completed

```
$ret = $from->recvfrom($sock, $flags, $buf, $len);
```

- **obj: `$from` (`APR::SockAddr object`)**

  The apr_sockaddr_t to fill in the recipient info

- **arg1: `$sock` (`APR::SockAddr object`)**

  The socket to use

- **arg2: `$flags` (integer)**

  The flags to use

- **arg3: `$buf` (integer)**

The buffer to use

- **arg4: `$len` (string)**

  The length of the available buffer

- **ret: `$ret` (integer)**
- **since: subject to change**

41.4.6 `sendto`

META: Autogenerated - needs to be reviewed/completed

```
$ret = $sock->sendto($where, $flags, $buf, $len);
```

- **obj: `$sock` (`APR::Socket object`)**

  The socket to send from

- **arg1: `$where` (`APR::Socket object`)**

  The apr_sockaddr_t describing where to send the data

- **arg2: `$flags` (integer)**

  The flags to use

- **arg3: `$buf` (scalar)**

  The data to send

- **arg4: `$len` (string)**

  The length of the data to send

- **ret: `$ret` (integer)**
- **since: subject to change**

41.5 See Also

mod_perl 2.0 documentation.

41.6 Copyright

41.7 Authors

The mod_perl development team and numerous contributors.

42 APR::Status - Perl Interface to the APR_STATUS_IS_* macros

42.1 Synopsis

```
use APR::Status ();
eval { $obj->mp_method() };
if ($@ && $ref $@ eq 'APR::Error' && APR::Status::is_EAGAIN($@)) {
    # APR_STATUS_IS_EAGAIN(s) of apr_errno.h is satisfied
}
```

42.2 Description

An interface to *apr_errno.h* composite error codes.

As discussed in the `APR::Error` manpage, it is possible to handle APR/Apache/mod_perl exceptions in the following way:

```
eval { $obj->mp_method() };
if ($@ && $ref $@ eq 'APR::Error' && $@ == $some_code)
    warn "handled exception: $@";
}
```

However, in cases where `$some_code` is an APR::Const constant, there may be more than one condition satisfying the intent of this exception. For this purpose the APR C library provides in *apr_errno.h* a series of macros, `APR_STATUS_IS_*`, which are the recommended way to check for such conditions. For example, the `APR_STATUS_IS_EAGAIN` macro is defined as

```
#define APR_STATUS_IS_EAGAIN(s)         ((s) == APR_EAGAIN \
                || (s) == APR_OS_START_SYSERR + ERROR_NO_DATA \
                || (s) == APR_OS_START_SYSERR + SOCEWOULDBLOCK \
                || (s) == APR_OS_START_SYSERR + ERROR_LOCK_VIOLATION)
```

The purpose of `APR::Status` is to provide functions corresponding to these macros.

42.3 Functions

42.3.1 `is_EACCES`

Check if the error is matching `EACCES` and its variants (corresponds to the `APR_STATUS_IS_EACCES` macro).

```
$status = APR::Status::is_EACCES($error_code);
```

- **arg1: `$error_code` (integer or `APR::Error object`)**

  The error code or to check, normally `$@` blessed into `APR::Error object`.

- **ret: `$status` (boolean)**
- **since: 2.0.00**

An example of using is_EACCES is when reading the contents of a file where access may be forbidden:

```
eval { $obj->slurp_filename(0) };
if ($@) {
    return Apache2::Const::FORBIDDEN
        if ref $@ eq 'APR::Error' && APR::Status::is_EACCES($@);
    die $@;
 }
```

Due to possible variants in conditions matching EACCES, the use of this function is recommended for checking error codes against this value, rather than just using APR::Const::EACCES directly.

42.3.2 is_EAGAIN

Check if the error is matching EAGAIN and its variants (corresponds to the APR_STATUS_IS_EAGAIN macro).

```
$status = APR::Status::is_EAGAIN($error_code);
```

- **arg1: $error_code (integer or APR::Error object)**

  The error code or to check, normally $@ blessed into APR::Error object.

- **ret: $status (boolean)**
- **since: 2.0.00**

For example, here is how you may want to handle socket read exceptions and do retries:

```
use APR::Status ();
# ....
my $tries = 0;
my $buffer;
RETRY: my $rlen = eval { $socket->recv($buffer, SIZE) };
if ($@ && ref($@) && APR::Status::is_EAGAIN($@)) {
    if ($tries++ < 3) {
        goto RETRY;
    }
    else {
        # do something else
    }
}
else {
    die "eval block has failed: $@";
}
```

Notice that just checking against APR::Const::EAGAIN may work on some Unices, but then it will certainly break on win32. Thefore make sure to use this macro and not APR::Const::EAGAIN unless you know what you are doing.

42.3.3 is_ENOENT

Check if the error is matching ENOENT and its variants (corresponds to the APR_STATUS_IS_ENOENT macro).

```
$status = APR::Status::is_ENOENT($error_code);
```

- **arg1: $error_code (integer or APR::Error object)**

  The error code or to check, normally $@ blessed into APR::Error object.

- **ret: $status (boolean)**
- **since: 2.0.00**

An example of using is_ENOENT is when reading the contents of a file which may not exist:

```
eval { $obj->slurp_filename(0) };
if ($@) {
    return Apache2::Const::NOT_FOUND
        if ref $@ eq 'APR::Error' && APR::Status::is_ENOENT($@);
    die $@;
}
```

Due to possible variants in conditions matching ENOENT, the use of this function is recommended for checking error codes against this value, rather than just using APR::Const::ENOENT directly.

42.3.4 is_EOF

Check if the error is matching EOF and its variants (corresponds to the APR_STATUS_IS_EOF macro).

```
$status = APR::Status::is_EOF($error_code);
```

- **arg1: $error_code (integer or APR::Error object)**

  The error code or to check, normally $@ blessed into APR::Error object.

- **ret: $status (boolean)**
- **since: 2.0.00**

Due to possible variants in conditions matching EOF, the use of this function is recommended for checking error codes against this value, rather than just using APR::Const::EOF directly.

42.3.5 is_ECONNABORTED

Check if the error is matching ECONNABORTED and its variants (corresponds to the APR_STATUS_IS_ECONNABORTED macro).

```
$status = APR::Status::is_ECONNABORTED($error_code);
```

- **arg1: `$error_code` (integer or `APR::Error object`)**

  The error code or to check, normally `$@` blessed into `APR::Error object`.

- **ret: `$status` (boolean)**
- **since: 2.0.00**

Due to possible variants in conditions matching `ECONNABORTED`, the use of this function is recommended for checking error codes against this value, rather than just using `APR::Const::ECONNABORTED` directly.

42.3.6 `is_ECONNRESET`

Check if the error is matching `ECONNRESET` and its variants (corresponds to the `APR_STATUS_IS_ECONNRESET` macro).

```
$status = APR::Status::is_ECONNRESET($error_code);
```

- **arg1: `$error_code` (integer or `APR::Error object`)**

  The error code or to check, normally `$@` blessed into `APR::Error object`.

- **ret: `$status` (boolean)**
- **since: 2.0.00**

Due to possible variants in conditions matching `ECONNRESET`, the use of this function is recommended for checking error codes against this value, rather than just using `APR::Const::ECONNRESET` directly.

42.3.7 `is_TIMEUP`

Check if the error is matching `TIMEUP` and its variants (corresponds to the `APR_STATUS_IS_TIMEUP` macro).

```
$status = APR::Status::is_TIMEUP($error_code);
```

- **arg1: `$error_code` (integer or `APR::Error object`)**

  The error code or to check, normally `$@` blessed into `APR::Error object`.

- **ret: `$status` (boolean)**
- **since: 2.0.00**

Due to possible variants in conditions matching `TIMEUP`, the use of this function is recommended for checking error codes against this value, rather than just using `APR::Const::TIMEUP` directly.

42.4 See Also

mod_perl 2.0 documentation.

42.5 Copyright

42.6 Authors

The mod_perl development team and numerous contributors.

43 APR::String - Perl API for manipulating APR UUIDs

43.1 Synopsis

```
use APR::String ();

# 42_000 => " 41K",
my $size_str = APR::String::format_size($size);
```

43.2 Description

`APR::String` provides strings manipulation API.

43.3 API

`APR::String` provides the following functions and/or methods:

43.3.1 `format_size`

```
my $size_str = APR::String::format_size($size);
```

- **arg1: `$size` (integer)**
- **ret: `$size_str`**

  returns a formatted size string representation of a number. The size given in the string will be in units of bytes, kilobytes, or megabytes, depending on the size. The length of that string is always 4 chars long. For example:

  ```
  0               => "  0 ",
  42              => " 42 ",
  42_000          => " 41K",
  42_000_000      => " 40M",
  ```

- **since: 2.0.00**

43.4 See Also

mod_perl 2.0 documentation.

43.5 Copyright

43.6 Authors

The mod_perl development team and numerous contributors.

44 APR::Table - Perl API for manipulating APR opaque string-content tables

44.1 Synopsis

```
use APR::Table ();

$table = APR::Table::make($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);

use APR::Const -compile qw(:table);
$table_overlay = $table_base->overlay($table_overlay, $pool);
$table_overlay->compress(APR::Const::OVERLAP_TABLES_MERGE);

$table_a->overlap($table_b, APR::Const::OVERLAP_TABLES_SET);

$table->do(sub {print "key $_[0], value $_[1]\n"}, @valid_keys);

#Tied Interface
$value = $table->{$key};
$table->{$key} = $value;
print "got it" if exists $table->{$key};

foreach my $key (keys %{$table}) {
    print "$key = $table->{$key}\n";
}
```

44.2 Description

`APR::Table` allows its users to manipulate opaque string-content tables.

On the C level the "opaque string-content" means: you can put in ’\0’-terminated strings and whatever you put in your get out.

On the Perl level that means that we convert scalars into strings and store those strings. Any special information that was in the Perl scalar is not stored. So for example if a scalar was marked as utf8, tainted or tied, that information is not stored. When you get the data back as a Perl scalar you get only the string.

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.

44.3 API

`APR::Table` provides the following functions and/or methods:

44.3.1 `add`

Add data to a table, regardless of whether there is another element with the same key.

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

- **obj: `$table` (`APR::Table object`)**

  The table to add to.

- **arg1: `$key` (string)**

  The key to use.

- **arg2: `$val` (string)**

  The value to add.

- **ret: no return value**
- **since: 2.0.00**

When adding data, this function makes a copy of both the key and the value.

44.3.2 `clear`

Delete all of the elements from a table.

```
$table->clear();
```

- **obj: `$table` (`APR::Table object`)**

  The table to clear.

- **ret: no return value**
- **since: 2.0.00**

44.3.3 compress

Eliminate redundant entries in a table by either overwriting or merging duplicates:

```
$table->compress($flags);
```

- **obj: `$table` (`APR::Table object`)**

  The table to compress.

- **arg1: `$flags` (`APR::Const constant`)**

  ```
  APR::Const::OVERLAP_TABLES_MERGE -- to merge
  APR::Const::OVERLAP_TABLES_SET   -- to overwrite
  ```

- **ret: no return value**
- **since: 2.0.00**

Converts multi-valued keys in `$table` into single-valued keys. This function takes duplicate table entries and flattens them into a single entry. The flattening behavior is controlled by the (mandatory) `$flags` argument.

When `$flags` == `APR::Const::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::Const::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.

Access the constants via:

```
use APR::Const -compile qw(:table);
```

or an explicit:

```
use APR::Const -compile qw(OVERLAP_TABLES_SET OVERLAP_TABLES_MERGE);
```

`compress()` combined with `overlay()` does the same thing as `overlap()`.

Examples:

- **`APR::Const::OVERLAP_TABLES_SET`**

  Start with table `$table`:

  ```
  foo => "one"
  foo => "two"
  foo => "three"
  bar => "beer"
  ```

which is done by:

```
use APR::Const    -compile => ':table';
my $table = APR::Table::make($r->pool, TABLE_SIZE);

$table->set(bar => 'beer');
$table->set(foo => 'one');
$table->add(foo => 'two');
$table->add(foo => 'three');
```

Now compress it using `APR::Const::OVERLAP_TABLES_SET`:

```
$table->compress(APR::Const::OVERLAP_TABLES_SET);
```

Now table `$table` contains:

```
foo => "three"
bar => "beer"
```

The value *three* for the key *foo*, that was added last, took over the other values.

- **`APR::Const::OVERLAP_TABLES_MERGE`**

  Start with table `$table`:

  ```
  foo => "one"
  foo => "two"
  foo => "three"
  bar => "beer"
  ```

  as in the previous example, now compress it using `APR::Const::OVERLAP_TABLES_MERGE`:

  ```
  $table->compress(APR::Const::OVERLAP_TABLES_MERGE);
  ```

  Now table `$table` contains:

  ```
  foo => "one, two, three"
  bar => "beer"
  ```

  All the values for the same key were merged into one value.

44.3.4 copy

Create a new table and copy another table into it.

```
$table_copy = $table->copy($p);
```

- **obj: `$table` (`APR::Table object`)**

  The table to copy.

- **arg1: `$p` (`APR::Pool object`)**

  The pool to allocate the new table out of.

- **ret: `$table_copy` (`APR::Table object`)**

  A copy of the table passed in.

- **since: 2.0.00**

44.3.5 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);
```

- **obj: `$table` (`APR::Table object`)**

  The table to operate on.

- **arg1: `$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 f`@filter` will be processed.

- **ret: no return value**
- **since: 2.0.00**

Examples:

- This filter simply prints out the key/value pairs and counts how many pairs did it see.

```
use constant TABLE_SIZE => 20;
our $filter_count;
my $table = APR::Table::make($r->pool, TABLE_SIZE);

# populate the table with ascii data
for (1..TABLE_SIZE) {
    $table->set(chr($_+97), $_);
}

$filter_count = 0;
$table->do("my_filter");
print "Counted $filter_count elements";

sub my_filter {
    my ($key, $value) = @_;
```

```
        warn "$key => $value\n";
        $filter_count++;
        return 1;
    }
```

Notice that `my_filter` always returns 1, ensuring that `do()` will pass all the key/value pairs.

- This filter is similar to the one from the previous example, but this time it decides to abort the filtering after seeing half of the table, by returning 0 when this happens.

```
sub my_filter {
    my ($key, $value) = @_;
    $filter_count++;
    return $filter_count == int(TABLE_SIZE)/2 ? 0 : 1;
}
```

44.3.6 `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);
```

- **obj: `$table` (`APR::Table object`)**

  The table to search for the key.

- **arg1: `$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). If nothing matches `undef` is returned.

  In the list context the whole table is traversed and all matching values are returned. An empty list is returned if nothing matches.

- **since: 2.0.00**

44.3.7 `make`

Make a new table.

```
$table = APR::Table::make($p, $nelts);
```

- **obj: `$p` (`APR::Pool object`)**

The pool to allocate the pool out of.

- **arg1: `$nelts` (integer)**

  The number of elements in the initial table. At least 1 or more. If 0 is passed APR will still allocate 1.

- **ret: `$table` (`APR::Table object`)**

  The new table.

- **since: 2.0.00**

This table can only store text data.

44.3.8 `merge`

Add data to a table by merging the value with data that has already been stored using ", " as a separator:

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

- **obj: `$table` (`APR::Table object`)**

  The table to search for the data.

- **arg1: `$key` (string)**

  The key to merge data for.

- **arg2: `$val` (string)**

  The data to add.

- **ret: no return value**
- **since: 2.0.00**

If the key is not found, then this function acts like `add()`.

If there is more than one value for the same key, only the first (the oldest) value gets merged.

Examples:

- Start with a pair:

  ```
  merge => "1"
  ```

  and merge "a" to the value:

  ```
  $table->set(  merge => '1');
  $table->merge(merge => 'a');
  $val = $table->get('merge');
  ```

Result:

```
$val == "1, a";
```

- Start with a multivalued pair:

  ```
  merge => "1"
  merge => "2"
  ```

  and merge "a" to the first value;

  ```
  $table->set(  merge => '1');
  $table->add(  merge => '2');
  $table->merge(merge => 'a');
  @val = $table->get('merge');
  ```

  Result:

  ```
  $val[0] == "1, a";
  $val[1] == "2";
  ```

  Only the first value for the same key is affected.

- Have no entry and merge "a";

  ```
  $table->merge(miss => 'a');
  $val = $table->get('miss');
  ```

  Result:

  ```
  $val == "a";
  ```

44.3.9 overlap

For each key/value pair in `$table_b`, add the data to `$table_a`. The definition of `$flags` explains how `$flags` define the overlapping method.

```
$table_a->overlap($table_b, $flags);
```

- **obj: `$table_a` (`APR::Table object`)**

  The table to add the data to.

- **arg1: `$table_b` (`APR::Table object`)**

  The table to iterate over, adding its data to table `$table_a`

- **arg2: `$flags` (integer)**

  How to add the table to table `$table_a`.

When `$flags == APR::Const::OVERLAP_TABLES_SET`, if another element already exists with the same key, this will over-write the old data.

When `$flags == APR::Const::OVERLAP_TABLES_MERGE`, the key/value pair from `$table_b` is added, regardless of whether there is another element with the same key in `$table_a`.

- **ret: no return value**
- **since: 2.0.00**

Access the constants via:

```
use APR::Const -compile qw(:table);
```

or an explicit:

```
use APR::Const -compile qw(OVERLAP_TABLES_SET OVERLAP_TABLES_MERGE);
```

This function is highly optimized, and uses less memory and CPU cycles than a function that just loops through table `$table_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-E<gt>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 `$table_b` has many elements.

Notice the assumptions on the keys and values in `$table_b` -- they must be in an ancestor of `$table_a`'s pool. In practice `$table_b` and `$table_a` are usually from the same pool.

Examples:

- **`APR::Const::OVERLAP_TABLES_SET`**

  Start with table `$base`:

  ```
  foo => "one"
  foo => "two"
  bar => "beer"
  ```

and table $add:

```
foo => "three"
```

which is done by:

```
use APR::Const    -compile => ':table';
my $base = APR::Table::make($r->pool, TABLE_SIZE);
my $add  = APR::Table::make($r->pool, TABLE_SIZE);

$base->set(bar => 'beer');
$base->set(foo => 'one');
$base->add(foo => 'two');

$add->set(foo => 'three');
```

Now overlap using APR::Const::OVERLAP_TABLES_SET:

```
$base->overlap($add, APR::Const::OVERLAP_TABLES_SET);
```

Now table $add is unmodified and table $base contains:

```
foo => "three"
bar => "beer"
```

The value from table add has overwritten all previous values for the same key both had (*foo*). This is the same as doing overlay() followed by compress() with APR::Const::OVERLAP_TABLES_SET.

