

1 A Reference to mod_perl 1.0 to mod_perl 2.0 Migration.

1.1 Description

This chapter is a reference for porting code and configuration files from mod_perl 1.0 to mod_perl 2.0.

To learn about the porting process you should first read about porting Perl modules (and may be about porting XS modules).

As will be explained in details later loading `Apache::compat` at the server startup, should make the code running properly under 1.0 work under mod_perl 2.0. If you want to port your code to mod_perl 2.0 or writing from scratch and not concerned about backwards compatibility, this document explains what has changed compared to mod_perl 1.0.

Several configuration directives were changed, renamed or removed. Several APIs have changed, renamed, removed, or moved to new packages. Certain functions while staying exactly the same as in mod_perl 1.0, now reside in different packages. Before using them you need to find out those packages and load them.

You should be able to find the destiny of the functions that you cannot find any more or which behave differently now under the package names the functions belong in mod_perl 1.0.

1.2 Configuration Files Porting

To migrate the configuration files to the mod_perl 2.0 syntax, you may need to do certain adjustments. Several configuration directives are deprecated in 2.0, but still available for backwards compatibility with mod_perl 1.0 unless 2.0 was built with `MP_COMPAT_1X=0`. If you don't need the backwards compatibility consider using the directives that have replaced them.

1.2.1 *PerlHandler*

`PerlHandler` was replaced with `PerlResponseHandler`.

1.2.2 *PerlSendHeader*

`PerlSendHeader` was replaced with `PerlOptions +/ParseHeaders` directive.

```
PerlSendHeader On  => PerlOptions +ParseHeaders
PerlSendHeader Off => PerlOptions -ParseHeaders
```

1.2.3 *PerlSetupEnv*

`PerlSetupEnv` was replaced with `PerlOptions +/SetupEnv` directive.

```
PerlSetupEnv On  => PerlOptions +SetupEnv
PerlSetupEnv Off => PerlOptions -SetupEnv
```

1.2.4 *PerlTaintCheck*

The taint mode now can be turned on with:

```
PerlSwitches -T
```

As with standard Perl, by default the taint mode is disabled and once enabled cannot be turned off inside the code.

1.2.5 *PerlWarn*

Warnings now can be enabled globally with:

```
PerlSwitches -w
```

1.2.6 *PerlFreshRestart*

PerlFreshRestart is a mod_perl 1.0 legacy and doesn't exist in mod_perl 2.0. A full teardown and startup of interpreters is done on restart.

If you need to use the same *httpd.conf* for 1.0 and 2.0, use:

```
<IfDefine !MODPERL2>  
    PerlFreshRestart  
</IfDefine>
```

1.2.7 *Apache Configuration Customization*

mod_perl 2.0 has slightly changed the mechanism for adding custom configuration directives and now also makes it easy to access an Apache parsed configuration tree's values.

META: add to the config tree access when it'll be written.

1.2.8 *@INC Manipulation*

- **Directories Added Automatically to @INC**

Only if mod_perl was built with MP_COMPAT_1X=1, two directories: *\$ServerRoot* and *\$ServerRoot/lib/perl* are pushed onto @INC. *\$ServerRoot* is as defined by the *ServerRoot* directive in *httpd.conf*.

- **PERL5LIB and PERLLIB Environment Variables**

mod_perl 2.0 doesn't do anything special about PERL5LIB and PERLLIB Environment Variables. If -T is in effect these variables are ignored by Perl. There are several other ways to adjust @INC.

1.3 Code Porting

mod_perl 2.0 is trying hard to be back compatible with mod_perl 1.0. However some things (mostly APIs) have been changed. In order to gain a complete compatibility with 1.0 while running under 2.0, you should load the compatibility module as early as possible:

```
use Apache::compat;
```

at the server startup. And unless there are forgotten things or bugs, your code should work without any changes under 2.0 series.

However, unless you want to keep the 1.0 compatibility, you should try to remove the compatibility layer and adjust your code to work under 2.0 without it. You want to do it mainly for the performance improvement.

This document explains what APIs have changed and what new APIs should be used instead.

If you have mod_perl 1.0 and 2.0 installed on the same system and the two use the same perl libraries directory (e.g. */usr/lib/perl5*), to use mod_perl 2.0 make sure to load first the Apache2 module which will perform the necessary adjustments to @INC.

```
use Apache2; # if you have 1.0 and 2.0 installed
use Apache::compat;
```

So if before loading *Apache2.pm* the @INC array consisted of:

```
/home/stas/perl/ithread/lib/5.8.0/i686-linux-thread-multi
/home/stas/perl/ithread/lib/5.8.0
/home/stas/perl/ithread/lib/site_perl/5.8.0/i686-linux-thread-multi
/home/stas/perl/ithread/lib/site_perl/5.8.0
/home/stas/perl/ithread/lib/site_perl
.
```