- **APR::Const::OVERLAP_TABLES_MERGE**

  Start with table $base:

  ```
  foo => "one"
  foo => "two"
  ```

  and table $add:

  ```
  foo => "three"
  bar => "beer"
  ```

  which is done by:

  ```
  use APR::Const    -compile => ':table';
  my $base = APR::Table::make($r->pool, TABLE_SIZE);
  my $add  = APR::Table::make($r->pool, TABLE_SIZE);

  $base->set(foo => 'one');
  $base->add(foo => 'two');

  $add->set(foo => 'three');
  $add->set(bar => 'beer');
  ```

Now overlap using `APR::Const::OVERLAP_TABLES_MERGE`:

```
$base->overlap($add, APR::Const::OVERLAP_TABLES_MERGE);
```

Now table `$add` is unmodified and table `$base` contains:

```
foo => "one, two, three"
bar => "beer"
```

Values from both tables for the same key were merged into one value. This is the same as doing `overlay()` followed by `compress()` with `APR::Const::OVERLAP_TABLES_MERGE`.

44.3.10 `overlay`

Merge two tables into one new table. The resulting table may have more than one value for the same key.

```
$table = $table_base->overlay($table_overlay, $p);
```

- **obj: `$table_base` (`APR::Table object`)**

  The table to add at the end of the new table.

- **arg1: `$table_overlay` (`APR::Table object`)**

  The first table to put in the new table.

- **arg2: `$p` (`APR::Pool object`)**

  The pool to use for the new table.

- **ret: `$table` (`APR::Table object`)**

  A new table containing all of the data from the two passed in.

- **since: 2.0.00**

Examples:

- Start with table `$base`:

  ```
  foo => "one"
  foo => "two"
  bar => "beer"
  ```

  and table `$add`:

  ```
  foo => "three"
  ```

  which is done by:

```
use APR::Const    -compile => ':table';
my $base = APR::Table::make($r->pool, TABLE_SIZE);
my $add  = APR::Table::make($r->pool, TABLE_SIZE);

$base->set(bar => 'beer');
$base->set(foo => 'one');
$base->add(foo => 'two');

$add->set(foo => 'three');
```

Now overlay using `APR::Const::OVERLAP_TABLES_SET`:

```
my $overlay = $base->overlay($add, APR::Const::OVERLAP_TABLES_SET);
```

That resulted in a new table `$overlay` (tables `add` and `$base` are unmodified) which contains:

```
foo => "one"
foo => "two"
foo => "three"
bar => "beer"
```

44.3.11 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.

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

- **obj: `$table` (`APR::Table object`)**

  The table to add the data to.

- **arg1: `$key` (string)**

  The key to use.

- **arg2: `$val` (string)**

  The value to add.

- **ret: no return value**
- **since: 2.0.00**

When adding data, this function makes a copy of both the key and the value.

44.3.12 unset

Remove data from the table.

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

- **obj: `$table` (`APR::Table object`)**

  The table to remove data from.

- **arg1: `$key` (string)**

  The key of the data being removed.

- **ret: no return value**
- **since: 2.0.00**

44.4 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`.

Note regarding the use of `values()`. `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. With Perl 5.8.0 and higher `values()` will correctly list values the corresponding to the list generated by `keys()`. That doesn't work with Perl 5.6. Therefore to portably iterate over the key-value pairs, use `each()` (which fully supports multivalued keys), or `APR::Table::do`.

44.4.1 EXISTS

```
$ret = $table->EXISTS($key);
```

- **obj: `$table` (`APR::Table object`)**
- **arg1: `$key` (string)**
- **ret: `$ret` (integer)**

  true or false

- **since: 2.0.00**

44.4.2 CLEAR

```
$table->CLEAR();
```

- **obj: `$table` (`APR::Table object`)**
- **ret: no return value**
- **since: 2.0.00**

44.4.3 STORE

```
$table->STORE($key, $val);
```

- **obj: `$table` (`APR::Table object`)**
- **arg1: `$key` (string)**
- **arg2: `$val` (string)**
- **ret: no return value**
- **since: 2.0.00**

44.4.4 DELETE

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

- **obj: `$table` (`APR::Table object`)**
- **arg1: `$key` (string)**
- **ret: no return value**
- **since: 2.0.00**

44.4.5 FETCH

```
$ret = $table->FETCH($key);
```

- **obj: `$table` (`APR::Table object`)**
- **arg1: `$key` (string)**
- **ret: `$ret` (string)**
- **since: 2.0.00**

When iterating through the table's entries with `each()`, `FETCH` will return the current value of a multi-valued key. For example:

```
$table->add("a" => 1);
$table->add("b" => 2);
$table->add("a" => 3);

($k, $v) = each %$table; # (a, 1)
print $table->{a};       # prints 1

($k, $v) = each %$table; # (b, 2)
print $table->{a};       # prints 1

($k, $v) = each %$table; # (a, 3)
print $table->{a};       # prints 3 !!!

($k, $v) = each %$table; # (undef, undef)
print $table->{a};       # prints 1
```

44.5 See Also

mod_perl 2.0 documentation.

44.6 Copyright

44.7 Authors

The mod_perl development team and numerous contributors.

45 APR::ThreadMutex - Perl API for APR thread mutexes

45.1 Synopsis

```
use APR::ThreadMutex ();

my $mutex = APR::ThreadMutex->new($r->pool);
$mutex->lock;
$mutex->unlock;
$mutex->trylock;
```

45.2 Description

`APR::ThreadMutex` interfaces APR thread mutexes.

45.3 API

`APR::ThreadMutex` provides the following functions and/or methods:

45.4 Unsupported API

`APR::ThreadMutex` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

45.4.1 DESTROY

META: Autogenerated - needs to be reviewed/completed

Destroy the mutex and free the memory associated with the lock.

```
$mutex->DESTROY();
```

- **obj: `$mutex` (`APR::ThreadMutex object`)**

  the mutex to destroy.

- **ret: no return value**
- **since: subject to change**

45.4.2 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();
```

- **obj: `$mutex` (`APR::ThreadMutex object`)**

  the mutex on which to acquire the lock.

- **ret: `$ret` (integer)**
- **since: subject to change**

45.4.3 new

Create a new mutex

```
my $mutex = APR::ThreadMutex->new($p);
```

- **obj: `APR::ThreadMutex` (class name)**
- **arg1: `$p` (`APR::Pool object`)**
- **ret: `$mutex` (`APR::ThreadMutex object`)**
- **since: subject to change**

45.4.4 pool_get

META: Autogenerated - needs to be reviewed/completed

META: should probably be renamed to pool(), like all other pool accessors

Get the pool used by this thread_mutex.

```
$ret = $obj->pool_get();
```

- **obj: `$obj` (`APR::ThreadMutex object`)**
- **ret: `$ret` (`APR::Pool object`)**

  apr_pool_t the pool

- **since: subject to change**

45.4.5 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();
```

- **obj: `$mutex` (`APR::ThreadMutex object`)**

  the mutex on which to attempt the lock acquiring.

- **ret: `$ret` (integer)**
- **since: subject to change**

45.4.6 `unlock`

META: Autogenerated - needs to be reviewed/completed

Release the lock for the given mutex.

```
$ret = $mutex->unlock();
```

- **obj: `$mutex` (`APR::ThreadMutex object`)**

  the mutex from which to release the lock.

- **ret: `$ret` (integer)**
- **since: subject to change**

45.5 See Also

mod_perl 2.0 documentation.

45.6 Copyright

45.7 Authors

The mod_perl development team and numerous contributors.

46 APR::ThreadRWLock - Perl API for APR thread read/write locks

46.1 Synopsis

```
use APR::ThreadRWLock ();

my $mutex = APR::ThreadRWLock->new($r->pool);
$mutex->rdlock;
$mutex->wrlock;
$mutex->tryrdlock;
$mutex->trywrlock;
$mutex->unlock;
```

46.2 Description

`APR::ThreadRWLock` interfaces APR thread read/write locks.

See *srclib/apr/locks/unix/thread_rwlock.c* in your Apache source tree. At the time of this writing these methods are not supported on all platforms. Thus, check your libraries!

46.3 API

`APR::ThreadRWLock` provides the following functions and/or methods:

46.4 Unsupported API

`APR::ThreadRWLock` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

46.4.1 DESTROY

META: Autogenerated - needs to be reviewed/completed

Destroy the lock and free the associated memory.

```
$lock->DESTROY();
```

- **obj: `$lock` (`APR::ThreadRWLock object`)**

  the lock to destroy.

- **ret: no return value**
- **since: subject to change**

46.4.2 `rdlock`

META: Autogenerated - needs to be reviewed/completed

Acquire the read lock for the given lock. The calling thread acquires the read lock if a writer does not hold the lock and there are no writers blocked on the lock. Otherwize it is put to sleep until these conditions are met.

```
$ret = $lock->rdlock();
```

- **obj: `$lock` (`APR::ThreadRWLock object`)**

  the lock on which to acquire the lock.

- **ret: `$ret` (integer)**
- **since: subject to change**

46.4.3 `tryrdlock`

META: Autogenerated - needs to be reviewed/completed

Performs the same operation as `rdlock` with the exception that the function shall fail if the thread would be blocked.

```
$ret = $lock->tryrdlock();
```

- **obj: `$lock` (`APR::ThreadRWLock object`)**

  the lock on which to acquire the lock.

- **ret: `$ret` (integer)**
- **since: subject to change**

46.4.4 `wrlock`

META: Autogenerated - needs to be reviewed/completed

Acquire the write lock for the given lock. The calling thread acquires the write lock if if no other thread (reader or writer) holds it. Otherwize it is put to sleep until this condition is met.

```
$ret = $lock->wrlock();
```

- **obj: `$lock` (`APR::ThreadRWLock object`)**

  the lock on which to acquire the lock.

- **ret: `$ret` (integer)**
- **since: subject to change**

46.4.5 trywrlock

META: Autogenerated - needs to be reviewed/completed

Performs the same operation as `wrlock` with the exception that the function shall fail if the thread would be blocked.

```
$ret = $lock->trywrlock();
```

- **obj: `$lock` (`APR::ThreadRWLock object`)**

  the lock on which to acquire the lock.

- **ret: `$ret` (integer)**
- **since: subject to change**

46.4.6 new

Create a new lock

```
my $lock = APR::ThreadRWLock->new($p);
```

- **obj: `APR::ThreadRWLock` (class name)**
- **arg1: `$p` (`APR::Pool object`)**
- **ret: `$lock` (`APR::ThreadRWLock object`)**
- **since: subject to change**

46.4.7 pool_get

META: Autogenerated - needs to be reviewed/completed

META: should probably be renamed to pool(), like all other pool accessors

Get the pool used by this thread_lock.

```
$ret = $obj->pool_get();
```

- **obj: `$obj` (`APR::ThreadRWLock object`)**
- **ret: `$ret` (`APR::Pool object`)**

  apr_pool_t the pool

- **since: subject to change**

46.4.8 `unlock`

META: Autogenerated - needs to be reviewed/completed

Release the lock for the given lock.

```
$ret = $lock->unlock();
```

- **obj: `$lock` (`APR::ThreadRWLock object`)**

  the lock from which to release the lock.

- **ret: `$ret` (integer)**
- **since: subject to change**

46.5 See Also

mod_perl 2.0 documentation.

46.6 Copyright

46.7 Authors

The mod_perl development team and numerous contributors.

47 APR::URI - Perl API for URI manipulations

47.1 Synopsis

```
use APR::URI ();

my $url = 'http://user:pass@example.com:80/foo?bar#item5';

# parse and break the url into components
my $parsed = APR::URI->parse($r->pool, $url);
print $parsed->scheme;
print $parsed->user;
print $parsed->password;
print $parsed->hostname;
print $parsed->port;
print $parsed->path;
print $parsed->rpath;
print $parsed->query;
print $parsed->fragment;

# reconstruct the url, after changing some components and completely
# removing other
$parsed->scheme($new_scheme);
$parsed->user(undef);
$parsed->password(undef);
$parsed->hostname($new_hostname);
$parsed->port($new_port);
$parsed->path($new_path);
$parsed->query(undef);
$parsed->fragment(undef);
print $parsed->unparse;

# get the password field too (by default it's not revealed)
use APR::Const -compile => qw(URI_UNP_REVEALPASSWORD);
print $parsed->unparse(APR::Const::URI_UNP_REVEALPASSWORD);

# what the default port for the ftp protocol?
my $ftp_port = APR::URI::port_of_scheme("ftp");
```

47.2 Description

`APR::URI` allows you to parse URI strings, manipulate each of the URI elements and deparse them back into URIs.

All `APR::URI` object accessors accept a string or an `undef` value as an argument. Same goes for return value. It's important to distinguish between an empty string and `undef`. For example let's say your code was:

```
my $uri = 'http://example.com/foo?bar#item5';
my $parsed = APR::URI->parse($r->pool, $uri);
```

Now you no longer want to the query and fragment components in the final url. If you do:

```
$parsed->fragment('');
$parsed->query('');
```

followed by:

```
my $new_uri = parsed->unparse;
```

the resulting URI will be:

```
http://example.com/foo?#
```

which is probably not something that you've expected. In order to get rid of the separators, you must completely unset the fields you don't want to see. So, if you do:

```
$parsed->fragment(undef);
$parsed->query(undef);
```

followed by:

```
my $new_uri = parsed->unparse;
```

the resulting URI will be:

```
 http://example.com/foo
```

As mentioned earlier the same goes for return values, so continuing this example:

```
my $new_fragment = $parsed->fragment();
my $new_query    = $parsed->query();
```

Both values now contain `undef`, therefore you must be careful when using the return values, when you use them, as you may get warnings.

Also make sure you read through `the unparse() section` as various optional flags affect how the deparsed URI is rendered.

47.3 API

`APR::URI` provides the following functions and/or methods:

47.3.1 `fragment`

Get/set trailing "#fragment" string

```
$oldval = $parsed->fragment($newval);
```

- **obj: `$parsed` (`APR::URI object`)**
- **opt arg1: `$newval` (string or undef)**
- **ret: `$oldval` (string or undef)**
- **since: 2.0.00**

47.3.2 `hostinfo`

Get/set combined `[user[:password]@]host[:port]`

```
$oldval = $parsed->hostinfo($newval);
```

- **obj: `$parsed` (`APR::URI object`)**
- **opt arg1: `$newval` (string or undef)**
- **ret: `$oldval` (string or undef)**
- **since: 2.0.00**

The `hostinfo` value is set automatically when `parse()` is called.

It's not updated if any of the individual fields is modified.

It's not used when `unparse()` is called.

47.3.3 `hostname`

Get/set hostname

```
$oldval = $parsed->hostname($newval);
```

- **obj: `$parsed` (`APR::URI object`)**
- **opt arg1: `$newval` (string or undef)**
- **ret: `$oldval` (string or undef)**
- **since: 2.0.00**

47.3.4 `password`

Get/set password (as in http://user:password@host:port/)

```
$oldval = $parsed->password($newval);
```

- **obj: `$parsed` (`APR::URI object`)**
- **opt arg1: `$newval` (string or undef)**
- **ret: `$oldval` (string or undef)**
- **since: 2.0.00**

47.3.5 `parse`

Parse the URI string into URI components

```
$parsed = APR::URI->parse($pool, $uri);
```

- **obj: `$parsed` (`APR::URI object or class`)**
- **arg1: `$pool` (string) (`APR::Pool object`)**
- **arg2: `$uri` (string)**

  The URI to parse

- **ret: `$parsed` (`APR::URI object or class`)**

  The parsed URI object

- **since: 2.0.00**

After parsing, if a component existed but was an empty string (e.g. empty query *http://hostname/path?*) -- the corresponding accessor will return an empty string. If a component didn't exist (e.g. no query part *http://hostname/path*) -- the corresponding accessor will return `undef`.

47.3.6 `path`

Get/set the request path

```
$oldval = $parsed->path($newval);
```

- **obj: `$parsed` (`APR::URI object`)**
- **opt arg1: `$newval` (string or undef)**
- **ret: `$oldval` (string or undef)**

  `"/"` if only `scheme://host`

- **since: 2.0.00**

47.3.7 `rpath`

Gets the `path` minus the `path_info`

```
$rpath =  $parsed->rpath();
```

- **obj: `$parsed` (`APR::URI object`)**
- **opt arg1: `$newval` (string or undef)**
- **ret: `$oldval` (string or undef)**

  The path minus the *path_info*

- **since: 2.0.00**

47.3.8 `port`

Get/set port number

```
$oldval = $parsed->port($newval);
```

- **obj: `$parsed` (`APR::URI object`)**
- **opt arg1: `$newval` (number or string or undef)**
- **ret: `$oldval` (string or undef)**

  If the port component didn't appear in the parsed URI, APR internally calls `port_of_scheme()` to find out the port number for the given `scheme()`.

- **since: 2.0.00**

47.3.9 `port_of_scheme`

Return the default port for a given scheme. The recognized schemes are http, ftp, https, gopher, wais, nntp, snews and prospero.

```
$port = APR::URI::port_of_scheme($scheme);
```

- **obj: `$scheme` (string)**

  The scheme string

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

  The default port for this scheme

- **since: 2.0.00**

47.3.10 `query`

Get/set the query string (the part starting after `'?'` and all the way till the end or the `'#fragment'` part if the latter exists).

```
$oldval = $parsed->query($newval);
```

- **obj: `$parsed` (`APR::URI object`)**
- **opt arg1: `$newval` (string or undef)**
- **ret: `$oldval` (string or undef)**
- **since: 2.0.00**

47.3.11 scheme

Get/set the protocol scheme ("http", "ftp", ...)

```
$oldval = $parsed->scheme($newval);
```

- **obj: $parsed (APR::URI object)**
- **opt arg1: $newval (string or undef)**
- **ret: $oldval (string or undef)**
- **since: 2.0.00**

47.3.12 user

Get/set user name (as in http://user:password@host:port/)

```
$oldval = $parsed->user($newval);
```

- **obj: $parsed (APR::URI object)**
- **opt arg1: $newval (string or undef)**
- **ret: $oldval (string or undef)**
- **since: 2.0.00**

47.3.13 unparse

Unparse the URI components back into a URI string

```
$new_uri = $parsed->unparse();
$new_uri = $parsed->unparse($flags);
```

- **obj: $parsed (APR::URI object)**
- **opt arg1: $flags (the APR::Const :uri constants)**

  By default the constant APR::Const::URI_UNP_OMITPASSWORD is passed.

  If you need to pass more than one flag use unary |, e.g.:

  ```
  $flags = APR::Const::URI_UNP_OMITUSER|APR::Const::URI_UNP_OMITPASSWORD;
  ```

  The valid flags constants are listed next

- **ret: $new_uri (string)**
- **since: 2.0.00**

Valid flags constants:

To import all URI constants you could do:

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

but there is a significant amount of them, most irrelevant to this method. Therefore you probably don't want to do that. Instead specify explicitly the ones that you need. All the relevant to this method constants start with `APR::URI_UNP_`.

And the available constants are:

- **`APR::Const::URI_UNP_OMITSITEPART`**

  Don't show `scheme`, `user`, `password`, `hostname` and `port` components (i.e. if you want only the relative URI)

- **`APR::Const::URI_UNP_OMITUSER`**

  Hide the `user` component

- **`APR::Const::URI_UNP_OMITPASSWORD`**

  Hide the `password` component (the default)

- **`APR::Const::URI_UNP_REVEALPASSWORD`**

  Reveal the `password` component

- **`APR::Const::URI_UNP_OMITPATHINFO`**

  Don't show `path`, `query` and `fragment` components

- **`APR::Const::URI_UNP_OMITQUERY`**

  Don't show `query` and `fragment` components

Notice that some flags overlap.

If the optional `$flags` argument is passed and contains no `APR::Const::URI_UNP_OMITPASSWORD` and no `APR::Const::URI_UNP_REVEALPASSWORD` -- the `password` part will be rendered as a literal `"XXXXXXXX"` string.

If the `port` number matches the `port_of_scheme()`, the unparsed URI won't include it and there is no flag to force that `port` to appear. If the `port` number is non-standard it will show up in the unparsed string.

Examples:

Starting with the parsed URL:

```
use APR::URI ();
my $url = 'http://user:pass@example.com:80/foo?bar#item5';
my $parsed = APR::URI->parse($r->pool, $url);
```

deparse it back including and excluding parts, using different values for the optional `flags` argument:

- Show all but the `password` fields:

  ```
  print $parsed->unparse;
  ```

  Prints:

  ```
  http://user@example.com/foo?bar#item5
  ```

  Notice that the `port` field is gone too, since it was a default `port` for `scheme` `http://`.

- Include the `password` field (by default it's not revealed)

  ```
  use APR::Const -compile => qw(URI_UNP_REVEALPASSWORD);
  print $parsed->unparse(APR::Const::URI_UNP_REVEALPASSWORD);
  ```

  Prints:

  ```
  http://user:pass@example.com/foo?bar#item5
  ```

- Show all fields but the last three, `path`, `query` and `fragment`:

  ```
  use APR::Const -compile => qw(URI_UNP_REVEALPASSWORD
                                APR::Const::URI_UNP_OMITPATHINFO);
  print $parsed->unparse(
      APR::Const::URI_UNP_REVEALPASSWORD|URI_UNP_OMITPATHINFO);
  ```

  Prints:

  ```
  http://user:pass@example.com
  ```

47.4 See Also

`Apache2::URI`, mod_perl 2.0 documentation.

47.5 Copyright

47.6 Authors

The mod_perl development team and numerous contributors.

48 APR::Util - Perl API for Various APR Utilities

48.1 Synopsis

```
use APR::Util ();

$ok = password_validate($passwd, $hash);
```

48.2 Description

Various APR utilities that don't fit into any other group.

48.3 API

`APR::Util` provides the following functions and/or methods:

48.3.1 `password_validate`

Validate an encrypted password hash against a plain text password (with lots of restrictions and peculiarities).

```
$ok = password_validate($passwd, $hash);
```

- **arg1: `$passwd` (string)**

  Plain text password string

- **arg2: `$hash` (string)**

  Encrypted or encoded hash. See below for supported hash formats.

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

  The password either matches or not.

- **since: 2.0.00**

The function handles the output of the following functions (it knows to tell md5 and sha1 from the others, since they are have a special pattern recognized by apr):

- **md5**

  generated by `apr_md5_encode()` (for which at the moment we have no perl glue, ask if you need it).

- **sha1**

  generated by `apr_sha1_base64()` (for which at the moment we have no perl glue, ask if you need it).

and it's available only since Apache 2.0.50

- **crypt**

  On all but the following platforms: MSWin32, beos and NetWare. Therefore you probably don't want to use that feature, unless you know that your code will never end up running on those listed platforms.

  Moreover on these three platforms if that function sees that the hash is not of md5 and sha1 formats, it'll do a clear to clear text matching, always returning success, no matter what the hashed value is.

Warning: double check that you understand what this function does and does not before using it.

48.4 Unsupported API

`APR::Socket` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

48.4.1 `filepath_name_get`

META: Autogenerated - needs to be reviewed/completed

[We have File::Spec and File::Basename for this purpose, I can't see why this api is needed]

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

  For example:

  ```
  "/foo/bar/gum"    => "gum"
  "/foo/bar/gum/"   => ""
  "gum"             => "gum"
  "bs\\path\\stuff" => "stuff"
  ```

- **since: subject to change**

48.4.2 `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

- **arg3: `$bufsize` (number)**

  The length of the password buffer.

- **ret: `$ret` (integer)**
- **since: subject to change**

48.5 See Also

mod_perl 2.0 documentation.

48.6 Copyright

48.7 Authors

The mod_perl development team and numerous contributors.

49 APR::UUID - Perl API for manipulating APR UUIDs

49.1 Synopsis

```
use APR::UUID ();

# get a random UUID and format it as a string
my $uuid = APR::UUID->new->format;
# $uuid = e.g. 'd48889bb-d11d-b211-8567-ec81968c93c6';

# same as the object returned by APR::UUID->new
my $uuid_parsed = APR::UUID->parse($uuid);
```

49.2 Description

APR::UUID is used to get and manipulate random UUIDs.

It allows you to create random UUIDs, which when formatted returns a string like:

```
'd48889bb-d11d-b211-8567-ec81968c93c6';
```

which can be parsed back into the APR::UUID object with parse().

49.3 API

APR::UUID provides the following functions and/or methods:

49.3.1 format

Convert an APR::UUID object object into a string presentation:

```
my $uuid_str = $uuid->format;
```

- **obj: $uuid (APR::UUID object)**
- **ret: $uuid_str**

  returns a string representation of the object (.e.g 'd48889bb-d11d-b211-8567-ec81968c93c6').

- **since: 2.0.00**

49.3.2 new

Create a APR::UUID object using the random engine:

```
my $uuid = APR::UUID->new;
```

- **class: APR::UUID (APR::UUID class)**
- **ret: $uuid (APR::UUID object)**
- **since: 2.0.00**

49.3.3 DESTROY

```
$uuid->DESTROY;
```

- **obj: `APR::UUID` (`APR::UUID object`)**
- **ret: no return value**
- **since: 2.0.00**

Do not call this method, it's designed to be only called by Perl when the variable goes out of scope. If you call it yourself you will get a segfault when perl will call DESTROY on its own.

49.3.4 parse

Convert a UUID string into an `APR::UUID object` object:

```
$uuid = APR::UUID->parse($uuid_str)
```

- **arg1: `$uuid_str` (string)**

  UUID string (.e.g `'d48889bb-d11d-b211-8567-ec81968c93c6'`)

- **ret: `$uuid` (`APR::UUID object`)**

  The new object.

- **since: 2.0.00**

49.4 See Also

mod_perl 2.0 documentation.

49.5 Copyright

49.6 Authors

The mod_perl development team and numerous contributors.

50 ModPerl::Const -- ModPerl Constants

50.1 Synopsis

```
# make the constants available but don't import them
use ModPerl::Const -compile => qw(constant names ...);

# w/o the => syntax sugar
use ModPerl::Const ("-compile", qw(constant names ...));

# compile and import the constants
use ModPerl::Const qw(constant names ...);
```

50.2 Description

This package contains constants specific to mod_perl features.

Refer to `the Apache2::Const description section` for more information.

50.3 Constants

50.3.1 Other Constants

50.3.1.1 `ModPerl::EXIT`

- **since: 2.0.00**

See `ModPerl::Util::exit`.

50.4 See Also

mod_perl 2.0 documentation.

50.5 Copyright

50.6 Authors

The mod_perl development team and numerous contributors.

51 ModPerl::Global -- Perl API for manipulating special Perl lists

51.1 Synopsis

```
use ModPerl::Global ();
my $package = 'MyApache2::Package';

# make the END blocks of this package special
ModPerl::Global::special_list_register(END => $package);

# Execute all encoutered END blocks from this package now
ModPerl::Global::special_list_call(    END => $package);

# delete the list of END blocks
ModPerl::Global::special_list_clear(   END => $package);
```

51.2 Description

`ModPerl::Global` provides an API to manipulate special perl lists. At the moment only the `END` blocks list is supported.

This API allows you to change the normal Perl behavior, and execute special lists when you need to.

For example `ModPerl::RegistryCooker` uses it to run `END` blocks in the scripts at the end of each request.

Before loading a module containing package `$package`, you need to register it, so the special blocks will be intercepted by mod_perl and not given to Perl. `special_list_register` does that. Later on when you want to execute the special blocks, `special_list_call` should be called. Unless you want to call the list more than once, clear the list with `special_list_clear`.

51.3 API

`ModPerl::Global` provides the following methods:

51.3.1 `special_list_call`

Call the special list

```
$ok = special_list_call($key => $package);
```

- **arg1: `$key` (string)**

  The name of the special list. At the moment only `'END'` is supported.

- **arg2: `$package` (string)**

  what package to special list is for

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

  true value if `$key` is known, false otherwise.

- **since: 2.0.00**

51.3.2 `special_list_clear`

Clear the special list

```
$ok = special_list_clear($key => $package);
```

- **arg1: `$key` (string)**

  The name of the special list. At the moment only `'END'` is supported.

- **arg2: `$package` (string)**

  what package to special list is for

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

  true value if `$key` is known, false otherwise.

- **since: 2.0.00**

51.3.3 `special_list_register`

Register the special list

```
$ok = special_list_call($key => $package);
```

- **arg1: `$key` (string)**

  The name of the special list. At the moment only `'END'` is supported.

- **arg2: `$package` (string)**

  what package to special list is for

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

  true value if `$key` is known, false otherwise.

- **since: 2.0.00**

Notice that you need to register the package before it is loaded. If you register it after, Perl has already compiled the `END` blocks and there are no longer under your control.

51.4 See Also

mod_perl 2.0 documentation.

51.5 Copyright

51.6 Authors

The mod_perl development team and numerous contributors.

52 ModPerl::MethodLookup -- Lookup mod_perl modules, objects and methods

52.1 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 'Apache2::Filter'
# (here $filter is an Apache2::Filter object)
my ($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print', $filter);
# or
my ($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print', 'Apache2::Filter');

# what XS methods defined by module 'Apache2::Filter'
my ($hint, @methods) =
    ModPerl::MethodLookup::lookup_module('Apache2::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('Apache2::RequestRec');

# preload all mp2 modules in startup.pl
ModPerl::MethodLookup::preload_all_modules();

# command line shortcuts
% perl -MModPerl::MethodLookup -e print_module \
  Apache2::RequestRec Apache2::Filter
% perl -MModPerl::MethodLookup -e print_object Apache2
% perl -MModPerl::MethodLookup -e print_method \
  get_server_built request
% perl -MModPerl::MethodLookup -e print_method read
% perl -MModPerl::MethodLookup -e print_method read APR::Bucket
```

52.2 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, to find out out what methods defined by some module, what methods can be called on a given object, etc.

52.3 API

52.3.1 lookup_method()

Find modules (packages) containing a certain method

```
($hint, @modules) = lookup_method($method_name);
($hint, @modules) = lookup_method($method_name, $object);
($hint, @modules) = lookup_method($method_name, $class));
```

- **arg1: `$method_name` (string)**

  the method name to look up

- **opt arg2: `$object` or `$class`**

  a blessed object or the name of the class it's blessed into. If there is more than one match, this extra information is used to return only modules containing methods operating on the objects of the same kind.

  If a sub-classed object is passed it'll be handled correctly, by checking its super-class(es). This usage is useful when the `AUTOLOAD` is used to find a not yet loaded module which include the called method.

- **ret1: `$hint`**

  a string containing a human readable lookup result, suggesting which modules should be loaded, ready for copy-n-paste or explaining the failure if the lookup didn't succeed.

- **ret2: `@modules`**

  an array of modules which have matched the query, i.e. the names of the modules which contain the requested method.

- **since: 2.0.00**

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 `Apache2::Filter`:

```
my $filter = bless {}, 'Apache2::Filter';
my ($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print', $filter);
```

or:

```
my ($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print', 'Apache2::Filter');
```

52.3.2 lookup_module()

Find methods contained in a certain module (package)

```
($hint, @methods) = lookup_module($module_name);
```

- **arg1: `$module_name` (string)**

  the module name

- **ret1: `$hint`**

  a string containing a human readable lookup result, suggesting, which methods the module `$module_name` implements, or explaining the failure if the lookup failed.

- **ret2: `@methods`**

  an array of methods which have matched the query, i.e. the names of the methods defined in the requested module.

- **since: 2.0.00**

Example:

What XS methods defined by module `Apache2::Filter`:

```
my ($hint, @methods) =
    ModPerl::MethodLookup::lookup_module('Apache2::Filter');
```

52.3.3 lookup_object()

```
($hint, @methods) = lookup_object($object);
($hint, @methods) = lookup_object($class);
```

- **arg1: `$object` or `$class`**

  an object or a name of a class an object is blessed into

  If a sub-classed object is passed it'll be handled correctly, by including methods provided by its super-class(es).

- **ret1: `$hint`**

  a string containing a human readable lookup result, 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 failed.

- **ret2: `@methods`**

  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).

- **since: 2.0.00**

META: As of this writing this function 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 (`Apache2::Log`).

Examples:

What XS methods can be invoked on the object `$r`:

```
my ($hint, @methods) =
    ModPerl::MethodLookup::lookup_object($r);
```

or `$r`'s class -- `Apache2::RequestRec`:

```
my ($hint, @methods) =
    ModPerl::MethodLookup::lookup_object('Apache2::RequestRec');
```

52.3.4 `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.

- **since: 2.0.00**

52.3.5 `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 -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 -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 -MModPerl::MethodLookup -e print_method \
  get_server_built request
```

The function `print_method()` is exported by default.

- **since: 2.0.00**

52.3.6 `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 `Apache2::RequestRec`, followed by methods defined in the module `Apache2::Filter` you can run:

```
% perl -MModPerl::MethodLookup -e print_module \
  Apache2::RequestRec Apache2::Filter
```

The function `print_module()` is exported by default.

- **since: 2.0.00**

52.3.7 `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 `Apache2::RequestRec` run:

```
% perl -MModPerl::MethodLookup -e print_object \
  Apache2::RequestRec
```

Similar to `print_object()`, more than one class can be passed to this function.

The function `print_object()` is exported by default.

- **since: 2.0.00**

52.4 Applications

52.4.1 `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 facil-

ity 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 $hintand 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 {

        *UNIVERSAL::AUTOLOAD = sub {
            my $method = $AUTOLOAD;
            return if $method =~ /DESTROY/; # exclude DESTROY resolving

            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 `Apache2::porting`.

META: there is a better version of AUTOLOAD discussed on the dev list. Replace the current one with it. (search the archive for EazyLife)

52.4.2 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
"Apache2::RequestRec" at ...
```

you can ask `ModPerl::MethodLookup` for help:

```
% perl -MModPerl::MethodLookup -e print_method construct_url
To use method ’construct_url’ add:
        use Apache2::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 -MModPerl::MethodLookup -e print_method"
```

For Bash-style shell users:

```
 % alias lookup="perl -MModPerl::MethodLookup -e print_method"
```

Now the lookup is even easier:

```
% lookup construct_url
to use method ’construct_url’ add:
        use Apache2::URI;
```

Similar aliases can be provided for `print_object()` and `print_module()`.

52.5 Todo

These methods aren’t yet picked by this module (the extract from the map file):

```
modperl_filter_attributes     | MODIFY_CODE_ATTRIBUTES
modperl_spawn_proc_prog       | spawn_proc_prog
apr_ipsubnet_create           | new
```

Please report to the mod_perl development mailing list if you find any other missing methods. But remember that as of this moment the module reports only XS functions. In the future we may add support for pure perl functions/methods as well.

52.6 See Also

- the mod_perl 1.0 backward compatibility document
- porting Perl modules
- porting XS modules
- `Apache2::porting`

52.7 Copyright

52.8 Authors

The mod_perl development team and numerous contributors.

53 ModPerl::MM -- A "subclass" of ExtUtils::MakeMaker for mod_perl 2.0

53.1 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'),
    ...
);
```

53.2 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.

53.3 `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 overriden, 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 overriden by `ModPerl::MM`, add to your *Makefile.PL*:

```
sub MY::constants { shift->MM::constants(@_); }";
```

`ModPerl::MM` overrides the following methods:

53.3.1 `ModPerl::MM::MY::post_initialize`

This method is deprecated.

53.4 `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 overriden 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:

```
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:

53.4.1 CCFLAGS

53.4.2 LIBS

53.4.3 INC

53.4.4 OPTIMIZE

53.4.5 LDDLFLAGS

53.4.6 TYPEMAPS

53.4.7 dynamic_lib

53.4.7.1 OTHERLDFLAGS

```
dynamic_lib => { OTHERLDFLAGS => ... }
```

53.4.8 macro

53.4.8.1 MOD_INSTALL

```
macro => { MOD_INSTALL => ... }
```

makes sure that Apache-Test/ is added to @INC.

53.5 Public API

The following functions are a part of the public API. They are described elsewhere in this document.

53.5.1 WriteMakefile()

```
ModPerl::MM::WriteMakefile(...);
```

53.5.2 get_def_opt()

```
my $def_val = ModPerl::MM::get_def_opt($key);
```

54 ModPerl::PerlRun - Run unaltered CGI scripts under mod_perl

54.1 Synopsis

```
# httpd.conf
PerlModule ModPerl::PerlRun
Alias /perl-run/ /home/httpd/perl/
<Location /perl-run>
    SetHandler perl-script
    PerlResponseHandler ModPerl::PerlRun
    PerlOptions +ParseHeaders
    Options +ExecCGI
</Location>
```

54.2 Description

META: document that for now we don't chdir() into the script's dir, because it affects the whole process under threads. `ModPerl::PerlRunPrefork` should be used by those who run only under prefork MPM.

54.3 Special Blocks

54.3.1 `BEGIN` Blocks

When running under the `ModPerl::PerlRun` handler `BEGIN` blocks behave as follows:

- `BEGIN` blocks defined in scripts running under the `ModPerl::PerlRun` handler are executed on each and every request.
- `BEGIN` blocks defined in modules loaded from scripts running under `ModPerl::PerlRun` (and which weren't already loaded prior to the request) are executed on each and every request only if those modules declare no package. If a package is declared `BEGIN` blocks will be run only the first time each module is loaded, since those modules don't get reloaded on subsequent requests.

See also `BEGIN` blocks in mod_perl handlers.

54.3.2 `CHECK` and `INIT` Blocks

Same as normal mod_perl handlers.

54.3.3 `END` Blocks

Same as `ModPerl::Registry`.

54.4 Authors

Doug MacEachern

Stas Bekman

54.5 See Also

`ModPerl::RegistryCooker` and `ModPerl::Registry`.

55 ModPerl::PerlRunPrefork - Run unaltered CGI scripts under mod_perl

55.1 Synopsis

```
# httpd.conf
PerlModule ModPerl::PerlRunPrefork
Alias /perl-run/ /home/httpd/perl/
<Location /perl-run>
    SetHandler perl-script
    PerlResponseHandler ModPerl::PerlRunPrefork
    PerlOptions +ParseHeaders
    Options +ExecCGI
</Location>
```

55.2 Description

55.3 Copyright

55.4 Authors

The mod_perl development team and numerous contributors.

55.5 See Also

`ModPerl::RegistryCooker` and `ModPerl::Registry`.

56 ModPerl::Registry - Run unaltered CGI scripts persistently under mod_perl

56.1 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>
```

56.2 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 = Apache2::RequestUtil->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. `ModPerl::RegistryPrefork` should be used by those who run only under prefork MPM.