It will now look as:

```
/home/stas/perl/ithread/lib/site_perl/5.8.0/i686-linux-thread-multi/Apache2
/home/stas/perl/ithread/lib/5.8.0/i686-linux-thread-multi
/home/stas/perl/ithread/lib/5.8.0
/home/stas/perl/ithread/lib/site_perl/5.8.0/i686-linux-thread-multi
/home/stas/perl/ithread/lib/site_perl/5.8.0
/home/stas/perl/ithread/lib/site_perl
.
```

Notice that a new directory was prepended to the search path, so if for example the code attempts to load *Apache::RequestRec* and there are two versions of this module under */home/stas/perl/ithread/lib/site_perl/*:

```
5.8.0/i686-linux-thread-multi/Apache/RequestRec.pm
5.8.0/i686-linux-thread-multi/Apache2/Apache/RequestRec.pm
```

The mod_perl 2.0 version will be loaded first, because the directory *5.8.0/i686-linux-thread-multi/Apache2* is coming before the directory *5.8.0/i686-linux-thread-multi* in @INC.

Finally, mod_perl 2.0 has all its methods spread across many modules. In order to use these methods the modules containing them have to be loaded first. The module `ModPerl::MethodLookup` can be used to find out which modules need to be used. This module also provides a function `preload_all_modules()` that will load all mod_perl 2.0 modules, implementing their API in XS, which is useful when one starts to port their mod_perl 1.0 code, though preferably avoided in the production environment if you want to save memory.

1.4 Apache::Registry, Apache::PerlRun and Friends

`Apache::Registry`, `Apache::PerlRun` and other modules from the registry family now live in the `ModPerl::` namespace. In mod_perl 2.0 we put mod_perl specific functionality into the `ModPerl::` namespace, similar to `APR::` and `Apache::` which are used for apr and apache features, respectively.

At this moment `ModPerl::Registry` (and others) doesn't `chdir()` into the script's dir like `Apache::Registry` does, because `chdir()` affects the whole process under threads. This should be resolved by the time mod_perl 2.0 is released. Arthur Bergman works on the solution in form of: `ex::threads::cwd`. See: <http://www.perl.com/pub/a/2002/06/11/threads.html?page=2> Someone should pick up and complete this module to make it really useful.

Meanwhile if you are using a prefork MPM and you have to rely on mod_perl performing `chdir` to the script's directory, you can use the following subclass of `ModPerl::Registry`:

```
#file:ModPerl/RegistryPrefork.pm
#-----
package ModPerl::RegistryPrefork;

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

our $VERSION = '0.01';

use base qw(ModPerl::Registry);

use File::Basename ();

sub handler : method {
    my $class = (@_ >= 2) ? shift : __PACKAGE__;
    my $r = shift;
    return $class->new($r)->default_handler();
}

sub chdir_file {
    my $file = @_ == 2 ? $_[1] : $_[0]->{FILENAME};
    my $dir = File::Basename::dirname($file);
    chdir $dir or die "Can't chdir to $dir: $!";
}
```

1.5 Apache::Constants

```
}  
  
1;  
__END__
```

Adjust your *httpd.conf* to have:

```
Alias /perl /path/to/perl/scripts  
<Location /perl>  
    SetHandler perl-script  
    PerlResponseHandler ModPerl::RegistryPrefork  
    Options +ExecCGI  
    PerlOptions +ParseHeaders  
</Location>
```

Otherwise `ModPerl::Registry` modules are configured and used similarly to `Apache::Registry` modules. Refer to one of the following manpages for more information:

`ModPerl::RegistryCooker`, `ModPerl::Registry`, `ModPerl::RegistryBB` and `ModPerl::PerlRun`.

1.4.1 *ModPerl::RegistryLoader*

In `mod_perl 1.0` it was only possible to preload scripts as `Apache::Registry` handlers. In 2.0 the loader can use any of the registry classes to preload into. The old API works as before, but new options can be passed. See the `ModPerl::RegistryLoader` manpage for more information.

1.5 Apache::Constants

`Apache::Constants` has been replaced by three classes:

- **Apache::Const**

Apache constants

- **APR::Const**

Apache Portable Runtime constants

- **ModPerl::Const**

`mod_perl` specific constants

See the manpages of the respective modules to figure out which constants they provide.

META: add the info how to perform the transition. XXX: may be write a script, which can tell you how to port the constants to 2.0? Currently `Apache::compat` doesn't provide a complete back compatibility layer.