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.

56.3 DirectoryIndex

If you are trying setup a DirectoryIndex under a Location covered by ModPerl::Registry* you might run into some trouble.

META: if this gets added to core, replace with real documenation. See http://marc.theaims-group.com/?l=apache-modperl&m=112805393100758&w=2

56.4 Special Blocks

56.4.1 `BEGIN` Blocks

`BEGIN` blocks defined in scripts running under the `ModPerl::Registry` handler behave similarly to the normal mod_perl handlers plus:

- Only once, if pulled in by the parent process via `Apache2::RegistryLoader`.
- An additional time, once per child process or Perl interpreter, each time the script file changes on disk.

`BEGIN` blocks defined in modules loaded from `ModPerl::Registry` scripts behave identically to the normal mod_perl handlers, regardless of whether they define a package or not.

56.4.2 `CHECK` and `INIT` Blocks

Same as normal mod_perl handlers.

56.4.3 `END` Blocks

`END` blocks encountered during compilation of a script, are called after the script has completed its run, including subsequent invocations when the script is cached in memory. This is assuming that the script itself doesn't define a package on its own. If the script defines its own package, the `END` blocks in the scope of that package will be executed at the end of the interpretor's life.

`END` blocks residing in modules loaded by registry script will be executed only once, when the interpreter exits.

56.5 Security

`ModPerl::Registry::handler` performs the same sanity checks as mod_cgi does, before running the script.

56.6 Environment

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

56.7 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.

56.8 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
```

56.9 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! Some other important caveats to keep in mind are discussed on the Perl Reference page.

56.10 Authors

Andreas J. Koenig, Doug MacEachern and Stas Bekman.

56.11 See Also

`ModPerl::RegistryCooker`, `ModPerl::RegistryBB` and `ModPerl::PerlRun`.

57 ModPerl::RegistryBB - Run unaltered CGI scripts persistently under mod_perl

57.1 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>
```

57.2 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.

57.3 Authors

Doug MacEachern

Stas Bekman

57.4 See Also

`ModPerl::RegistryCooker`, `ModPerl::Registry` and `ModPerl::PerlRun`.

58 ModPerl::RegistryCooker - Cook mod_perl 2.0 Registry Modules

58.1 Synopsis

```
# shouldn't be used as-is but sub-classed first
# see ModPerl::Registry for an example
```

58.2 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 overriden 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` -- `Apache2::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()

- **shebang_to_perl()**

  default: shebang_to_perl()

- **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()

58.2.1 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.

58.3 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 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 then 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.

58.4 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]

58.5 Authors

Doug MacEachern

Stas Bekman

58.6 See Also

`ModPerl::Registry`, `ModPerl::RegistryBB` and `ModPerl::PerlRun`.

59 ModPerl::RegistryLoader - Compile ModPerl::RegistryCooker scripts at server startup

59.1 Synopsis

```
# in startup.pl
use ModPerl::RegistryLoader ();
use File::Spec ();

# 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 File::Spec->catfile(Apache2::ServerUtil::server_root, $uri);
}
my $rl = ModPerl::RegistryLoader->new(
    package => 'ModPerl::Registry',
    trans   => \&trans,
);
$rl->handler($uri);

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

59.2 Description

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.

This module is of course useless for those running the `ModPerl::PerlRun` handler, because the scripts get recompiled on each request under this handler.

59.3 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 File::Spec ();
    use DirHandle ();
    use strict;

    my $dir = File::Spec->catdir(Apache2::ServerUtil::server_root,
                                 "cgi-bin");

    sub trans {
        my $uri = shift;
        $uri =~ s|^/registry/|cgi-bin/|;
        return File::Spec->catfile(Apache2::ServerUtil::server_root,
                                   $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.

59.4 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.

59.5 Authors

The original `Apache2::RegistryLoader` implemented by Doug MacEachern.

Stas Bekman did the porting to the new registry framework based on `ModPerl::RegistryLoader`.

59.6 SEE ALSO

`ModPerl::RegistryCooker`, `ModPerl::Registry`, `ModPerl::RegistryBB`, `ModPerl::PerlRun`, Apache(3), mod_perl(3)

60 ModPerl::RegistryPrefork - Run unaltered CGI scripts under mod_perl

60.1 Synopsis

```
# httpd.conf
PerlModule ModPerl::RegistryPrefork
Alias /perl-run/ /home/httpd/perl/
<Location /perl-run>
    SetHandler perl-script
    PerlResponseHandler ModPerl::RegistryPrefork
    PerlOptions +ParseHeaders
    Options +ExecCGI
</Location>
```

60.2 Description

60.3 Copyright

60.4 Authors

The mod_perl development team and numerous contributors.

60.5 See Also

`ModPerl::RegistryCooker` and `ModPerl::Registry`.

61 ModPerl::Util - Helper mod_perl Functions

61.1 Synopsis

```
use ModPerl::Util;

# e.g. PerlResponseHandler
$callback = ModPerl::Util::current_callback;

# exit w/o killing the interpreter
ModPerl::Util::exit();

# untaint a string (do not use it! see the doc)
ModPerl::Util::untaint($string);

# removes a stash (.so, %INC{$stash}, etc.) as best as it can
ModPerl::Util::unload_package($stash);

# current perl's address (0x92ac760 or 0x0 under non-threaded perl)
ModPerl::Util::current_perl_id();
```

61.2 Description

`ModPerl::Util` provides mod_perl utilities API.

61.3 API

`ModPerl::Util` provides the following functions and/or methods:

61.3.1 `current_callback`

Returns the currently running callback name, e.g. `'PerlResponseHandler'`.

```
$callback = ModPerl::Util::current_callback();
```

- **ret: `$callback` (string)**
- **since: 2.0.00**

61.3.2 `current_perl_id`

Return the memory address of the perl interpreter

```
$perl_id = ModPerl::Util::current_perl_id();
```

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

  Under threaded perl returns something like: `0x92ac760`

Under non-thread perl returns `0x0`

- **since: 2.0.00**

Mainly useful for debugging applications running under threaded-perl.

61.3.3 `exit`

Terminate the request, but not the current process (or not the current Perl interpreter with threaded mpms).

```
ModPerl::Util::exit($status);
```

- **opt arg1: `$status` (integer)**

  The exit status, which as of this writing is ignored. (it's accepted to be compatible with the core `exit` function.)

- **ret: no return value**
- **since: 2.0.00**

Normally you will use the plain `exit()` in your code. You don't need to use `ModPerl::Util::exit` explicitly, since mod_perl overrides `exit()` by setting `CORE::GLOBAL::exit` to `ModPerl::Util::exit`. Only if you redefine `CORE::GLOBAL::exit` once mod_perl is running, you may want to use this function.

The original `exit()` is still available via `CORE::exit()`.

`ModPerl::Util::exit` is implemented as a special `die()` call, therefore if you call it inside `eval BLOCK` or `eval "STRING"`, while an exception is being thrown, it is caught by `eval`. For example:

```
exit;
print "Still running";
```

will not print anything. But:

```
eval {
   exit;
}
print "Still running";
```

will print *Still running*. So you either need to check whether the exception is specific to `exit` and call `exit()` again:

```
use ModPerl::Const -compile => 'EXIT';
eval {
   exit;
}
exit if $@ && ref $@ eq 'APR::Error' && $@ == ModPerl::EXIT;
print "Still running";
```

or use `CORE::exit()`:

```
eval {
   CORE::exit;
}
print "Still running";
```

and nothing will be printed. The problem with the latter is the current process (or a Perl Interpreter) will be killed; something that you really want to avoid under mod_perl.

61.3.4 unload_package

Unloads a stash from the current Perl interpreter in the safest way possible.

```
ModPerl::Util::unload_package($stash);
```

- **arg1: `$stash` (string)**

  The Perl stash to unload. e.g. `MyApache2::MyData`.

- **ret: no return value**
- **since: 2.0.00**

Unloading a Perl stash (package) is a complicated business. This function tries very hard to do the right thing. After calling this function, it should be safe to `use()` a new version of the module that loads the wiped package.

References to stash elements (functions, variables, etc.) taken from outside the unloaded package will still be valid.

This function may wipe off things loaded by other modules, if the latter have inserted things into the `$stash` it was told to unload.

If a stash had a corresponding XS shared object (.so) loaded it will be unloaded as well.

If the stash had a corresponding entry in `%INC`, it will be removed from there.

`unload_package()` takes care to leave sub-stashes intact while deleting the requested stash. So for example if `CGI` and `CGI::Carp` are loaded, calling `unload_package('CGI')` won't affect `CGI::Carp`.

61.3.5 untaint

Untaint the variable, by turning its tainted SV flag off (used internally).

```
ModPerl::Util::untaint($tainted_var);
```

- **arg1: `$tainted_var` (scalar)**
- **ret: no return value**

  `$tainted_var` is untainted.

- **since: 2.0.00**

Do not use this function unless you know what you are doing. To learn how to properly untaint variables refer to the *perlsec* manpage.

61.4 See Also

mod_perl 2.0 documentation.

61.5 Copyright

61.6 Authors

The mod_perl development team and numerous contributors.

62 Apache2::compat -- 1.0 backward compatibility functions deprecated in 2.0

62.1 Synopsis

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

# override and restore compat functions colliding with mp2 API
Apache2::compat::override_mp2_api('Apache2::Connection::local_addr');
my ($local_port, $local_addr) = sockaddr_in($c->local_addr);
Apache2::compat::restore_mp2_api('Apache2::Connection::local_addr');
```

62.2 Description

`Apache2::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.

62.3 Compatibility Functions Colliding with mod_perl 2.0 API

Most of the functions provided by Apache2::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 Apache2::Connection;
require APR::SockAddr;
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::SockAddr` object, which is a totally different beast. If Apache2::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:

```
Apache2::compat::override_mp2_api('Apache2::Connection::local_addr');
my ($local_port, $local_addr) = Socket::sockaddr_in($c->local_addr);
Apache2::compat::restore_mp2_api('Apache2::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.

62.3.1 Available Overridable Functions

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

- **Apache2::RequestRec::notes**
- **Apache2::RequestRec::filename**
- **Apache2::RequestRec::finfo**
- **Apache2::Connection::local_addr**
- **Apache2::Connection::remote_addr**
- **Apache2::Util::ht_time**
- **Apache2::Module::top_module**
- **Apache2::Module::get_config**
- **APR::URI::unparse**

62.4 Use in CPAN Modules

The short answer: **Do not use** `Apache2::compat` in CPAN modules.

The long answer:

`Apache2::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 `Apache2::compat`

user's code is forced to use the potentially slower method. Which is quite bad.

Some users may choose to keep using `Apache2::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 `Apache2::compat`, can prevent this from happening by adding

```
$INC{'Apache2/compat.pm'} = __FILE__;
```

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

62.5 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.

62.6 See Also

mod_perl 2.0 documentation.

62.7 Copyright

62.8 Authors

The mod_perl development team and numerous contributors.

63 Apache2::porting -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting

63.1 Synopsis

```
# either add at the very beginning of startup.pl
use Apache2::porting;

# or httpd.conf
PerlModule Apache2::porting

# now issue requests and look at the error_log file for hints
```

63.2 Description

`Apache2::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 to trap methods calls 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.

`Apache2::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 `Apache2::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.

63.3 Culprits

`Apache2::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`:

```
Apache2::ServerUtil->server->push_handlers(PerlPostConfigHandler => \&porting_autoload);
```

META: Though there is a better solution at work, which assigns AUTOLOAD for each class separately, instead of using UNIVERSAL. See the discussion on the dev list (hint: search the archive for EazyLife)

63.4 See Also

mod_perl 2.0 documentation.

63.5 Copyright

63.6 Authors

The mod_perl development team and numerous contributors.

64 Apache2::Reload - Reload Perl Modules when Changed on Disk

64.1 Synopsis

```
# Monitor and reload all modules in %INC:
# httpd.conf:
PerlModule Apache2::Reload
PerlInitHandler Apache2::Reload

# when working with protocols and connection filters
# PerlPreConnectionHandler Apache2::Reload

# Reload groups of modules:
# httpd.conf:
PerlModule Apache2::Reload
PerlInitHandler Apache2::Reload
PerlSetVar ReloadAll Off
PerlSetVar ReloadModules "ModPerl::* Apache2::*"
#PerlSetVar ReloadDebug On

# Reload a single module from within itself:
package My::Apache2::Module;
use Apache2::Reload;
sub handler { ... }
1;
```

64.2 Description

`Apache2::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 `Apache2::Reload`. It can also do the check for modified modules, when a special touch-file has been modified.

Note that `Apache2::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 `ModPerl::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 `Apache2::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
```

64.2.1 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 Apache2::Reload
PerlInitHandler Apache2::Reload
```

When working with connection filters and protocol modules `Apache2::Reload` should be invoked in the pre_connection stage:

```
PerlPreConnectionHandler Apache2::Reload
```

See also the discussion on `PerlPreConnectionHandler`.

64.2.2 Register Modules Implicitly

To only reload modules that have registered with `Apache2::Reload`, add the following to the *httpd.conf*:

```
PerlModule Apache2::Reload
PerlInitHandler Apache2::Reload
PerlSetVar ReloadAll Off
# ReloadAll defaults to On
```

Then any modules with the line:

```
use Apache2::Reload;
```

Will be reloaded when they change.

64.2.3 Register Modules Explicitly

You can also register modules explicitly in your *httpd.conf* file that you want to be reloaded on change:

```
PerlModule Apache2::Reload
PerlInitHandler Apache2::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::* Apache2::*"
```

will monitor all modules under the namespaces `ModPerl::` and `Apache2::`.

64.2.4 Monitor Only Certain Sub Directories

To reload modules only in certain directories (and their subdirectories) add the following to the *httpd.conf*:

```
PerlModule Apache2::Reload
PerlInitHandler Apache2::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 Apache2::Reload
PerlInitHandler Apache2::Reload
PerlSetVar ReloadDirectories "/tmp/project1 /tmp/project2"
PerlSetVar ReloadAll Off
PerlSetVar ReloadModules "MyApache2::*"
```

In this configuration example only modules from the namespace `MyApache2::` found in the directories */tmp/project1/* and */tmp/project2/* (and their subdirectories) will be reloaded.

64.2.5 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.

64.2.6 Unregistering a module

In some cases, it might be necessary to explicitely stop reloading a module.

```
Apache2::Reload->unregister_module('Some::Module');
```

But be carefull, since unregistering a module in this way will only do so for the current interpreter. This feature should be used with care.

64.3 Performance Issues

This module is perfectly suited for a development environment. Though it's possible that you would like to use it in a production environment, since with `Apache2::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, `Apache2::Reload` has to stat(2) the touch file on each request, which adds a slight but most likely insignificant overhead to response times. Otherwise `Apache2::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.

64.4 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
```

64.5 Caveats

64.5.1 Problems With Reloading Modules Which Do Not Declare Their Package Name

If you modify modules, which don't declare their `package`, and rely on `Apache2::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 `Apache2::Reload` `require()`s such a module all the global symbols end up in the `Apache2::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.

64.5.2 Failing to Find a File to Reload

`Apache2::Reload` uses `%INC` to find the files on the filesystem. If an entry for a certain filepath in `%INC` is relative, `Apache2::Reload` will use `@INC` to try to resolve that relative path. Now remember that mod_perl freezes the value of `@INC` at the server startup, and you can modify it only for the duration of one request when you need to load some module which is not in on of the `@INC` directories. So a module gets loaded, and registered in `%INC` with a relative path. Now when `Apache2::Reload` tries to find that module to check whether it has been modified, it can't find since its directory is not in `@INC`. So `Apache2::Reload` will silently skip that module.

You can enable the `Debug|/Debug` mode to see what `Apache2::Reload` does behind the scenes.

64.5.3 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`.

64.5.3.1 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. `Apache2::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
```

64.5.3.2 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 secondnd 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.

64.5.3.3 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.

64.6 Threaded MPM and Multiple Perl Interpreters

If you use `Apache2::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 `Apache2::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.

64.7 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.

64.8 Copyright

64.9 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.

65 Apache2::Resource - Limit resources used by httpd children

65.1 Synopsis

```
PerlModule Apache2::Resource
# set child memory limit in megabytes
# default is 64 Meg
PerlSetEnv PERL_RLIMIT_DATA 32:48

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

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

PerlChildInitHandler Apache2::Resource
```

65.2 Description

`Apache2::Resource` uses the `BSD::Resource` module, which uses the C function `setrlimit` to set limits on system resources such as memory and cpu usage.

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

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

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

65.3 Defaults

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

```
PerlSetEnv PERL_RLIMIT_DEFAULTS On
PerlModule Apache2::Resource
```

65.4 See Also

BSD::Resource(3), setrlimit(2)

65.5 Copyright

65.6 Author

Doug MacEachern

66 Apache2::Status - Embedded interpreter status information

66.1 Synopsis

```
<Location /perl-status>

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

    SetHandler modperl
    PerlOptions +GlobalRequest
    PerlResponseHandler Apache2::Status
</Location>

or

<Location /perl-status>

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

    SetHandler perl-script
    PerlResponseHandler Apache2::Status
</Location>
```

66.2 Description

The `Apache2::Status` module provides some information about the status of the Perl interpreter embedded in the server.

Configure like so:

```
<Location /perl-status>

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

    SetHandler modperl
    PerlOptions +GlobalRequest
    PerlResponseHandler Apache2::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 `Apache2::Status` as:

```
<Location /perl-status>

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

    SetHandler perl-script
    PerlResponseHandler Apache2::Status
</Location>
```

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

```
require Apache2::Module;
Apache2::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 Apache2::Module::loaded('Apache2::Status');
```

WARNING: `Apache2::Status` must be loaded before these modules via the `PerlModule` or `PerlRequire` directives (or from *startup.pl*).

A very common setup might be: Perl Module B::TerseSize

```
<Location /perl-status>
    SetHandler perl-script
    PerlResponseHandler Apache2::Status
    PerlSetVar StatusOptionsAll On
    PerlSetVar StatusDeparseOptions "-p -sC"
</Location>
```

due to the implementation of Apache2::Status::noh_fileline in B::TerseSize, you must load B::TerseSize first.

66.3 Options

66.3.1 `StatusOptionsAll`

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

```
PerlSetVar StatusOptionsAll On
```

66.3.2 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
```

66.3.3 StatusPeek

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

```
PerlSetVar StatusPeek On
```

66.3.4 StatusLexInfo

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

```
PerlSetVar StatusLexInfo On
```

66.3.5 StatusDeparse

With this option `On` 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.

66.3.6 StatusTerse

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

```
PerlSetVar StatusTerse On
```

66.3.7 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
```

66.3.8 StatusTerseSizeMainSummary

With this option `On` and the `B::TerseSize` module installed, a *"Memory Usage"* will be added to the `Apache2::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
```

66.3.9 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.graphviz.org/.

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

66.3.10 Dot

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

66.3.11 GraphDir

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

66.4 Prerequisites

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

Other optional functionality requirements: `B::Deparse` - 0.59, `B::Fathom` - 0.05, `B::Graph` - 0.03.

66.5 Copyright

66.6 See Also

perl(1), Apache(3), Devel::Symdump(3), Data::Dumper(3), B(3), `B::Graph`(3), mod_perl 2.0 documentation.

66.7 Authors

Doug MacEachern with contributions from Stas Bekman

67 Apache2::SizeLimit - Because size does matter.

67.1 Synopsis

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

```
# in your startup.pl, or a <Perl> section:
use Apache2::SizeLimit;
# sizes are in KB
$Apache2::SizeLimit::MAX_PROCESS_SIZE  = 12000; # 12MB
$Apache2::SizeLimit::MIN_SHARE_SIZE    = 6000;  # 6MB
$Apache2::SizeLimit::MAX_UNSHARED_SIZE = 5000;  # 5MB

# in your httpd.conf:
PerlCleanupHandler Apache2::SizeLimit
```

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

```
# in your script:
use Apache2::SizeLimit;
# sizes are in KB
Apache2::SizeLimit::setmax(12000);
Apache2::SizeLimit::setmin(6000);
Apache2::SizeLimit::setmax_unshared(5000);
```

This will work in places where you are using `SetHandler perl-script` or anywhere you enable `PerlOptions +GlobalRequest`. If you want to avoid turning on `GlobalRequest`, you can pass an `Apache2::RequestRec` object as the second argument in these subs:

```
my $r = shift; # if you don't have $r already
Apache2::SizeLimit::setmax(12000, $r);
Apache2::SizeLimit::setmin(6000, $r);
Apache2::SizeLimit::setmax_unshared(5000, $r);
```

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

```
$Apache2::SizeLimit::CHECK_EVERY_N_REQUESTS = 2;
```

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

67.2 Description

This module is highly platform dependent, please read the Caveats section. It also does not work under threaded MPMs.

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

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

This module will not really help you with the first problem. For that you should probably look into `Apache2::Resource` or some other means of setting a limit on the data size of your program. BSD-ish systems have `setrlimit()` which will croak your memory gobbling processes. However it is a little violent, terminating your process in mid-request.

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

By using this module, you should be able to discontinue using the Apache configuration directive `MaxRequestsPerChild`, although you can use both if you are feeling paranoid. Most users use the technique shown in this module and set their `MaxRequestsPerChild` value to `0`.

67.3 Shared Memory Options

In addition to simply checking the total size of a process, this module can factor in how much of the memory used by the process is actually being shared by copy-on-write. If you don’t understand how memory is shared in this way, take a look at the extensive documentation at http://perl.apache.org/docs/.

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

67.4 Caveats

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

67.4.1 Supported OSes

- **linux**

  For linux we read the process size out of */proc/self/statm*. This seems to be fast enough on modern systems. If you are worried about performance, try setting the `CHECK_EVERY_N_REQUESTS` option.

Since linux 2.6 */proc/self/statm* does not report the amount of memory shared by the copy-on-write mechanism as shared memory. Hence decisions made on the basis of `MAX_UNSHARED_SIZE` or `MIN_SHARE_SIZE` are inherently wrong.

To correct the situation there is a patch to the linux kernel that adds a */proc/self/smaps* entry for each process. At the time of this writing the patch is included in the mm-tree (linux-2.6.13-rc4-mm1) and is expected to make it into the vanilla kernel in the near future.

/proc/self/smaps reports various sizes for each memory segment of a process and allows to count the amount of shared memory correctly.

If `Apache2::SizeLimit` detects a kernel that supports */proc/self/smaps* and if the `Linux::Smaps` module is installed it will use them instead of */proc/self/statm*. You can prevent `Apache2::SizeLimit` from using */proc/self/smaps* and turn on the old behaviour by setting `$Apache2::SizeLimit::USE_SMAPS` to 0 before the first check.

`Apache2::SizeLimit` also resets `$Apache2::SizeLimit::USE_SMAPS` to 0 if it somehow decides not to use */proc/self/smaps*. Thus, you can check it to determine what is actually used.

NOTE: Reading */proc/self/smaps* is expensive compared to */proc/self/statm*. It must look at each page table entry of a process. Further, on multiprocessor systems the access is synchronized with spin-locks. Hence, you are encouraged to set the `CHECK_EVERY_N_REQUESTS` option.

The following example shows the effect of copy-on-write:

```
<Perl>
  require Apache2::SizeLimit;
  package X;
  use strict;
  use Apache2::RequestRec ();
  use Apache2::RequestIO ();
  use Apache2::Const -compile=>qw(OK);

  my $x= "a" x (1024*1024);

  sub handler {
    my $r = shift;
    my ($size, $shared) = $Apache2::SizeLimit::HOW_BIG_IS_IT->();
    $x =~ tr/a/b/;
    my ($size2, $shared2) = $Apache2::SizeLimit::HOW_BIG_IS_IT->();
    $r->content_type('text/plain');
    $r->print("1: size=$size shared=$shared\n");
    $r->print("2: size=$size2 shared=$shared2\n");
    return Apache2::Const::OK;
  }
</Perl>

<Location /X>
  SetHandler modperl
  PerlResponseHandler X
</Location>
```

The parent apache allocates a megabyte for the string in `$x`. The `tr`-command then overwrites all "a" with "b" if the handler is called with an argument. This write is done in place, thus, the process size doesn't change. Only `$x` is not shared anymore by means of copy-on-write between the parent and the child.

If */proc/self/smaps* is available curl shows:

```
r2@s93:~/work/mp2> curl http://localhost:8181/X
1: size=13452 shared=7456
2: size=13452 shared=6432
```

Shared memory has lost 1024 kB. The process' overall size remains unchanged.

Without */proc/self/smaps* it says:

```
r2@s93:~/work/mp2> curl http://localhost:8181/X
1: size=13052 shared=3628
2: size=13052 shared=3636
```

One can see the kernel lies about the shared memory. It simply doesn't count copy-on-write pages as shared.

- **Solaris 2.6 and above**

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

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

- **BSD**

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

- **AIX?**

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

- **Win32**

  Under mod_perl 1, SizeLimit provided basic functionality by using `Win32::API` to access process memory information. This worked because there was only one mod_perl thread. With mod_perl 2, Win32 runs a true threaded MPM, which unfortunately means that we can't tell the size of each interpreter. Win32 support is disabled until a solution for this can be found.

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

67.4.2 Supported MPMs

At this time, `Apache2::SizeLimit` does not support use under threaded MPMs. This is because there is no efficient way to get the memory usage of a thread, or make a thread exit cleanly. Suggestions and patches are welcome on the mod_perl dev mailing list.

67.5 Copyright

67.6 Author

Doug Bagley <doug+modperl bagley.org>, channeling Procrustes.

Brian Moseley <ix maz.org>: Solaris 2.6 support

Doug Steinwand and Perrin Harkins <perrin elem.com>: added support for shared memory and additional diagnostic info

Matt Phillips <mphillips virage.com> and Mohamed Hendawi <mhendawi virage.com>: Win32 support

Torsten Foertsch <torsten.foertsch gmx.net>: Linux::Smaps support

68 ModPerl::BuildMM -- A "subclass" of ModPerl::MM used for building mod_perl 2.0

68.1 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;
}
```

68.2 DESCRIPTION

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

68.3 OVERRIDEN METHODS

`ModPerl::BuildMM` overrides the following methods:

68.3.1 ModPerl::BuildMM::MY::constants

68.3.2 ModPerl::BuildMM::MY::top_targets

68.3.3 ModPerl::BuildMM::MY::postamble

68.3.4 ModPerl::BuildMM::MY::post_initialize

68.3.5 ModPerl::BuildMM::MY::libscan

Table of Contents:

mod_perl 2.0 API 1
Apache2::Access - A Perl API for Apache request object: Access, Authentication and Authorization. 8
1 Apache2::Access - A Perl API for Apache request object: Access, Authentication and Authorization. 8
1.1 Synopsis 9
1.2 Description 9
1.3 API 10
1.3.1 allow_methods 10
1.3.2 allow_options 10
1.3.3 allow_overrides 11
1.3.4 allow_override_opts 11
1.3.5 auth_name 12
1.3.6 auth_type 13
1.3.7 get_basic_auth_pw 13
1.3.8 get_remote_logname 14
1.3.9 note_auth_failure 14
1.3.10 note_basic_auth_failure 14
1.3.11 note_digest_auth_failure 15
1.3.12 requires 15
1.3.13 satisfies 17
1.3.14 some_auth_required 17
1.4 See Also 18
1.5 Copyright 18
1.6 Authors 18
Apache2::CmdParms - Perl API for Apache command parameters object 19
2 Apache2::CmdParms - Perl API for Apache command parameters object 19
2.1 Synopsis 20
2.2 Description 21
2.3 API 21
2.3.1 add_config 21
2.3.2 check_cmd_context 21
2.3.3 cmd 22
2.3.4 directive 22
2.3.5 info 22
2.3.6 method_is_limited 23
2.3.7 override 23
2.3.8 override_opts 24
2.3.9 path 24
2.3.10 pool 25
2.3.11 server 25
2.3.12 temp_pool 25
2.4 Unsupported API 25
2.4.1 context 25

2.5 See Also 26
2.6 Copyright 26
2.7 Authors 26
Apache2::Command - Perl API for accessing Apache module command information . . . 27
3 Apache2::Command - Perl API for accessing Apache module command information . . . 27
3.1 Synopsis 28
3.2 Description 28
3.3 API 28
3.3.1 `args_how` 28
3.3.2 `errmsg` 28
3.3.3 `name` 29
3.3.4 `next` 29
3.3.5 `req_override` 29
3.4 See Also 30
3.5 Copyright 30
3.6 Authors 30
Apache2::Connection - Perl API for Apache connection object 31
4 Apache2::Connection - Perl API for Apache connection object 31
4.1 Synopsis 32
4.2 Description 33
4.3 API 33
4.3.1 `aborted` 33
4.3.2 `base_server` 33
4.3.3 `bucket_alloc` 33
4.3.4 `client_socket` 34
4.3.5 `get_remote_host` 34
4.3.6 `id` 35
4.3.7 `input_filters` 35
4.3.8 `keepalive` 36
4.3.9 `keepalives` 37
4.3.10 `local_addr` 37
4.3.11 `local_host` 38
4.3.12 `local_ip` 38
4.3.13 `notes` 38
4.3.14 `output_filters` 39
4.3.15 `pool` 39
4.3.16 `remote_addr` 39
4.3.17 `remote_ip` 39
4.3.18 `remote_host` 40
4.4 Unsupported API 40
4.4.1 `conn_config` 40
4.4.2 `sbh` 41
4.5 See Also 41
4.6 Copyright 41
4.7 Authors 41

Apache2::ConnectionUtil - Perl API for Apache connection utils 42
5 Apache2::ConnectionUtil - Perl API for Apache connection utils 42
5.1 Synopsis 43
5.2 Description 43
5.3 API 43
5.3.1 pnotes 43
5.4 See Also 44
5.5 Copyright 44
5.6 Authors 44
Apache2::Const - Perl Interface for Apache Constants 45
6 Apache2::Const - Perl Interface for Apache Constants 45
6.1 Synopsis 46
6.2 Description 46
6.3 Constants 47
6.3.1 :cmd_how 47
6.3.1.1 Apache2::Const::FLAG 47
6.3.1.2 Apache2::Const::ITERATE 47
6.3.1.3 Apache2::Const::ITERATE2 47
6.3.1.4 Apache2::Const::NO_ARGS 47
6.3.1.5 Apache2::Const::RAW_ARGS 47
6.3.1.6 Apache2::Const::TAKE1 48
6.3.1.7 Apache2::Const::TAKE12 48
6.3.1.8 Apache2::Const::TAKE123 48
6.3.1.9 Apache2::Const::TAKE13 48
6.3.1.10 Apache2::Const::TAKE2 48
6.3.1.11 Apache2::Const::TAKE23 48
6.3.1.12 Apache2::Const::TAKE3 48
6.3.2 :common 48
6.3.2.1 Apache2::Const::AUTH_REQUIRED 49
6.3.2.2 Apache2::Const::DECLINED 49
6.3.2.3 Apache2::Const::DONE 49
6.3.2.4 Apache2::Const::FORBIDDEN 49
6.3.2.5 Apache2::Const::NOT_FOUND 49
6.3.2.6 Apache2::Const::OK 49
6.3.2.7 Apache2::Const::REDIRECT 49
6.3.2.8 Apache2::Const::SERVER_ERROR 49
6.3.3 :config 49
6.3.3.1 Apache2::Const::DECLINE_CMD 49
6.3.4 :conn_keepalive 49
6.3.4.1 Apache2::Const::CONN_CLOSE 50
6.3.4.2 Apache2::Const::CONN_KEEPALIVE 50
6.3.4.3 Apache2::Const::CONN_UNKNOWN 50
6.3.5 :context 50
6.3.5.1 Apache2::Const::NOT_IN_VIRTUALHOST 50
6.3.5.2 Apache2::Const::NOT_IN_LIMIT 50
6.3.5.3 Apache2::Const::NOT_IN_DIRECTORY 50
6.3.5.4 Apache2::Const::NOT_IN_LOCATION 51

6.3.5.5 Apache2::Const::NOT_IN_FILES 51
6.3.5.6 Apache2::Const::NOT_IN_DIR_LOC_FILE 51
6.3.5.7 Apache2::Const::GLOBAL_ONLY 51
6.3.6 :filter_type 51
6.3.6.1 Apache2::Const::FTYPE_CONNECTION 51
6.3.6.2 Apache2::Const::FTYPE_CONTENT_SET 51
6.3.6.3 Apache2::Const::FTYPE_NETWORK 51
6.3.6.4 Apache2::Const::FTYPE_PROTOCOL 51
6.3.6.5 Apache2::Const::FTYPE_RESOURCE 52
6.3.6.6 Apache2::Const::FTYPE_TRANSCODE 52
6.3.7 :http 52
6.3.7.1 Apache2::Const::HTTP_ACCEPTED 52
6.3.7.2 Apache2::Const::HTTP_BAD_GATEWAY 52
6.3.7.3 Apache2::Const::HTTP_BAD_REQUEST 52
6.3.7.4 Apache2::Const::HTTP_CONFLICT 52
6.3.7.5 Apache2::Const::HTTP_CONTINUE 52
6.3.7.6 Apache2::Const::HTTP_CREATED 52
6.3.7.7 Apache2::Const::HTTP_EXPECTATION_FAILED 52
6.3.7.8 Apache2::Const::HTTP_FAILED_DEPENDENCY 52
6.3.7.9 Apache2::Const::HTTP_FORBIDDEN 53
6.3.7.10 Apache2::Const::HTTP_GATEWAY_TIME_OUT 53
6.3.7.11 Apache2::Const::HTTP_GONE 53
6.3.7.12 Apache2::Const::HTTP_INSUFFICIENT_STORAGE 53
6.3.7.13 Apache2::Const::HTTP_INTERNAL_SERVER_ERROR 53
6.3.7.14 Apache2::Const::HTTP_LENGTH_REQUIRED 53
6.3.7.15 Apache2::Const::HTTP_LOCKED 53
6.3.7.16 Apache2::Const::HTTP_METHOD_NOT_ALLOWED 53
6.3.7.17 Apache2::Const::HTTP_MOVED_PERMANENTLY 53
6.3.7.18 Apache2::Const::HTTP_MOVED_TEMPORARILY 53
6.3.7.19 Apache2::Const::HTTP_MULTIPLE_CHOICES 53
6.3.7.20 Apache2::Const::HTTP_MULTI_STATUS 54
6.3.7.21 Apache2::Const::HTTP_NON_AUTHORITATIVE 54
6.3.7.22 Apache2::Const::HTTP_NOT_ACCEPTABLE 54
6.3.7.23 Apache2::Const::HTTP_NOT_EXTENDED 54
6.3.7.24 Apache2::Const::HTTP_NOT_FOUND 54
6.3.7.25 Apache2::Const::HTTP_NOT_IMPLEMENTED 54
6.3.7.26 Apache2::Const::HTTP_NOT_MODIFIED 54
6.3.7.27 Apache2::Const::HTTP_NO_CONTENT 54
6.3.7.28 Apache2::Const::HTTP_OK 54
6.3.7.29 Apache2::Const::HTTP_PARTIAL_CONTENT 54
6.3.7.30 Apache2::Const::HTTP_PAYMENT_REQUIRED 54
6.3.7.31 Apache2::Const::HTTP_PRECONDITION_FAILED 55
6.3.7.32 Apache2::Const::HTTP_PROCESSING 55
6.3.7.33 Apache2::Const::HTTP_PROXY_AUTHENTICATION_REQUIRED . . 55
6.3.7.34 Apache2::Const::HTTP_RANGE_NOT_SATISFIABLE 55
6.3.7.35 Apache2::Const::HTTP_REQUEST_ENTITY_TOO_LARGE . . . 55
6.3.7.36 Apache2::Const::HTTP_REQUEST_TIME_OUT 55

6.3.7.37 Apache2::Const::HTTP_REQUEST_URI_TOO_LARGE 55
6.3.7.38 Apache2::Const::HTTP_RESET_CONTENT 55
6.3.7.39 Apache2::Const::HTTP_SEE_OTHER 55
6.3.7.40 Apache2::Const::HTTP_SERVICE_UNAVAILABLE 55
6.3.7.41 Apache2::Const::HTTP_SWITCHING_PROTOCOLS 55
6.3.7.42 Apache2::Const::HTTP_TEMPORARY_REDIRECT 56
6.3.7.43 Apache2::Const::HTTP_UNAUTHORIZED 56
6.3.7.44 Apache2::Const::HTTP_UNPROCESSABLE_ENTITY 56
6.3.7.45 Apache2::Const::HTTP_UNSUPPORTED_MEDIA_TYPE 56
6.3.7.46 Apache2::Const::HTTP_UPGRADE_REQUIRED 56
6.3.7.47 Apache2::Const::HTTP_USE_PROXY 56
6.3.7.48 Apache2::Const::HTTP_VARIANT_ALSO_VARIES 56
6.3.8 :input_mode 56
6.3.8.1 Apache2::Const::MODE_EATCRLF 56
6.3.8.2 Apache2::Const::MODE_EXHAUSTIVE 56
6.3.8.3 Apache2::Const::MODE_GETLINE 57
6.3.8.4 Apache2::Const::MODE_INIT 57
6.3.8.5 Apache2::Const::MODE_READBYTES 57
6.3.8.6 Apache2::Const::MODE_SPECULATIVE 57
6.3.9 :log 57
6.3.9.1 Apache2::Const::LOG_ALERT 57
6.3.9.2 Apache2::Const::LOG_CRIT 57
6.3.9.3 Apache2::Const::LOG_DEBUG 57
6.3.9.4 Apache2::Const::LOG_EMERG 58
6.3.9.5 Apache2::Const::LOG_ERR 58
6.3.9.6 Apache2::Const::LOG_INFO 58
6.3.9.7 Apache2::Const::LOG_LEVELMASK 58
6.3.9.8 Apache2::Const::LOG_NOTICE 58
6.3.9.9 Apache2::Const::LOG_STARTUP 58
6.3.9.10 Apache2::Const::LOG_TOCLIENT 58
6.3.9.11 Apache2::Const::LOG_WARNING 58
6.3.10 :methods 59
6.3.10.1 Apache2::Const::METHODS 59
6.3.10.2 Apache2::Const::M_BASELINE_CONTROL 59
6.3.10.3 Apache2::Const::M_CHECKIN 59
6.3.10.4 Apache2::Const::M_CHECKOUT 59
6.3.10.5 Apache2::Const::M_CONNECT 59
6.3.10.6 Apache2::Const::M_COPY 59
6.3.10.7 Apache2::Const::M_DELETE 59
6.3.10.8 Apache2::Const::M_GET 59
6.3.10.9 Apache2::Const::M_INVALID 59
6.3.10.10 Apache2::Const::M_LABEL 60
6.3.10.11 Apache2::Const::M_LOCK 60
6.3.10.12 Apache2::Const::M_MERGE 60
6.3.10.13 Apache2::Const::M_MKACTIVITY 60
6.3.10.14 Apache2::Const::M_MKCOL 60
6.3.10.15 Apache2::Const::M_MKWORKSPACE 60