1.5.1 mod_perl 1.0 and 2.0 Constants Coexistence

If the same codebase is used for both mod_perl generations, the following technique can be used for using constants:

```
package MyApache::Foo;

use strict;
use warnings;

use mod_perl;
use constant MP2 => $mod_perl::VERSION >= 1.99;

BEGIN {
    if (MP2) {
        require Apache::Const;
        Apache::Const->import(-compile => qw(OK DECLINED));
    }
    else {
        require Apache::Constants;
        Apache::Constants->import(qw(OK DECLINED));
    }
}

sub handler {
    # ...
    return MP2 ? Apache::OK : Apache::Constants::OK;
}
1;
```

Notice that if you don't use the idiom:

```
return MP2 ? Apache::OK : Apache::Constants::OK;
```

but something like the following:

```
sub handler1 {
    ...
    return Apache::Constants::OK();
}
sub handler2 {
    ...
    return Apache::OK();
}
```

You need to add `()`. If you don't do that, let's say that you run under mod_perl 2.0, perl will complain about mod_perl 1.0 constant:

```
Bareword "Apache::Constants::OK" not allowed while "strict subs" ...
```

Adding `()` prevents this warning.

1.5.2 Deprecated Constants

REDIRECT and similar constants have been deprecated in Apache for years, in favor of the HTTP_* names (they no longer exist Apache 2.0). mod_perl 2.0 API performs the following aliasing behind the scenes:

```
NOT_FOUND      => 'HTTP_NOT_FOUND' ,
FORBIDDEN      => 'HTTP_FORBIDDEN' ,
AUTH_REQUIRED  => 'HTTP_UNAUTHORIZED' ,
SERVER_ERROR   => 'HTTP_INTERNAL_SERVER_ERROR' ,
REDIRECT       => 'HTTP_MOVED_TEMPORARILY' ,
```

but we suggest moving to use the HTTP_* names. For example if running in 1.0 compatibility mode change:

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

to:

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

This will work in both mod_perl generations.

1.5.3 SERVER_VERSION()

Apache::Constants::SERVER_VERSION() has been replaced with:

```
Apache::get_server_version();
```

1.5.4 export()

Apache::Constants::export() has no replacement in 2.0 as it's not needed.

1.6 Issues with Environment Variables

There are several thread-safety issues with setting environment variables.

Environment variables set during request time won't be seen by C code. See the DBD::Oracle issue for possible workarounds.

Forked processes (including backticks) won't see CGI emulation environment variables. (META: This will hopefully be resolved in the future, it's documented in modperl_env.c:modperl_env_magic_set_all.)

1.7 Special Environment Variables

1.7.1 ***\$ENV{GATEWAY_INTERFACE}***

The environment variable `$ENV{GATEWAY_INTERFACE}` is deprecated in mod_perl 2.0 (See: `MP_COMPAT_1X=0`). Instead use `$ENV{MOD_PERL}` (available in both mod_perl generations), which is set to something like this:

```
mod_perl/1.99_03-dev
```

However to check the version it's better to use `$mod_perl::VERSION`:

```
use mod_perl;
use constant MP2 => ($mod_perl::VERSION >= 1.99);
```

1.8 Apache::Methods

1.8.1 ***Apache->request***

`Apache->request` usage should be avoided under mod_perl 2.0 `$r` should be passed around as an argument instead (or in the worst case maintain your own global variable). Since your application may run under threaded mpm, the `Apache->request` usage involves storage and retrieval from the thread local storage, which is expensive.

It's possible to use `$r` even in CGI scripts running under Registry modules, without breaking the `mod_cgi` compatibility. Registry modules convert a script like:

```
print "Content-type: text/plain";
print "Hello";
```

into something like:

```
package Foo;
sub handler {
    print "Content-type: text/plain\n\n";
    print "Hello";
    return Apache::OK;
}
```

where the `handler()` function always receives `$r` as an argument, so if you change your script to be:

```
my $r;
$r = shift if $ENV{MOD_PERL};
if ($r) {
    $r->content_type('text/plain');
}
else {
    print "Content-type: text/plain\n\n";
}
print "Hello"
```

it'll really be converted into something like:

```
package Foo;
sub handler {
    my $r;
    $r = shift if $ENV{MOD_PERL};
    if ($r) {
        $r->content_type('text/plain');
    }
    else {
        print "Content-type: text/plain\n\n";
    }
    print "Hello"
    return Apache::OK;
}
```

The script works under both `mod_perl` and `mod_cgi`.

For example `CGI.pm 2.93` or higher accepts `$r` as an argument to its `new()` function. So does `CGI::Cookie::fetch` from the same distribution.

Moreover, user's configuration may preclude from `Apache->request` being available at run time. For any location that uses `Apache->request` and uses `SetHandler modperl`, the configuration should either explicitly enable this feature:

```
<Location ...>
    SetHandler modperl
    PerlOptions +GlobalRequest
    ...
</Location>
```

It's already enabled for `SetHandler perl-script`:

```
<Location ...>
    SetHandler perl-script
    ...
</Location>
```

This configuration makes `Apache->request` available **only** during the response phase (`PerlResponseHandler`). Other phases can make `Apache->request` available, by explicitly setting it in the handler that has an access to `$r`. For example the following skeleton for an *authen* phase handler makes the `Apache->request` available in the calls made from it:

```
package MyApache::Auth;

# PerlAuthenHandler MyApache::Auth

use Apache::RequestUtil ();
#...
sub handler {
    my $r = shift;
```

```
Apache->request($r);  
# do some calls that rely on Apache->request being available  
#...  
}
```

1.8.2 Apache->define

Apache->define has been replaced with `Apache::exists_config_define()` residing inside `Apache::ServerUtil`.