6.3.10.16 Apache2::Const::M_MOVE 60
6.3.10.17 Apache2::Const::M_OPTIONS 60
6.3.10.18 Apache2::Const::M_PATCH 60
6.3.10.19 Apache2::Const::M_POST 60
6.3.10.20 Apache2::Const::M_PROPFIND 60
6.3.10.21 Apache2::Const::M_PROPPATCH 61
6.3.10.22 Apache2::Const::M_PUT 61
6.3.10.23 Apache2::Const::M_REPORT 61
6.3.10.24 Apache2::Const::M_TRACE 61
6.3.10.25 Apache2::Const::M_UNCHECKOUT 61
6.3.10.26 Apache2::Const::M_UNLOCK 61
6.3.10.27 Apache2::Const::M_UPDATE 61
6.3.10.28 Apache2::Const::M_VERSION_CONTROL 61
6.3.11 :mpmq 61
6.3.11.1 Apache2::Const::MPMQ_NOT_SUPPORTED 61
6.3.11.2 Apache2::Const::MPMQ_STATIC 62
6.3.11.3 Apache2::Const::MPMQ_DYNAMIC 62
6.3.11.4 Apache2::Const::MPMQ_MAX_DAEMON_USED 62
6.3.11.5 Apache2::Const::MPMQ_IS_THREADED 62
6.3.11.6 Apache2::Const::MPMQ_IS_FORKED 62
6.3.11.7 Apache2::Const::MPMQ_HARD_LIMIT_DAEMONS 62
6.3.11.8 Apache2::Const::MPMQ_HARD_LIMIT_THREADS 62
6.3.11.9 Apache2::Const::MPMQ_MAX_THREADS 62
6.3.11.10 Apache2::Const::MPMQ_MIN_SPARE_DAEMONS 62
6.3.11.11 Apache2::Const::MPMQ_MIN_SPARE_THREADS 62
6.3.11.12 Apache2::Const::MPMQ_MAX_SPARE_DAEMONS 62
6.3.11.13 Apache2::Const::MPMQ_MAX_SPARE_THREADS 63
6.3.11.14 Apache2::Const::MPMQ_MAX_REQUESTS_DAEMON 63
6.3.11.15 Apache2::Const::MPMQ_MAX_DAEMONS 63
6.3.12 :options 63
6.3.12.1 Apache2::Const::OPT_ALL 63
6.3.12.2 Apache2::Const::OPT_EXECCGI 63
6.3.12.3 Apache2::Const::OPT_INCLUDES 63
6.3.12.4 Apache2::Const::OPT_INCNOEXEC 63
6.3.12.5 Apache2::Const::OPT_INDEXES 63
6.3.12.6 Apache2::Const::OPT_MULTI 63
6.3.12.7 Apache2::Const::OPT_NONE 64
6.3.12.8 Apache2::Const::OPT_SYM_LINKS 64
6.3.12.9 Apache2::Const::OPT_SYM_OWNER 64
6.3.12.10 Apache2::Const::OPT_UNSET 64
6.3.13 :override 64
6.3.13.1 Apache2::Const::ACCESS_CONF 64
6.3.13.2 Apache2::Const::EXEC_ON_READ 64
6.3.13.3 Apache2::Const::OR_ALL 64
6.3.13.4 Apache2::Const::OR_AUTHCFG 65
6.3.13.5 Apache2::Const::OR_FILEINFO 65
6.3.13.6 Apache2::Const::OR_INDEXES 65

6.3.13.7 Apache2::Const::OR_LIMIT 65
6.3.13.8 Apache2::Const::OR_NONE 65
6.3.13.9 Apache2::Const::OR_OPTIONS 65
6.3.13.10 Apache2::Const::OR_UNSET 65
6.3.13.11 Apache2::Const::RSRC_CONF 65
6.3.14 :platform 66
6.3.14.1 Apache2::Const::CRLF 66
6.3.14.2 Apache2::Const::CR 66
6.3.14.3 Apache2::Const::LF 66
6.3.15 :remotehost 66
6.3.15.1 Apache2::Const::REMOTE_DOUBLE_REV 66
6.3.15.2 Apache2::Const::REMOTE_HOST 66
6.3.15.3 Apache2::Const::REMOTE_NAME 66
6.3.15.4 Apache2::Const::REMOTE_NOLOOKUP 66
6.3.16 :satisfy 66
6.3.16.1 Apache2::Const::SATISFY_ALL 67
6.3.16.2 Apache2::Const::SATISFY_ANY 67
6.3.16.3 Apache2::Const::SATISFY_NOSPEC 67
6.3.17 :types 67
6.3.17.1 Apache2::Const::DIR_MAGIC_TYPE 67
6.3.18 :proxy 67
6.3.18.1 Apache2::Const::PROXYREQ_NONE 67
6.3.18.2 Apache2::Const::PROXYREQ_PROXY 67
6.3.18.3 Apache2::Const::PROXYREQ_REVERSE 67
6.3.18.4 Apache2::Const::PROXYREQ_RESPONSE 68
6.4 See Also 68
6.5 Copyright 68
6.6 Authors 68
Apache2::Directive - Perl API for manipulating the Apache configuration tree 69
7 Apache2::Directive - Perl API for manipulating the Apache configuration tree 69
7.1 Synopsis 70
7.2 Description 70
7.3 API 70
7.3.1 args 71
7.3.2 as_hash 71
7.3.3 as_string 72
7.3.4 conftree 72
7.3.5 directive 73
7.3.6 filename 73
7.3.7 first_child 73
7.3.8 line_num 74
7.3.9 lookup 74
7.3.10 next 75
7.3.11 parent 75
7.4 See Also 75
7.5 Copyright 75
7.6 Authors 75

Apache2::Filter - Perl API for Apache 2.0 Filtering 76
8 Apache2::Filter - Perl API for Apache 2.0 Filtering 76
8.1 Synopsis 77
8.2 Description 77
8.3 Common Filter API 77
8.3.1 c 77
8.3.2 ctx 78
8.3.3 frec 79
8.3.4 next 79
8.3.5 r 80
8.3.6 remove 80
8.4 Bucket Brigade Filter API 80
8.4.1 fflush 81
8.4.2 get_brigade 81
8.4.3 pass_brigade 84
8.5 Streaming Filter API 86
8.5.1 print 86
8.5.2 read 86
8.5.3 seen_eos 87
8.6 Other Filter-related API 88
8.6.1 add_input_filter 88
8.6.2 add_output_filter 89
8.7 Filter Handler Attributes 89
8.7.1 FilterRequestHandler 89
8.7.2 FilterConnectionHandler 90
8.7.3 FilterInitHandler 90
8.7.4 FilterHasInitHandler 90
8.8 Configuration 91
8.8.1 PerlInputFilterHandler 91
8.8.2 PerlOutputFilterHandler 91
8.8.3 PerlSetInputFilter 91
8.8.4 PerlSetOutputFilter 91
8.9 TIE Interface 91
8.9.1 TIEHANDLE 91
8.9.2 PRINT 91
8.10 See Also 92
8.11 Copyright 92
8.12 Authors 92
Apache2::FilterRec - Perl API for manipulating the Apache filter record 93
9 Apache2::FilterRec - Perl API for manipulating the Apache filter record 93
9.1 Synopsis 94
9.2 Description 94
9.3 API 94
9.3.1 name 94
9.4 See Also 95
9.5 Copyright 95
9.6 Authors 95

Apache2::HookRun - Perl API for Invoking Apache HTTP phases 96
10 Apache2::HookRun - Perl API for Invoking Apache HTTP phases 96
10.1 Synopsis 97
10.2 Description 98
10.3 API 98
10.3.1 `die` 98
10.3.2 `invoke_handler` 99
10.3.3 `run_access_checker` 99
10.3.4 `run_auth_checker` 100
10.3.5 `run_check_user_id` 100
10.3.6 `run_fixups` 101
10.3.7 `run_handler` 101
10.3.8 `run_header_parser` 102
10.3.9 `run_log_transaction` 102
10.3.10 `run_map_to_storage` 102
10.3.11 `run_post_read_request` 103
10.3.12 `run_translate_name` 103
10.3.13 `run_type_checker` 104
10.4 See Also 104
10.5 Copyright 104
10.6 Authors 104
Apache2::Log - Perl API for Apache Logging Methods 105
11 Apache2::Log - Perl API for Apache Logging Methods 105
11.1 Synopsis 106
11.2 Description 107
11.3 Constants 107
11.3.1 LogLevel Constants 108
11.3.1.1 `Apache2::Const::LOG_EMERG` 108
11.3.1.2 `Apache2::Const::LOG_ALERT` 108
11.3.1.3 `Apache2::Const::LOG_CRIT` 108
11.3.1.4 `Apache2::Const::LOG_ERR` 108
11.3.1.5 `Apache2::Const::LOG_WARNING` 108
11.3.1.6 `Apache2::Const::LOG_NOTICE` 108
11.3.1.7 `Apache2::Const::LOG_INFO` 108
11.3.1.8 `Apache2::Const::LOG_DEBUG` 108
11.3.2 Other Constants 108
11.3.2.1 `Apache2::Const::LOG_LEVELMASK` 108
11.3.2.2 `Apache2::Const::LOG_TOCLIENT` 108
11.3.2.3 `Apache2::Const::LOG_STARTUP` 109
11.4 Server Logging Methods 109
11.4.1 `$s->log` 109
11.4.2 `$s->log_error` 110
11.4.3 `$s->log_serror` 110
11.4.4 `$s->warn` 111
11.5 Request Logging Methods 111
11.5.1 `$r->log` 111
11.5.2 `$r->log_error` 112

11.5.3 $r->log_reason 112
11.5.4 $r->log_rerror 113
11.5.5 $r->warn 113
11.6 Other Logging Methods 113
11.6.1 LogLevel Methods 114
11.6.2 alert 114
11.6.3 crit 114
11.6.4 debug 114
11.6.5 emerg 114
11.6.6 error 115
11.6.7 info 115
11.6.8 notice 115
11.6.9 warn 115
11.7 General Functions 115
11.7.1 LOG_MARK 115
11.8 Virtual Hosts 115
11.9 Unsupported API 117
11.9.1 log_pid 117
11.10 See Also 117
11.11 Copyright 117
11.12 Authors 117
Apache2::MPM - Perl API for accessing Apache MPM information 118
12 Apache2::MPM - Perl API for accessing Apache MPM information 118
12.1 Synopsis 119
12.2 Description 119
12.3 API 119
12.3.1 query 119
12.3.2 is_threaded 120
12.3.3 show 120
12.4 See Also 120
12.5 Copyright 120
12.6 Authors 121
Apache2::Module - Perl API for creating and working with Apache modules 122
13 Apache2::Module - Perl API for creating and working with Apache modules 122
13.1 Synopsis 123
13.2 Description 123
13.3 API 124
13.3.1 add 124
13.3.2 ap_api_major_version 124
13.3.3 ap_api_minor_version 124
13.3.4 cmds 125
13.3.5 get_config 125
13.3.6 find_linked_module 126
13.3.7 loaded 126
13.3.8 module_index 127
13.3.9 name 127
13.3.10 next 127

13.3.11 remove_loaded_module . . . 128
13.3.12 top_module . . . 128
13.4 See Also . . . 128
13.5 Copyright . . . 128
13.6 Authors . . . 128
Apache2::PerlSections - write Apache configuration files in Perl . . . 129
14 Apache2::PerlSections - write Apache configuration files in Perl . . . 129
14.1 Synopsis . . . 130
14.2 Description . . . 130
14.3 API . . . 131
14.3.1 server . . . 131
14.4 @PerlConfig and $PerlConfig . . . 132
14.5 Configuration Variables . . . 132
14.5.1 $Apache2::PerlSections::Save . . . 132
14.6 PerlSections Dumping . . . 132
14.6.1 Apache2::PerlSections->dump . . . 133
14.6.2 Apache2::PerlSections->store . . . 134
14.7 Advanced API . . . 134
14.8 Verifying <Perl> Sections . . . 135
14.9 Bugs . . . 136
14.9.1 <Perl> directive missing closing '>' . . . 136
14.9.2 <Perl>[...]> was not closed. . . . 136
14.10 See Also . . . 137
14.11 Copyright . . . 137
14.12 Authors . . . 137
Apache2::Process - Perl API for Apache process record . . . 138
15 Apache2::Process - Perl API for Apache process record . . . 138
15.1 Synopsis . . . 139
15.2 Description . . . 139
15.3 API . . . 139
15.3.1 pconf . . . 139
15.3.2 pool . . . 139
15.3.3 short_name . . . 140
15.4 See Also . . . 140
15.5 Copyright . . . 140
15.6 Authors . . . 140
Apache2::RequestIO - Perl API for Apache request record IO . . . 141
16 Apache2::RequestIO - Perl API for Apache request record IO . . . 141
16.1 Synopsis . . . 142
16.2 Description . . . 142
16.3 API . . . 142
16.3.1 discard_request_body . . . 142
16.3.2 print . . . 143
16.3.3 printf . . . 143
16.3.4 puts . . . 144
16.3.5 read . . . 144
16.3.6 rflush . . . 145

16.3.7 sendfile 145
16.3.8 write 146
16.4 TIE Interface 147
16.4.1 BINMODE 147
16.4.2 CLOSE 148
16.4.3 FILENO 148
16.4.4 GETC 148
16.4.5 OPEN 148
16.4.6 PRINT 148
16.4.7 PRINTF 148
16.4.8 READ 148
16.4.9 TIEHANDLE 149
16.4.10 UNTIE 149
16.4.11 WRITE 149
16.5 Deprecated API 149
16.5.1 get_client_block 149
16.5.2 setup_client_block 149
16.5.3 should_client_block 149
16.6 See Also 149
16.7 Copyright 150
16.8 Authors 150
Apache2::RequestRec - Perl API for Apache request record accessors 151
17 Apache2::RequestRec - Perl API for Apache request record accessors 151
17.1 Synopsis 152
17.2 Description 154
17.3 API 154
17.3.1 allowed 154
17.3.2 ap_auth_type 155
17.3.3 args 156
17.3.4 assbackwards 156
17.3.5 bytes_sent 157
17.3.6 connection 157
17.3.7 content_encoding 158
17.3.8 content_languages 158
17.3.9 content_type 159
17.3.10 err_headers_out 159
17.3.11 filename 160
17.3.12 finfo 160
17.3.13 handler 161
17.3.14 header_only 162
17.3.15 headers_in 162
17.3.16 headers_out 162
17.3.17 hostname 162
17.3.18 input_filters 163
17.3.19 main 164
17.3.20 method 165
17.3.21 method_number 165

17.3.22 mtime 165
17.3.23 next 166
17.3.24 no_local_copy 166
17.3.25 notes 167
17.3.26 output_filters 167
17.3.27 path_info 168
17.3.28 per_dir_config 168
17.3.29 pool 169
17.3.30 prev 169
17.3.31 proto_input_filters 169
17.3.32 proto_num 170
17.3.33 proto_output_filters 170
17.3.34 protocol 170
17.3.35 proxyreq 171
17.3.36 request_time 172
17.3.37 server 172
17.3.38 status 172
17.3.39 status_line 173
17.3.40 subprocess_env 174
17.3.41 the_request 175
17.3.42 unparsed_uri 175
17.3.43 uri 175
17.3.44 user 176
17.4 Unsupported API 176
17.4.1 allowed_methods 176
17.4.2 allowed_xmethods 177
17.4.3 request_config 177
17.4.4 used_path_info 177
17.5 See Also 178
17.6 Copyright 178
17.7 Authors 178
Apache2::RequestUtil - Perl API for Apache request record utils 179
18 Apache2::RequestUtil - Perl API for Apache request record utils 179
18.1 Synopsis 180
18.2 Description 181
18.3 API 181
18.3.1 add_config 181
18.3.2 as_string 183
18.3.3 child_terminate 183
18.3.4 default_type 183
18.3.5 dir_config 184
18.3.6 document_root 185
18.3.7 get_handlers 185
18.3.8 get_limit_req_body 186
18.3.9 get_server_name 186
18.3.10 get_server_port 187
18.3.11 get_status_line 187

18.3.12 is_initial_req . . . 188
18.3.13 is_perl_option_enabled . . . 188
18.3.14 location . . . 188
18.3.15 location_merge . . . 189
18.3.16 new . . . 189
18.3.17 no_cache . . . 190
18.3.18 pnotes . . . 190
18.3.19 psignature . . . 193
18.3.20 request . . . 193
18.3.21 push_handlers . . . 194
18.3.22 set_basic_credentials . . . 194
18.3.23 set_handlers . . . 195
18.3.24 slurp_filename . . . 196
18.4 See Also . . . 197
18.5 Copyright . . . 197
18.6 Authors . . . 197
Apache2::Response - Perl API for Apache HTTP request response methods . . . 198
19 Apache2::Response - Perl API for Apache HTTP request response methods . . . 198
19.1 Synopsis . . . 199
19.2 Description . . . 199
19.3 API . . . 199
19.3.1 custom_response . . . 199
19.3.2 make_etag . . . 200
19.3.3 meets_conditions . . . 201
19.3.4 rationalize_mtime . . . 201
19.3.5 send_cgi_header . . . 202
19.3.6 set_content_length . . . 202
19.3.7 set_etag . . . 202
19.3.8 set_keepalive . . . 203
19.3.9 set_last_modified . . . 203
19.3.10 update_mtime . . . 203
19.4 Unsupported API . . . 204
19.4.1 send_error_response . . . 204
19.4.2 send_mmap . . . 204
19.5 See Also . . . 205
19.6 Copyright . . . 205
19.7 Authors . . . 205
Apache2::ServerRec - Perl API for Apache server record accessors . . . 206
20 Apache2::ServerRec - Perl API for Apache server record accessors . . . 206
20.1 Synopsis . . . 207
20.2 Description . . . 207
20.3 API . . . 207
20.3.1 error_fname . . . 207
20.3.2 is_virtual . . . 208
20.3.3 keep_alive . . . 208
20.3.4 keep_alive_max . . . 209
20.3.5 keep_alive_timeout . . . 209

20.3.6 limit_req_fields 210
20.3.7 limit_req_fieldsize 210
20.3.8 limit_req_line 211
20.3.9 loglevel 211
20.3.10 next 212
20.3.11 path 212
20.3.12 port 213
20.3.13 process 213
20.3.14 server_admin 213
20.3.15 server_hostname 214
20.3.16 timeout 214
20.4 Notes 215
20.4.1 Limited Functionality under Threaded MPMs 215
20.5 Unsupported API 215
20.5.1 addrs 215
20.5.2 lookup_defaults 216
20.5.3 module_config 216
20.5.4 names 216
20.5.5 wild_names 217
20.6 See Also 217
20.7 Copyright 217
20.8 Authors 217
Apache2::ServerUtil - Perl API for Apache server record utils 218
21 Apache2::ServerUtil - Perl API for Apache server record utils 218
21.1 Synopsis 219
21.2 Description 220
21.3 Methods API 220
21.3.1 add_config 220
21.3.2 add_version_component 220
21.3.3 dir_config 221
21.3.4 exists_config_define 222
21.3.5 get_handlers 223
21.3.6 get_server_built 223
21.3.7 get_server_version 224
21.3.8 get_server_banner 224
21.3.9 get_server_description 224
21.3.10 group_id 224
21.3.11 is_perl_option_enabled 225
21.3.12 method_register 225
21.3.13 push_handlers 225
21.3.14 restart_count 226
21.3.15 server 228
21.3.16 server_root 228
21.3.17 server_root_relative 228
21.3.18 server_shutdown_cleanup_register 229
21.3.19 set_handlers 230
21.3.20 user_id 231

21.4 Unsupported API 231
21.4.1 error_log2stderr 231
21.5 See Also 232
21.6 Copyright 232
21.7 Authors 232
Apache2::SubProcess -- Executing SubProcesses under mod_perl 233
22 Apache2::SubProcess -- Executing SubProcesses under mod_perl 233
22.1 Synopsis 234
22.2 Description 234
22.3 API 235
22.3.1 spawn_proc_prog 235
22.4 See Also 237
22.5 Copyright 237
22.6 Authors 237
Apache2::SubRequest - Perl API for Apache subrequests 238
23 Apache2::SubRequest - Perl API for Apache subrequests 238
23.1 Synopsis 239
23.2 Description 239
23.3 API 239
23.3.1 DESTROY 239
23.3.2 internal_redirect 240
23.3.3 internal_redirect_handler 240
23.3.4 lookup_file 241
23.3.5 lookup_method_uri 241
23.3.6 lookup_uri 242
23.3.7 run 243
23.4 Unsupported API 243
23.4.1 internal_fast_redirect 243
23.4.2 lookup_dirent 244
23.5 See Also 245
23.6 Copyright 245
23.7 Authors 245
Apache2::URI - Perl API for manipulating URIs 246
24 Apache2::URI - Perl API for manipulating URIs 246
24.1 Synopsis 247
24.2 Description 247
24.3 API 247
24.3.1 construct_server 247
24.3.2 construct_url 248
24.3.3 parse_uri 250
24.3.4 parsed_uri 250
24.3.5 unescape_url 251
24.4 See Also 251
24.5 Copyright 251
24.6 Authors 251

Apache2::Util - Perl API for Misc Apache Utility functions 252
25 Apache2::Util - Perl API for Misc Apache Utility functions 252
25.1 Synopsis 253
25.2 Description 253
25.3 Functions API 253
25.3.1 escape_path 253
25.3.2 ht_time 254
25.4 See Also 255
25.5 Copyright 255
25.6 Authors 255
APR - Perl Interface for Apache Portable Runtime (libapr and libaprutil Libraries) . . . 256
26 APR - Perl Interface for Apache Portable Runtime (libapr and libaprutil Libraries) . . . 256
26.1 Synopsis 257
26.2 Description 257
26.3 Using APR modules outside mod_perl 2.0 257
26.4 See Also 257
26.5 Copyright 257
26.6 Authors 257
APR::Base64 - Perl API for APR base64 encoding/decoding functionality 258
27 APR::Base64 - Perl API for APR base64 encoding/decoding functionality 258
27.1 Synopsis 259
27.2 Description 259
27.3 API 259
27.3.1 decode 259
27.3.2 encode 259
27.3.3 encode_len 260
27.4 See Also 260
27.5 Copyright 260
27.6 Authors 260
APR::Brigade - Perl API for manipulating APR Bucket Brigades 261
28 APR::Brigade - Perl API for manipulating APR Bucket Brigades 261
28.1 Synopsis 262
28.2 Description 262
28.3 API 262
28.3.1 cleanup 262
28.3.2 concat 263
28.3.3 destroy 263
28.3.4 is_empty 263
28.3.5 first 264
28.3.6 flatten 264
28.3.7 insert_head 264
28.3.8 insert_tail 265
28.3.9 last 265
28.3.10 length 266
28.3.11 new 266
28.3.12 bucket_alloc 266
28.3.13 next 266

28.3.14 pool 267
28.3.15 prev 267
28.3.16 split 268
28.4 See Also 268
28.5 Copyright 269
28.6 Authors 269
APR::Bucket - Perl API for manipulating APR Buckets 270
29 APR::Bucket - Perl API for manipulating APR Buckets 270
29.1 Synopsis 271
29.2 Description 271
29.3 API 271
29.3.1 delete 272
29.3.2 destroy 272
29.3.3 eos_create 273
29.3.4 flush_create 273
29.3.5 insert_after 274
29.3.6 insert_before 274
29.3.7 is_eos 274
29.3.8 is_flush 275
29.3.9 length 275
29.3.10 new 275
29.3.11 read 276
29.3.12 remove 277
29.3.13 setaside 278
29.3.14 type 279
29.4 Unsupported API 279
29.4.1 data 279
29.4.2 start 280
29.5 See Also 280
29.6 Copyright 280
29.7 Authors 280
APR::BucketAlloc - Perl API for Bucket Allocation 281
30 APR::BucketAlloc - Perl API for Bucket Allocation 281
30.1 Synopsis 282
30.2 Description 282
30.2.1 new 282
30.2.2 destroy 282
30.3 See Also 283
30.4 Copyright 283
30.5 Authors 283
APR::BucketType - Perl API for APR bucket types 284
31 APR::BucketType - Perl API for APR bucket types 284
31.1 Synopsis 285
31.2 Description 285
31.3 API 285
31.3.1 name 285
31.4 See Also 285

31.5 Copyright 285
31.6 Authors 286
APR::Const - Perl Interface for APR Constants 287
32 APR::Const - Perl Interface for APR Constants 287
32.1 Synopsis 288
32.2 Description 288
32.3 Constants 288
32.3.1 :common 288
32.3.1.1 APR::Const::SUCCESS 288
32.3.2 :error 288
32.3.2.1 APR::Const::EABOVEROOT 288
32.3.2.2 APR::Const::EABSOLUTE 288
32.3.2.3 APR::Const::EACCES 289
32.3.2.4 APR::Const::EAGAIN 289
32.3.2.5 APR::Const::EBADDATE 289
32.3.2.6 APR::Const::EBADF 289
32.3.2.7 APR::Const::EBADIP 289
32.3.2.8 APR::Const::EBADMASK 289
32.3.2.9 APR::Const::EBADPATH 289
32.3.2.10 APR::Const::EBUSY 289
32.3.2.11 APR::Const::ECONNABORTED 289
32.3.2.12 APR::Const::ECONNREFUSED 290
32.3.2.13 APR::Const::ECONNRESET 290
32.3.2.14 APR::Const::EDSOOPEN 290
32.3.2.15 APR::Const::EEXIST 290
32.3.2.16 APR::Const::EFTYPE 290
32.3.2.17 APR::Const::EGENERAL 290
32.3.2.18 APR::Const::EHOSTUNREACH 290
32.3.2.19 APR::Const::EINCOMPLETE 290
32.3.2.20 APR::Const::EINIT 290
32.3.2.21 APR::Const::EINPROGRESS 290
32.3.2.22 APR::Const::EINTR 291
32.3.2.23 APR::Const::EINVAL 291
32.3.2.24 APR::Const::EINVALSOCK 291
32.3.2.25 APR::Const::EMFILE 291
32.3.2.26 APR::Const::EMISMATCH 291
32.3.2.27 APR::Const::ENAMETOOLONG 291
32.3.2.28 APR::Const::END 291
32.3.2.29 APR::Const::ENETUNREACH 291
32.3.2.30 APR::Const::ENFILE 291
32.3.2.31 APR::Const::ENODIR 291
32.3.2.32 APR::Const::ENOENT 291
32.3.2.33 APR::Const::ENOLOCK 292
32.3.2.34 APR::Const::ENOMEM 292
32.3.2.35 APR::Const::ENOPOLL 292
32.3.2.36 APR::Const::ENOPOOL 292
32.3.2.37 APR::Const::ENOPROC 292

32.3.2.38 APR::Const::ENOSHMAVAIL 292
32.3.2.39 APR::Const::ENOSOCKET 292
32.3.2.40 APR::Const::ENOSPC 292
32.3.2.41 APR::Const::ENOSTAT 292
32.3.2.42 APR::Const::ENOTDIR 292
32.3.2.43 APR::Const::ENOTEMPTY 292
32.3.2.44 APR::Const::ENOTHDKEY 293
32.3.2.45 APR::Const::ENOTHREAD 293
32.3.2.46 APR::Const::ENOTIME 293
32.3.2.47 APR::Const::ENOTIMPL 293
32.3.2.48 APR::Const::ENOTSOCK 293
32.3.2.49 APR::Const::EOF 293
32.3.2.50 APR::Const::EPATHWILD 293
32.3.2.51 APR::Const::EPIPE 293
32.3.2.52 APR::Const::EPROC_UNKNOWN 293
32.3.2.53 APR::Const::ERELATIVE 293
32.3.2.54 APR::Const::ESPIPE 294
32.3.2.55 APR::Const::ESYMNOTFOUND 294
32.3.2.56 APR::Const::ETIMEDOUT 294
32.3.2.57 APR::Const::EXDEV 294
32.3.3 :fopen 294
32.3.3.1 APR::Const::FOPEN_BINARY 294
32.3.3.2 APR::Const::FOPEN_BUFFERED 294
32.3.3.3 APR::Const::FOPEN_CREATE 294
32.3.3.4 APR::Const::FOPEN_DELONCLOSE 294
32.3.3.5 APR::Const::FOPEN_EXCL 294
32.3.3.6 APR::Const::FOPEN_PEND 294
32.3.3.7 APR::Const::FOPEN_READ 295
32.3.3.8 APR::Const::FOPEN_TRUNCATE 295
32.3.3.9 APR::Const::FOPEN_WRITE 295
32.3.4 :filepath 295
32.3.4.1 APR::Const::FILEPATH_ENCODING_LOCALE 295
32.3.4.2 APR::Const::FILEPATH_ENCODING_UNKNOWN 295
32.3.4.3 APR::Const::FILEPATH_ENCODING_UTF8 295
32.3.4.4 APR::Const::FILEPATH_NATIVE 295
32.3.4.5 APR::Const::FILEPATH_NOTABOVEROOT 295
32.3.4.6 APR::Const::FILEPATH_NOTABSOLUTE 295
32.3.4.7 APR::Const::FILEPATH_NOTRELATIVE 295
32.3.4.8 APR::Const::FILEPATH_SECUREROOT 296
32.3.4.9 APR::Const::FILEPATH_SECUREROOTTEST 296
32.3.4.10 APR::Const::FILEPATH_TRUENAME 296
32.3.5 :fprot 296
32.3.5.1 APR::Const::FPROT_GEXECUTE 296
32.3.5.2 APR::Const::FPROT_GREAD 296
32.3.5.3 APR::Const::FPROT_GSETID 296
32.3.5.4 APR::Const::FPROT_GWRITE 296
32.3.5.5 APR::Const::FPROT_OS_DEFAULT 296

32.3.5.6 APR::Const::FPROT_UEXECUTE 297
32.3.5.7 APR::Const::FPROT_UREAD 297
32.3.5.8 APR::Const::FPROT_USETID 297
32.3.5.9 APR::Const::FPROT_UWRITE 297
32.3.5.10 APR::Const::FPROT_WEXECUTE 297
32.3.5.11 APR::Const::FPROT_WREAD 297
32.3.5.12 APR::Const::FPROT_WSTICKY 297
32.3.5.13 APR::Const::FPROT_WWRITE 298
32.3.6 :filetype 298
32.3.6.1 APR::Const::FILETYPE_BLK 298
32.3.6.2 APR::Const::FILETYPE_CHR 298
32.3.6.3 APR::Const::FILETYPE_DIR 298
32.3.6.4 APR::Const::FILETYPE_LNK 298
32.3.6.5 APR::Const::FILETYPE_NOFILE 298
32.3.6.6 APR::Const::FILETYPE_PIPE 298
32.3.6.7 APR::Const::FILETYPE_REG 299
32.3.6.8 APR::Const::FILETYPE_SOCK 299
32.3.6.9 APR::Const::FILETYPE_UNKFILE 299
32.3.7 :finfo 299
32.3.7.1 APR::Const::FINFO_ATIME 299
32.3.7.2 APR::Const::FINFO_CSIZE 299
32.3.7.3 APR::Const::FINFO_CTIME 299
32.3.7.4 APR::Const::FINFO_DEV 299
32.3.7.5 APR::Const::FINFO_DIRENT 300
32.3.7.6 APR::Const::FINFO_GPROT 300
32.3.7.7 APR::Const::FINFO_GROUP 300
32.3.7.8 APR::Const::FINFO_ICASE 300
32.3.7.9 APR::Const::FINFO_IDENT 300
32.3.7.10 APR::Const::FINFO_INODE 300
32.3.7.11 APR::Const::FINFO_LINK 300
32.3.7.12 APR::Const::FINFO_MIN 300
32.3.7.13 APR::Const::FINFO_MTIME 301
32.3.7.14 APR::Const::FINFO_NAME 301
32.3.7.15 APR::Const::FINFO_NLINK 301
32.3.7.16 APR::Const::FINFO_NORM 301
32.3.7.17 APR::Const::FINFO_OWNER 301
32.3.7.18 APR::Const::FINFO_PROT 301
32.3.7.19 APR::Const::FINFO_SIZE 301
32.3.7.20 APR::Const::FINFO_TYPE 301
32.3.7.21 APR::Const::FINFO_UPROT 302
32.3.7.22 APR::Const::FINFO_USER 302
32.3.7.23 APR::Const::FINFO_WPROT 302
32.3.8 :flock 302
32.3.8.1 APR::Const::FLOCK_EXCLUSIVE 302
32.3.8.2 APR::Const::FLOCK_NONBLOCK 302
32.3.8.3 APR::Const::FLOCK_SHARED 302
32.3.8.4 APR::Const::FLOCK_TYPEMASK 302

32.3.9 :hook 302
32.3.9.1 APR::Const::HOOK_FIRST 303
32.3.9.2 APR::Const::HOOK_LAST 303
32.3.9.3 APR::Const::HOOK_MIDDLE 303
32.3.9.4 APR::Const::HOOK_REALLY_FIRST 303
32.3.9.5 APR::Const::HOOK_REALLY_LAST 303
32.3.10 :limit 303
32.3.10.1 APR::Const::LIMIT_CPU 303
32.3.10.2 APR::Const::LIMIT_MEM 303
32.3.10.3 APR::Const::LIMIT_NOFILE 303
32.3.10.4 APR::Const::LIMIT_NPROC 303
32.3.11 :lockmech 303
32.3.11.1 APR::Const::LOCK_DEFAULT 304
32.3.11.2 APR::Const::LOCK_FCNTL 304
32.3.11.3 APR::Const::LOCK_FLOCK 304
32.3.11.4 APR::Const::LOCK_POSIXSEM 304
32.3.11.5 APR::Const::LOCK_PROC_PTHREAD 304
32.3.11.6 APR::Const::LOCK_SYSVSEM 304
32.3.12 :poll 304
32.3.12.1 APR::Const::POLLERR 304
32.3.12.2 APR::Const::POLLHUP 304
32.3.12.3 APR::Const::POLLIN 305
32.3.12.4 APR::Const::POLLNVAL 305
32.3.12.5 APR::Const::POLLOUT 305
32.3.12.6 APR::Const::POLLPRI 305
32.3.13 :read_type 305
32.3.13.1 APR::Const::BLOCK_READ 305
32.3.13.2 APR::Const::NONBLOCK_READ 305
32.3.14 :shutdown_how 305
32.3.14.1 APR::Const::SHUTDOWN_READ 306
32.3.14.2 APR::Const::SHUTDOWN_READWRITE 306
32.3.14.3 APR::Const::SHUTDOWN_WRITE 306
32.3.15 :socket 306
32.3.15.1 APR::Const::SO_DEBUG 306
32.3.15.2 APR::Const::SO_DISCONNECTED 306
32.3.15.3 APR::Const::SO_KEEPALIVE 306
32.3.15.4 APR::Const::SO_LINGER 307
32.3.15.5 APR::Const::SO_NONBLOCK 307
32.3.15.6 APR::Const::SO_RCVBUF 307
32.3.15.7 APR::Const::SO_REUSEADDR 307
32.3.15.8 APR::Const::SO_SNDBUF 308
32.3.16 :status 308
32.3.16.1 APR::Const::TIMEUP 308
32.3.17 :table 308
32.3.17.1 APR::Const::OVERLAP_TABLES_MERGE 308
32.3.17.2 APR::Const::OVERLAP_TABLES_SET 308
32.3.18 :uri 309