See the `Apache::ServerUtil` manpage.

1.8.3 Apache->can_stack_handlers

Apache->can_stack_handlers is no longer needed, as mod_perl 2.0 can always stack handlers.

1.8.4 Apache->untaint

Apache->untaint has moved to `Apache::ServerUtil` and now is a function, rather a class method. It'll will untaint all its arguments. You shouldn't be using this function unless you know what you are doing. Refer to the *perlsec* manpage for more information.

`Apache::compat` provides the backward compatible with mod_perl 1.0 implementation.

1.8.5 Apache->get_handlers

To get handlers for the server level, mod_perl 2.0 code should use:

```
$s->get_handlers(...);
```

or:

```
Apache->server->get_handlers(...);
```

Apache->get_handlers is available via `Apache::compat`.

1.8.6 Apache->push_handlers

To push handlers at the server level, mod_perl 2.0 code should use:

```
$s->push_handlers(...);
```

or:

```
Apache->server->push_handlers(...);
```

Apache->push_handlers is available via `Apache::compat`.

1.8.7 Apache->set_handlers

To set handlers at the server level, `mod_perl` 2.0 code should use:

```
$s->set_handlers(...);
```

or:

```
Apache->server->set_handlers(...);
```

Apache->set_handlers is available via `Apache::compat`.

1.8.8 Apache->httpd_conf

Apache->httpd_conf is now `$s->add_config` or `$r->add_config`. e.g.:

```
require Apache::ServerUtil;
Apache->server->httpd_conf(['require valid-user']);
```

See the `Apache::ServerUtil` manpage.

Apache->httpd_conf is available via `Apache::compat`.

1.8.9 Apache::exit()

`Apache::exit()` has been replaced with `ModPerl::Util::exit()`, which is a function (not a method) and accepts a single optional argument: status, whose default is 0 (== do nothing).

See the `ModPerl::Util` manpage.

1.8.10 Apache::gensym()

Since Perl 5.6.1 filehandlers are autovivified and there is no need for `Apache::gensym()` function, since now it can be done with:

```
open my $fh, "foo" or die $!;
```

Though the C function `modperl_perl_gensym()` is available for XS/C extensions writers.

1.8.11 Apache::module()

`Apache::module()` has been replaced with the function `Apache::Module::loaded()`, which now accepts a single argument: the module name.

1.8.12 Apache::log_error()

Apache::log_error() is not available in mod_perl 2.0 API. You can use:

```
Apache->server->log_error
```

instead. See the Apache::Log manpage.

1.9 Apache:: Variables

1.9.1 \$Apache::__T

\$Apache::__T is deprecated in mod_perl 2.0. Use \${^TAINT} instead.

1.10 Apache::Server:: Methods and Variables

1.10.1 \$Apache::Server::CWD

\$Apache::Server::CWD is deprecated and exists only in Apache::compat.

1.10.2 \$Apache::Server::AddPerlVersion

\$Apache::Server::AddPerlVersion is deprecated and exists only in Apache::compat.

1.11 Server Object Methods

1.11.1 \$s->register_cleanup

\$s->register_cleanup has been replaced with APR::Pool::cleanup_register() which accepts the pool object as the first argument instead of the server object. e.g.:

```
sub cleanup_callback { my $data = shift; ... }
$s->pool->cleanup_register(\&cleanup_callback, $data);
```

where the last argument \$data is optional, and if supplied will be passed as the first argument to the callback function.

See the APR::Pool manpage.

1.12 Request Object Methods

1.12.1 `$r->cgi_env`

See the next item

1.12.2 `$r->cgi_var`

`$r->cgi_env` and `$r->cgi_var` should be replaced with `$r->subprocess_env`, which works identically in both `mod_perl` generations.

1.12.3 `$r->current_callback`

`$r->current_callback` is now simply a `Apache::current_callback` and can be called for any of the phases, including those where `$r` simply doesn't exist.

`Apache::compat` implements `$r->current_callback` for backwards compatibility.

1.12.4 `$r->get_remote_host`

`get_remote_host()` is now invoked on the *connection object*:

```
use Apache::Connection;
$r->connection->get_remote_host();
```

`$r->get_remote_host` is available through `Apache::compat`.

1.12.5 `$r->cleanup_for_exec`

`$r->cleanup_for_exec` doesn't exist in the Apache 2.0 API, it is now being internally called by the Apache process spawning functions. For more information see `Apache::SubProcess` manpage.

There is `$pool->cleanup_for_exec`, but it's not the same as `$r->cleanup_for_exec` in the `mod_perl` 1.0 API.

1.12.6 `$r->content`

See the next item.

1.12.7 `$r->args` in an Array Context

`$r->args` in 2.0 returns the query string without parsing and splitting it into an array. You can also set the query string by passing a string to this method.

`$r->content` and `$r->args` in an array context were mistakes that never should have been part of the `mod_perl` 1.0 API. There are multiple reason for that, among others:

- does not handle multi-value keys
- does not handle multi-part content types
- does not handle chunked encoding
- slurps `$r->headers_in->{'content-length'}` into a single buffer (bad for performance, memory bloat, possible dos attack, etc.)
- in general duplicates functionality (and does so poorly) that is done better in `Apache::Request`.
- if one wishes to simply read POST data, there is the more modern `{setup,should,get}_client_block` API, and even more modern filter API, along with continued support for `read(STDIN, ...)` and `$r->read($buf, $r->headers_in->{'content-length'})`

For now you can use `CGI.pm` or the code in `Apache::compat` (it's slower).

META: when `Apache::Request` will be ported to mod_perl 2.0, you will have the fast C implementation of these functions.

1.12.8 `$r->chdir_file`

`chdir()` cannot be used in the threaded environment, therefore `$r->chdir_file` is not in the mod_perl 2.0 API.

For more information refer to: Threads Coding Issues Under mod_perl.

1.12.9 `$r->is_main`

`$r->is_main` is not part of the mod_perl 2.0 API. Use `!$r->main` instead.

Refer to the `Apache::RequestRec` manpage.

1.12.10 `$r->finfo`

Probably won't be implemented, because Apache 2.0's finfo datastructure can't be mapped into the Perl finfo datastructure.

`Apache::compat` handles that for now with:

```
sub finfo {
    my $r = shift;
    stat $r->filename;
    \*_;
```

1.12.11 `$r->notes`

Similar to `headers_in()`, `headers_out()` and `err_headers_out()` in `mod_perl 2.0`, `$r->notes()` returns an `APR::Table` object, which can be used as a tied hash or calling its `get()/set()/add()/unset()` methods.

If `Apache::compat` is loaded the old API:

```
$r->notes($key => $val);
$val = $r->notes($key);
```

is supported as well.

See the `Apache::RequestRec` manpage.

1.12.12 `$r->header_in`

See the next item.

1.12.13 `$r->header_out`

See the next item.

1.12.14 `$r->err_header_out`

`header_in()`, `header_out()` and `err_header_out()` are not available in `2.0`. Use `headers_in()`, `headers_out()` and `err_headers_out()` instead (which should be used in `1.0` as well). For example you need to replace:

```
$r->err_header_out("Pragma" => "no-cache");
```

with:

```
$r->err_headers_out->{'Pragma'} = "no-cache";
```

See the `Apache::RequestRec` manpage.

1.12.15 `$r->log_reason`

`$r->log_reason` is not available in `mod_perl 2.0` API. Use the other standard logging functions provided by the `Apache::Log` module. For example:

```
$r->log_error("it works!");
```

See the `Apache::Log` manpage.

1.12.16 *\$r->register_cleanup*

`$r->register_cleanup` has been replaced with `APR::Pool::cleanup_register()` which accepts the pool object as the first argument instead of the request object. e.g.:

```
sub cleanup_callback { my $data = shift; ... }
$r->pool->cleanup_register(\&cleanup_callback, $data);
```

where the last argument `$data` is optional, and if supplied will be passed as the first argument to the callback function.

See the *APR::Pool* manpage.

1.12.17 *\$r->post_connection*

`$r->post_connection` has been replaced with:

```
$r->connection->pool->cleanup_register();
```

See the *APR::Pool* manpage.

1.12.18 *\$r->request*

Use `Apache->request`.

1.12.19 *\$r->send_fd*

Apache 2.0 provides a new method `sendfile()` instead of `send_fd`, so if your code used to do:

```
open my $fh, "<$file" or die "$!";
$r->send_fd($fh);
close $fh;
```

now all you need is:

```
$r->sendfile($fh);
```

There is also a compatibility implementation in pure perl in `Apache::compat`.

1.12.20 *\$r->send_fd_length*

currently available only in the 1.0 compatibility layer. The problem is that Apache has changed the API and its functionality. See the implementation in `Apache::compat`.

XXX: needs a better resolution

1.12.21 \$r->send_http_header

This method is not needed in 2.0, though available in `Apache::compat`. 2.0 handlers only need to set the *Content-type* via `$r->content_type($type)`.