32.3.18.1 APR::Const::URI_ACAP_DEFAULT_PORT 309
32.3.18.2 APR::Const::URI_FTP_DEFAULT_PORT 309
32.3.18.3 APR::Const::URI_GOPHER_DEFAULT_PORT 309
32.3.18.4 APR::Const::URI_HTTPS_DEFAULT_PORT 309
32.3.18.5 APR::Const::URI_HTTP_DEFAULT_PORT 309
32.3.18.6 APR::Const::URI_IMAP_DEFAULT_PORT 309
32.3.18.7 APR::Const::URI_LDAP_DEFAULT_PORT 309
32.3.18.8 APR::Const::URI_NFS_DEFAULT_PORT 309
32.3.18.9 APR::Const::URI_NNTP_DEFAULT_PORT 309
32.3.18.10 APR::Const::URI_POP_DEFAULT_PORT 310
32.3.18.11 APR::Const::URI_PROSPERO_DEFAULT_PORT 310
32.3.18.12 APR::Const::URI_RTSP_DEFAULT_PORT 310
32.3.18.13 APR::Const::URI_SIP_DEFAULT_PORT 310
32.3.18.14 APR::Const::URI_SNEWS_DEFAULT_PORT 310
32.3.18.15 APR::Const::URI_SSH_DEFAULT_PORT 310
32.3.18.16 APR::Const::URI_TELNET_DEFAULT_PORT 310
32.3.18.17 APR::Const::URI_TIP_DEFAULT_PORT 310
32.3.18.18 APR::Const::URI_UNP_OMITPASSWORD 310
32.3.18.19 APR::Const::URI_UNP_OMITPATHINFO 310
32.3.18.20 APR::Const::URI_UNP_OMITQUERY 311
32.3.18.21 APR::Const::URI_UNP_OMITSITEPART 311
32.3.18.22 APR::Const::URI_UNP_OMITUSER 311
32.3.18.23 APR::Const::URI_UNP_OMITUSERINFO 311
32.3.18.24 APR::Const::URI_UNP_REVEALPASSWORD 311
32.3.18.25 APR::Const::URI_WAIS_DEFAULT_PORT 311
32.3.19 Other Constants 311
32.3.19.1 APR::PerlIO::PERLIO_LAYERS_ARE_ENABLED 311
32.4 See Also 311
32.5 Copyright 312
32.6 Authors 312
APR::Date - Perl API for APR date manipulating functions 313
33 APR::Date - Perl API for APR date manipulating functions 313
33.1 Synopsis 314
33.2 Description 314
33.3 API 314
33.3.1 parse_http 314
33.3.2 parse_rfc 315
33.4 See Also 315
33.5 Copyright 315
33.6 Authors 315
APR::Error - Perl API for APR/Apache/mod_perl exceptions 316
34 APR::Error - Perl API for APR/Apache/mod_perl exceptions 316
34.1 Synopsis 317
34.2 Description 317
34.3 API 319
34.3.1 cluck 319
34.3.2 confess 319

34.3.3 strerror 319
34.4 See Also 320
34.5 Copyright 320
34.6 Authors 320
APR::Finfo - Perl API for APR fileinfo structure 321
35 APR::Finfo - Perl API for APR fileinfo structure 321
35.1 Synopsis 322
35.2 Description 322
35.3 API 322
35.3.1 atime 322
35.3.2 csize 323
35.3.3 ctime 323
35.3.4 device 324
35.3.5 filetype 324
35.3.6 fname 325
35.3.7 group 325
35.3.8 inode 325
35.3.9 mtime 326
35.3.10 name 326
35.3.11 nlink 326
35.3.12 protection 326
35.3.13 size 327
35.3.14 stat 327
35.3.15 user 328
35.3.16 valid 328
35.4 See Also 329
35.5 Copyright 329
35.6 Authors 329
APR::IpSubnet - Perl API for accessing APRs ip_subnet structures 330
36 APR::IpSubnet - Perl API for accessing APRs ip_subnet structures 330
36.1 Synopsis 331
36.2 Description 331
36.3 API 331
36.3.1 new 331
36.3.2 test 332
36.4 See Also 332
36.5 Copyright 332
36.6 Authors 332
APR::OS - Perl API for Platform-specific APR API 333
37 APR::OS - Perl API for Platform-specific APR API 333
37.1 Synopsis 334
37.2 Description 334
37.3 API 334
37.3.1 current_thread_id 334
37.4 See Also 334
37.5 Copyright 335
37.6 Authors 335

APR::PerlIO -- Perl IO layer for APR 336
38 APR::PerlIO -- Perl IO layer for APR 336
38.1 Synopsis 337
38.2 Description 337
38.3 Prerequisites 337
38.4 Constants 338
38.4.1 APR::PerlIO::PERLIO_LAYERS_ARE_ENABLED 338
38.5 API 338
38.5.1 open 338
38.5.2 seek 339
38.6 C API 339
38.7 See Also 339
38.8 Copyright 339
38.9 Authors 339
APR::Pool - Perl API for APR pools 340
39 APR::Pool - Perl API for APR pools 340
39.1 Synopsis 341
39.2 Description 341
39.3 API 342
39.3.1 cleanup_register 342
39.3.2 clear 343
39.3.3 DESTROY 343
39.3.4 destroy 344
39.3.5 is_ancestor 344
39.3.6 new 344
39.3.7 parent_get 345
39.3.8 tag 345
39.4 Unsupported API 346
39.4.1 cleanup_for_exec 346
39.5 See Also 346
39.6 Copyright 346
39.7 Authors 347
APR::SockAddr - Perl API for APR socket address structure 348
40 APR::SockAddr - Perl API for APR socket address structure 348
40.1 Synopsis 349
40.2 Description 349
40.3 API 349
40.3.1 ip_get 349
40.3.2 port 350
40.4 Unsupported API 350
40.4.1 equal 350
40.5 See Also 351
40.6 Copyright 351
40.7 Authors 351
APR::Socket - Perl API for APR sockets 352
41 APR::Socket - Perl API for APR sockets 352
41.1 Synopsis 353

41.2 Description 354
41.3 API 354
41.3.1 fileno 354
41.3.2 opt_get 354
41.3.3 opt_set 355
41.3.4 poll 355
41.3.5 recv 356
41.3.6 send 359
41.3.7 timeout_get 359
41.3.8 timeout_set 360
41.4 Unsupported API 360
41.4.1 bind 361
41.4.2 close 361
41.4.3 connect 361
41.4.4 listen 362
41.4.5 recvfrom 362
41.4.6 sendto 363
41.5 See Also 363
41.6 Copyright 363
41.7 Authors 364
APR::Status - Perl Interface to the APR_STATUS_IS_* macros 365
42 APR::Status - Perl Interface to the APR_STATUS_IS_* macros 365
42.1 Synopsis 366
42.2 Description 366
42.3 Functions 366
42.3.1 is_EACCES 366
42.3.2 is_EAGAIN 367
42.3.3 is_ENOENT 368
42.3.4 is_EOF 368
42.3.5 is_ECONNABORTED 368
42.3.6 is_ECONNRESET 369
42.3.7 is_TIMEUP 369
42.4 See Also 370
42.5 Copyright 370
42.6 Authors 370
APR::String - Perl API for manipulating APR UUIDs 371
43 APR::String - Perl API for manipulating APR UUIDs 371
43.1 Synopsis 372
43.2 Description 372
43.3 API 372
43.3.1 format_size 372
43.4 See Also 372
43.5 Copyright 372
43.6 Authors 373
APR::Table - Perl API for manipulating APR opaque string-content tables 374
44 APR::Table - Perl API for manipulating APR opaque string-content tables 374
44.1 Synopsis 375

44.2 Description 375
44.3 API 376
44.3.1 add 376
44.3.2 clear 376
44.3.3 compress 377
44.3.4 copy 378
44.3.5 do 379
44.3.6 get 380
44.3.7 make 380
44.3.8 merge 381
44.3.9 overlap 382
44.3.10 overlay 385
44.3.11 set 386
44.3.12 unset 386
44.4 TIE Interface 387
44.4.1 EXISTS 387
44.4.2 CLEAR 387
44.4.3 STORE 388
44.4.4 DELETE 388
44.4.5 FETCH 388
44.5 See Also 389
44.6 Copyright 389
44.7 Authors 389
APR::ThreadMutex - Perl API for APR thread mutexes 390
45 APR::ThreadMutex - Perl API for APR thread mutexes 390
45.1 Synopsis 391
45.2 Description 391
45.3 API 391
45.4 Unsupported API 391
45.4.1 DESTROY 391
45.4.2 lock 391
45.4.3 new 392
45.4.4 pool_get 392
45.4.5 trylock 392
45.4.6 unlock 393
45.5 See Also 393
45.6 Copyright 393
45.7 Authors 393
APR::ThreadRWLock - Perl API for APR thread read/write locks 394
46 APR::ThreadRWLock - Perl API for APR thread read/write locks 394
46.1 Synopsis 395
46.2 Description 395
46.3 API 395
46.4 Unsupported API 395
46.4.1 DESTROY 395
46.4.2 rdlock 396
46.4.3 tryrdlock 396

46.4.4 wrlock 396
46.4.5 trywrlock 397
46.4.6 new 397
46.4.7 pool_get 397
46.4.8 unlock 398
46.5 See Also 398
46.6 Copyright 398
46.7 Authors 398
APR::URI - Perl API for URI manipulations 399
47 APR::URI - Perl API for URI manipulations 399
47.1 Synopsis 400
47.2 Description 400
47.3 API 401
47.3.1 fragment 401
47.3.2 hostinfo 402
47.3.3 hostname 402
47.3.4 password 402
47.3.5 parse 402
47.3.6 path 403
47.3.7 rpath 403
47.3.8 port 404
47.3.9 port_of_scheme 404
47.3.10 query 404
47.3.11 scheme 405
47.3.12 user 405
47.3.13 unparse 405
47.4 See Also 407
47.5 Copyright 407
47.6 Authors 407
APR::Util - Perl API for Various APR Utilities 408
48 APR::Util - Perl API for Various APR Utilities 408
48.1 Synopsis 409
48.2 Description 409
48.3 API 409
48.3.1 password_validate 409
48.4 Unsupported API 410
48.4.1 filepath_name_get 410
48.4.2 password_get 411
48.5 See Also 411
48.6 Copyright 411
48.7 Authors 411
APR::UUID - Perl API for manipulating APR UUIDs 412
49 APR::UUID - Perl API for manipulating APR UUIDs 412
49.1 Synopsis 413
49.2 Description 413
49.3 API 413
49.3.1 format 413

49.3.2 new 413
49.3.3 DESTROY 414
49.3.4 parse 414
49.4 See Also 414
49.5 Copyright 414
49.6 Authors 414
ModPerl::Const -- ModPerl Constants 415
50 ModPerl::Const -- ModPerl Constants 415
50.1 Synopsis 416
50.2 Description 416
50.3 Constants 416
50.3.1 Other Constants 416
50.3.1.1 ModPerl::EXIT 416
50.4 See Also 416
50.5 Copyright 416
50.6 Authors 416
ModPerl::Global -- Perl API for manipulating special Perl lists 417
51 ModPerl::Global -- Perl API for manipulating special Perl lists 417
51.1 Synopsis 418
51.2 Description 418
51.3 API 418
51.3.1 special_list_call 418
51.3.2 special_list_clear 419
51.3.3 special_list_register 419
51.4 See Also 420
51.5 Copyright 420
51.6 Authors 420
ModPerl::MethodLookup -- Lookup mod_perl modules, objects and methods 421
52 ModPerl::MethodLookup -- Lookup mod_perl modules, objects and methods 421
52.1 Synopsis 422
52.2 Description 422
52.3 API 423
52.3.1 lookup_method() 423
52.3.2 lookup_module() 424
52.3.3 lookup_object() 424
52.3.4 preload_all_modules() 425
52.3.5 print_method() 425
52.3.6 print_module() 426
52.3.7 print_object() 426
52.4 Applications 426
52.4.1 AUTOLOAD 426
52.4.2 Command Line Lookups 428
52.5 Todo 428
52.6 See Also 429
52.7 Copyright 429
52.8 Authors 429

ModPerl::MM -- A "subclass" of ExtUtils::MakeMaker for mod_perl 2.0 430
53 ModPerl::MM -- A "subclass" of ExtUtils::MakeMaker for mod_perl 2.0 430
53.1 Synopsis 431
53.2 Description 431
53.3 `MY::` Default Methods 431
53.3.1 `ModPerl::MM::MY::post_initialize` 432
53.4 `WriteMakefile()` Default Arguments 432
53.4.1 `CCFLAGS` 433
53.4.2 `LIBS` 433
53.4.3 `INC` 433
53.4.4 `OPTIMIZE` 433
53.4.5 `LDDLFLAGS` 433
53.4.6 `TYPEMAPS` 433
53.4.7 `dynamic_lib` 433
53.4.7.1 `OTHERLDFLAGS` 433
53.4.8 `macro` 433
53.4.8.1 `MOD_INSTALL` 433
53.5 Public API 433
53.5.1 `WriteMakefile()` 433
53.5.2 `get_def_opt()` 433
ModPerl::PerlRun - Run unaltered CGI scripts under mod_perl 434
54 ModPerl::PerlRun - Run unaltered CGI scripts under mod_perl 434
54.1 Synopsis 435
54.2 Description 435
54.3 Special Blocks 435
54.3.1 `BEGIN` Blocks 435
54.3.2 `CHECK` and `INIT` Blocks 435
54.3.3 `END` Blocks 435
54.4 Authors 436
54.5 See Also 436
ModPerl::PerlRunPrefork - Run unaltered CGI scripts under mod_perl 437
55 ModPerl::PerlRunPrefork - Run unaltered CGI scripts under mod_perl 437
55.1 Synopsis 438
55.2 Description 438
55.3 Copyright 438
55.4 Authors 438
55.5 See Also 438
ModPerl::Registry - Run unaltered CGI scripts persistently under mod_perl 439
56 ModPerl::Registry - Run unaltered CGI scripts persistently under mod_perl 439
56.1 Synopsis 440
56.2 Description 440
56.3 DirectoryIndex 441
56.4 Special Blocks 441
56.4.1 `BEGIN` Blocks 441
56.4.2 `CHECK` and `INIT` Blocks 441
56.4.3 `END` Blocks 441
56.5 Security 441

56.6 Environment 442
56.7 Commandline Switches In First Line 442
56.8 Debugging 442
56.9 Caveats 442
56.10 Authors 442
56.11 See Also 443
ModPerl::RegistryBB - Run unaltered CGI scripts persistently under mod_perl 444
57 ModPerl::RegistryBB - Run unaltered CGI scripts persistently under mod_perl 444
57.1 Synopsis 445
57.2 Description 445
57.3 Authors 445
57.4 See Also 445
ModPerl::RegistryCooker - Cook mod_perl 2.0 Registry Modules 446
58 ModPerl::RegistryCooker - Cook mod_perl 2.0 Registry Modules 446
58.1 Synopsis 447
58.2 Description 447
58.2.1 Special Predefined Functions 449
58.3 Sub-classing Techniques 450
58.4 Examples 450
58.5 Authors 451
58.6 See Also 451
ModPerl::RegistryLoader - Compile ModPerl::RegistryCooker scripts at server startup . . 452
59 ModPerl::RegistryLoader - Compile ModPerl::RegistryCooker scripts at server startup . . 452
59.1 Synopsis 453
59.2 Description 453
59.3 Methods 453
59.4 Implementation Notes 456
59.5 Authors 456
59.6 SEE ALSO 456
ModPerl::RegistryPrefork - Run unaltered CGI scripts under mod_perl 457
60 ModPerl::RegistryPrefork - Run unaltered CGI scripts under mod_perl 457
60.1 Synopsis 458
60.2 Description 458
60.3 Copyright 458
60.4 Authors 458
60.5 See Also 458
ModPerl::Util - Helper mod_perl Functions 459
61 ModPerl::Util - Helper mod_perl Functions 459
61.1 Synopsis 460
61.2 Description 460
61.3 API 460
61.3.1 `current_callback` 460
61.3.2 `current_perl_id` 460
61.3.3 `exit` 461
61.3.4 `unload_package` 462
61.3.5 `untaint` 462
61.4 See Also 463

61.5 Copyright 463
61.6 Authors 463
Apache2::compat -- 1.0 backward compatibility functions deprecated in 2.0 464
62 Apache2::compat -- 1.0 backward compatibility functions deprecated in 2.0 464
62.1 Synopsis 465
62.2 Description 465
62.3 Compatibility Functions Colliding with mod_perl 2.0 API 465
62.3.1 Available Overridable Functions 466
62.4 Use in CPAN Modules 466
62.5 API 467
62.6 See Also 467
62.7 Copyright 467
62.8 Authors 467
Apache2::porting -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting 468
63 Apache2::porting -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting 468
63.1 Synopsis 469
63.2 Description 469
63.3 Culprits 469
63.4 See Also 470
63.5 Copyright 470
63.6 Authors 470
Apache2::Reload - Reload Perl Modules when Changed on Disk 471
64 Apache2::Reload - Reload Perl Modules when Changed on Disk 471
64.1 Synopsis 472
64.2 Description 472
64.2.1 Monitor All Modules in `%INC` 473
64.2.2 Register Modules Implicitly 473
64.2.3 Register Modules Explicitly 473
64.2.4 Monitor Only Certain Sub Directories 474
64.2.5 Special "Touch" File 474
64.2.6 Unregistering a module 474
64.3 Performance Issues 475
64.4 Debug 475
64.5 Caveats 475
64.5.1 Problems With Reloading Modules Which Do Not Declare Their Package Name . 475
64.5.2 Failing to Find a File to Reload 475
64.5.3 Problems with Scripts Running with Registry Handlers that Cache the Code . . . 476
64.5.3.1 The Problem 476
64.5.3.2 The Explanation 476
64.5.3.3 The Solution 477
64.6 Threaded MPM and Multiple Perl Interpreters 477
64.7 Pseudo-hashes 478
64.8 Copyright 478
64.9 Authors 478
Apache2::Resource - Limit resources used by httpd children 479
65 Apache2::Resource - Limit resources used by httpd children 479
65.1 Synopsis 480

65.2 Description 480
65.3 Defaults 480
65.4 See Also 480
65.5 Copyright 481
65.6 Author 481
Apache2::Status - Embedded interpreter status information 482
66 Apache2::Status - Embedded interpreter status information 482
66.1 Synopsis 483
66.2 Description 483
66.3 Options 484
66.3.1 `StatusOptionsAll` 484
66.3.2 `StatusDumper` 485
66.3.3 `StatusPeek` 485
66.3.4 `StatusLexInfo` 485
66.3.5 `StatusDeparse` 485
66.3.6 `StatusTerse` 485
66.3.7 `StatusTerseSize` 485
66.3.8 `StatusTerseSizeMainSummary` 486
66.3.9 `StatusGraph` 486
66.3.10 `Dot` 486
66.3.11 `GraphDir` 486
66.4 Prerequisites 486
66.5 Copyright 486
66.6 See Also 487
66.7 Authors 487
Apache2::SizeLimit - Because size does matter. 488
67 Apache2::SizeLimit - Because size does matter. 488
67.1 Synopsis 489
67.2 Description 489
67.3 Shared Memory Options 490
67.4 Caveats 490
67.4.1 Supported OSes 490
67.4.2 Supported MPMs 493
67.5 Copyright 493
67.6 Author 493
ModPerl::BuildMM -- A "subclass" of ModPerl::MM used for building mod_perl 2.0 . . 494
68 ModPerl::BuildMM -- A "subclass" of ModPerl::MM used for building mod_perl 2.0 . . 494
68.1 SYNOPSIS 495
68.2 DESCRIPTION 495
68.3 OVERRIDEN METHODS 495
68.3.1 `ModPerl::BuildMM::MY::constants` 495
68.3.2 `ModPerl::BuildMM::MY::top_targets` 495
68.3.3 `ModPerl::BuildMM::MY::postamble` 495
68.3.4 `ModPerl::BuildMM::MY::post_initialize` 495
68.3.5 `ModPerl::BuildMM::MY::libscan` 495