1.12.22 \$r->server_root_relative

`Apache::server_root_relative` is a function in 2.0 and its first argument is the *pool* object. For example:

```
# during request
my $conf_dir = Apache::server_root_relative($r->pool, 'conf');
# during startup
my $conf_dir = Apache::server_root_relative($s->pool, 'conf');
```

The old way:

```
my $conf_dir = Apache::server_root_relative('Apache', 'conf');
```

will work as well, but you shouldn't use it, because it'll internally use a global pool, which is a bit slower, but the worst drawback is that every time you use this way the allocated from this pool memory won't be cleaned up, till the server quits so there will be a memory leak. When you are inside a request and use `$r->pool`, this pool is freed at the end of each request.

See the `Apache::ServerUtil` manpage.

1.12.23 \$r->hard_timeout

See the next item.

1.12.24 \$r->reset_timeout

See the next item.

1.12.25 \$r->soft_timeout

See the next item.

1.12.26 \$r->kill_timeout

The functions `$r->hard_timeout`, `$r->reset_timeout`, `$r->soft_timeout` and `$r->kill_timeout` aren't needed in `mod_perl 2.0`.

1.12.27 *\$r->set_byterange*

See the next item.

1.12.28 *\$r->each_byterange*

The functions *\$r->set_byterange* and *\$r->each_byterange* aren't in the Apache 2.0 API, and therefore don't exist in mod_perl 2.0. The byterange serving functionality is now implemented in the *ap_byterange_filter*, which is a part of the core http module, meaning that it's automatically taking care of serving the requested ranges off the normal complete response. There is no need to configure it. It's executed only if the appropriate request headers are set. These headers aren't listed here, since there are several combinations of them, including the older ones which are still supported. For a complete info on these see *modules/http/http_protocol.c*.

1.13 Apache::Connection

1.13.1 *\$connection->auth_type*

The record *auth_type* doesn't exist in the Apache 2.0's connection struct. It exists only in the request record struct. The new accessor in 2.0 API is *\$r->ap_auth_type*.

Apache::compat provides a back compatibility method, though it relies on the availability of the global *Apache->request*, which requires the configuration to have:

```
PerlOptions +GlobalRequest
```

to set it up for earlier stages than response handler.

1.13.2 *\$connection->user*

This method is deprecated in mod_perl 1.0 and *\$r->user* should be used instead for both versions of mod_perl. *\$r->user()* method is available since mod_perl version 1.24_01.

1.13.3 *\$connection->local_addr*

See the next item.

1.13.4 *\$connection->remote_addr*

\$connection->local_addr and *\$connection->remote_addr* return an *APR::SocketAddr* object and you can use this object's methods to retrieve the wanted bits of information, so if you had a code like:

```
use Socket 'sockaddr_in';
my ($serverport, $serverip) = sockaddr_in($r->connection->local_addr);
my ($remoteport, $remoteip) = sockaddr_in($r->connection->remote_addr);
```

now it'll be written as:

```
require APR::SockAddr;
my $serverport = $c->local_addr->port_get;
my $serverip   = $c->local_addr->ip_get;
my $remoteport = $c->remote_addr->port_get;
my $remoteip   = $c->remote_addr->ip_get;
```

META: it's not simple to fix this in the compat layer, since it'll break the API for normal Apache 2.0 modules. Stay tuned for solutions.

1.14 Apache::File

The methods from mod_perl 1.0's module `Apache::File` have been either moved to other packages or removed.

1.14.1 *open() and close()*

The methods `open()` and `close()` were removed. See the back compatibility implementation in the module `Apache::compat`.

1.14.2 *tmpfile()*

The method `tmpfile()` was removed since Apache 2.0 doesn't have the API for this method anymore.

See `File::Temp`, or the back compatibility implementation in the module `Apache::compat`.

With Perl v5.8.0 you can create anonymous temporary files:

```
open $fh, "+>", undef or die $!;
```

That is a literal `undef`, not an undefined value.

1.15 Apache::Util

A few `Apache::Util` functions have changed their interface.

1.15.1 *Apache::Util::size_string()*

`Apache::Util::size_string()` has been replaced with `APR::String::format_size()`, which returns formatted strings of only 4 characters long. See the `APR::String` manpage.

1.15.2 Apache::Util::escape_uri()

Apache::Util::escape_uri() has been replaced with Apache::Util::escape_path() and requires a pool object as a second argument. For example:

```
$escaped_path = Apache::Util::escape_path($path, $r->pool);
```

1.15.3 Apache::Util::unescape_uri()

Apache::Util::unescape_uri() has been replaced with Apache::unescape_url().

1.15.4 Apache::Util::escape_html()

Apache::Util::escape_html currently is available only via Apache::compat until *ap_escape_html* is reworked to not require a pool.

1.15.5 Apache::Util::parsedate()

Apache::Util::parsedate() has been replaced with APR::Date::parse_http().

1.15.6 Apache::Util::ht_time()

Apache::Util::ht_time() has been replaced (temporary?) with Apache::Util::format_time(), which requires a pool object as a forth argument. All four arguments are now required.

For example:

```
use Apache::Util ();
$fmt = '%a, %d %b %Y %H:%M:%S %Z';
$gmt = 1;
$fmt_time = Apache::Util::format_time(time(), $fmt, $gmt, $r->pool);
```

See the Apache::Util manpage.

1.15.7 Apache::Util::validate_password()

Apache::Util::validate_password() has been replaced with APR::password_validate(). For example:

```
my $ok = Apache::Util::password_validate("stas", "ZeO.RAc3iYvpA");
```

1.16 Apache::URI

1.16.1 Apache::URI->parse(\$r, [\$uri])

`parse()` and its associate methods have moved into the `APR::URI` package. For example:

```
my $curl = $r->construct_url;  
APR::URI->parse($r->pool, $curl);
```

See the `APR::URI` manpage.

1.16.2 unparse()

Other than moving to the `APR::URI` package, `unparse` is now protocol-agnostic. Apache won't use `http` as the default protocol if `hostname` was set, but `scheme` wasn't not. So the following code:

```
# request http://localhost.localdomain:8529/TestAPI::uri  
my $parsed = $r->parsed_uri;  
$parsed->hostname($r->get_server_name);  
$parsed->port($r->get_server_port);  
print $parsed->unparse;
```

prints:

```
//localhost.localdomain:8529/TestAPI::uri
```

forcing you to make sure that the scheme is explicitly set. This will do the right thing:

```
# request http://localhost.localdomain:8529/TestAPI::uri  
my $parsed = $r->parsed_uri;  
$parsed->hostname($r->get_server_name);  
$parsed->port($r->get_server_port);  
$parsed->scheme('http');  
print $parsed->unparse;
```

prints:

```
http://localhost.localdomain:8529/TestAPI::uri
```

Notice that if `Apache::compat` is loaded, `unparse()` will transparently set `scheme` to `http` to preserve the backwards compatibility with `mod_perl` 1.0.

See the `APR::URI` manpage for more information.

1.17 Miscellaneous

1.17.1 Method Handlers

In mod_perl 1.0 the method handlers could be specified by using the (\$\$) prototype:

```
package Bird;
@ISA = qw(Eagle);

sub handler ( $$ ) {
    my( $class, $r ) = @_;
    ...;
}
```

mod_perl 2.0 doesn't handle callbacks with (\$\$) prototypes differently than other callbacks (as it did in mod_perl 1.0), mainly because several callbacks in 2.0 have more arguments than just \$r, so the (\$\$) prototype doesn't make sense anymore. Therefore if you want your code to work with both mod_perl generations and you can allow the luxury of:

```
require 5.6.0;
```

or if you need the code to run only on mod_perl 2.0, use the *method* subroutine attribute. (The subroutine attributes are supported in Perl since version 5.6.0.)

Here is the same example rewritten using the *method* subroutine attribute:

```
package Bird;
@ISA = qw(Eagle);

sub handler : method {
    my( $class, $r ) = @_;
    ...;
}
```

See the *attributes* manpage.

If `Class->method` syntax is used for a Perl*Handler, the `:method` attribute is not required.

The porting tutorial provides examples on how to use the same code base under both mod_perl generations when the handler has to be a method.

1.17.2 Stacked Handlers

Both mod_perl 1.0 and 2.0 support the ability to register more than one handler in each runtime phase, a feature known as stacked handlers. For example,

```
PerlAuthenHandler My::First My::Second
```

The behavior of stacked Perl handlers differs between mod_perl 1.0 and 2.0. In 2.0, mod_perl respects the run-type of the underlying hook - it does not run all configured Perl handlers for each phase but instead behaves in the same way as Apache does when multiple handlers are configured, respecting (or ignoring) the return value of each handler as it is called.

See Single Phase's Multiple Handlers Behavior for a complete description of each hook and its run-type.

1.18 Apache::src

For those who write 3rd party modules using XS, this module was used to supply mod_perl specific include paths, defines and other things, needed for building the extensions. mod_perl 2.0 makes things transparent with `ModPerl::MM`.

Here is how to write a simple *Makefile.PL* for modules wanting to build XS code against mod_perl 2.0:

```
use Apache2;
use mod_perl 1.99;
use ModPerl::MM ();

ModPerl::MM::WriteMakefile(
    NAME => "Foo",
);
```

and everything will be done for you.

META: we probably will have a compat layer at some point.

META: move this section to the devel/porting and link there instead

1.19 Apache::Table

`Apache::Table` has been renamed to `APR::Table`.

1.20 Apache::SIG

`Apache::SIG` currently exists only `Apache::compat` and it does nothing.

1.21 Apache::StatINC

`Apache::StatINC` has been replaced by `Apache::Reload`, which works for both mod_perl generations. To migrate to `Apache::Reload` simply replace:

```
PerlInitHandler Apache::StatINC
```

with:

```
PerlInitHandler Apache::Reload
```

However `Apache::Reload` provides an extra functionality, covered in the module's manpage.

1.22 Maintainers

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

- Stas Bekman <stas (at) stason.org>

1.23 Authors

- Stas Bekman <stas (at) stason.org>

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

Table of Contents:

1	A Reference to mod_perl 1.0 to mod_perl 2.0 Migration.	1
1.1	Description	2
1.2	Configuration Files Porting	2
1.2.1	PerlHandler	2
1.2.2	PerlSendHeader	2
1.2.3	PerlSetupEnv	2
1.2.4	PerlTaintCheck	3
1.2.5	PerlWarn	3
1.2.6	PerlFreshRestart	3
1.2.7	Apache Configuration Customization	3
1.2.8	@INC Manipulation	3
1.3	Code Porting	4
1.4	Apache::Registry, Apache::PerlRun and Friends	5
1.4.1	ModPerl::RegistryLoader	6
1.5	Apache::Constants	6
1.5.1	mod_perl 1.0 and 2.0 Constants Coexistence	7
1.5.2	Deprecated Constants	8
1.5.3	SERVER_VERSION()	8
1.5.4	export()	8
1.6	Issues with Environment Variables	8
1.7	Special Environment Variables	9
1.7.1	\$ENV{GATEWAY_INTERFACE}	9
1.8	Apache:: Methods	9
1.8.1	Apache->request	9
1.8.2	Apache->define	11
1.8.3	Apache->can_stack_handlers	11
1.8.4	Apache->untaint	11
1.8.5	Apache->get_handlers	11
1.8.6	Apache->push_handlers	11
1.8.7	Apache->set_handlers	12
1.8.8	Apache->httpd_conf	12
1.8.9	Apache::exit()	12
1.8.10	Apache::gensym()	12
1.8.11	Apache::module()	12
1.8.12	Apache::log_error()	13
1.9	Apache:: Variables	13
1.9.1	\$Apache::__T	13
1.10	Apache::Server:: Methods and Variables	13
1.10.1	\$Apache::Server::CWD	13
1.10.2	\$Apache::Server::AddPerlVersion	13
1.11	Server Object Methods	13
1.11.1	\$s->register_cleanup	13
1.12	Request Object Methods	13
1.12.1	\$r->cgi_env	14

Table of Contents:

1.12.2	\$r->cgi_var	14
1.12.3	\$r->current_callback	14
1.12.4	\$r->get_remote_host	14
1.12.5	\$r->cleanup_for_exec	14
1.12.6	\$r->content	14
1.12.7	\$r->args in an Array Context	14
1.12.8	\$r->chdir_file	15
1.12.9	\$r->is_main	15
1.12.10	\$r->finfo	15
1.12.11	\$r->notes	16
1.12.12	\$r->header_in	16
1.12.13	\$r->header_out	16
1.12.14	\$r->err_header_out	16
1.12.15	\$r->log_reason	16
1.12.16	\$r->register_cleanup	17
1.12.17	\$r->post_connection	17
1.12.18	\$r->request	17
1.12.19	\$r->send_fd	17
1.12.20	\$r->send_fd_length	17
1.12.21	\$r->send_http_header	18
1.12.22	\$r->server_root_relative	18
1.12.23	\$r->hard_timeout	18
1.12.24	\$r->reset_timeout	18
1.12.25	\$r->soft_timeout	18
1.12.26	\$r->kill_timeout	18
1.12.27	\$r->set_byterange	19
1.12.28	\$r->each_byterange	19
1.13	Apache::Connection	19
1.13.1	\$connection->auth_type	19
1.13.2	\$connection->user	19
1.13.3	\$connection->local_addr	19
1.13.4	\$connection->remote_addr	19
1.14	Apache::File	20
1.14.1	open() and close()	20
1.14.2	tmpfile()	20
1.15	Apache::Util	20
1.15.1	Apache::Util::size_string()	20
1.15.2	Apache::Util::escape_uri()	21
1.15.3	Apache::Util::unescape_uri()	21
1.15.4	Apache::Util::escape_html()	21
1.15.5	Apache::Util::parsedate()	21
1.15.6	Apache::Util::ht_time()	21
1.15.7	Apache::Util::validate_password()	21
1.16	Apache::URI	22
1.16.1	Apache::URI->parse(\$r, [\$uri])	22
1.16.2	unparse()	22
1.17	Miscellaneous	22

1.17.1 Method Handlers	23
1.17.2 Stacked Handlers	23
1.18 Apache::src	24
1.19 Apache::Table	24
1.20 Apache::SIG	24
1.21 Apache::StatINC	24
1.22 Maintainers	25
1.23 Authors	25