

# **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 it will be explained in details later, loading `Apache2::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 *PerlScript*

`PerlScript` was replaced with `PerlRequire`. `PerlRequire` is available in `mod_perl 1.0`, since 1997.

### 1.2.3 *PerlSendHeader*

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

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

## 1.2.4 *PerlSetupEnv*

PerlSetupEnv was replaced with PerlOptions +/-SetupEnv directive.

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

## 1.2.5 *PerlTaintCheck*

The taint mode now can be turned on with PerlSwitches:

```
PerlSwitches -T
```

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

## 1.2.6 *PerlWarn*

Warnings now can be enabled globally with PerlSwitches:

```
PerlSwitches -w
```

## 1.2.7 *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>
```

## 1.2.8 *\$Apache::Server::StrictPerlSections*

In mod\_perl 2.0, <Perl> sections errors are now always fatal. Any error in them will cause an immediate server startup abort, dumping the error to STDERR. To avoid this, `eval { }` can be used to trap errors and ignore them. In mod\_perl 1.0, `strict` was somewhat of a misnomer.

## 1.2.9 *\$Apache::Server::SaveConfig*

`$Apache::Server::SaveConfig` has been renamed to `$Apache2::PerlSections::Save`. see <Perl> sections for more information on this global variable.

### 1.2.10 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.

### 1.2.11 @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 Server Startup

mod\_perl 1.0 was always running its startup code as soon as it was encountered. In mod\_perl 2.0, it is not always the case. Refer to the mod\_perl 2.0 startup process section for details.

## 1.4 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 Apache2::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.

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 produc-

tion environment if you want to save memory.

## 1.5 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 Apache2:: which are used for libapr and Apache, respectively.

ModPerl::Registry (and others) doesn't chdir() into the script's dir like Apache::Registry does, because chdir() affects the whole process under threads. If you need this functionality use ModPerl::RegistryPrefork or ModPerl::PerlRunPrefork.

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.

### 1.5.1 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.

## 1.6 Apache::Constants

Apache::Constants has been replaced by three classes:

- **Apache2::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 Apache2::compat doesn't provide a complete back compatibility layer.

## 1.6.1 *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 MyApache2::Foo;

use strict;
use warnings;

use mod_perl;
use constant MP2 => ( exists $ENV{MOD_PERL_API_VERSION} and
                      $ENV{MOD_PERL_API_VERSION} >= 2 );

BEGIN {
    if (MP2) {
        require Apache2::Const;
        Apache2::Const->import(-compile => qw(OK DECLINED));
    }
    else {
        require Apache::Constants;
        Apache::Constants->import(qw(OK DECLINED));
    }
}

sub handler {
    # ...
    return MP2 ? Apache2::Const::OK : Apache::Constants::OK;
}
1;
```

Notice that if you don't use the idiom:

```
return MP2 ? Apache2::Const::OK : Apache::Constants::OK;
```

but something like the following:

```
sub handler1 {
    ...
    return Apache::Constants::OK();
}
sub handler2 {
    ...
    return Apache2::Const::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.

## 1.6.2 *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 mod\_perl 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.

## 1.6.3 *SERVER\_VERSION( )*

Apache::Constants::SERVER\_VERSION( ) has been replaced with Apache2::ServerUtil::get\_server\_version( ).

## 1.6.4 *export( )*

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

# 1.7 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.)

## 1.8 Special Environment Variables

### 1.8.1 `$ENV{GATEWAY_INTERFACE}`

The environment variable `$ENV{GATEWAY_INTERFACE}` is not special in `mod_perl 2.0`, but the same as any other CGI environment variables, i.e. it'll be enabled only if `PerlOptions +SetupEnv` is enabled and its value would be the default:

```
CGI/1.1
```

or anything else Apache decides to set it to, but not:

```
CGI-Perl/1.1
```

Instead use `$ENV{MOD_PERL}` (available in both `mod_perl` generations), which is set to the `mod_perl` version, like so:

```
mod_perl/2.000002
```

Therefore in order to check whether you are running under `mod_perl`, you'd say:

```
if ($ENV{MOD_PERL}) { ... }
```

To check for a specific version it's better to use `$ENV{MOD_PERL_API_VERSION}`

```
use mod_perl;
use constant MP2 => ( exists $ENV{MOD_PERL_API_VERSION} and
                     $ENV{MOD_PERL_API_VERSION} >= 2 );
```

## 1.9 Apache::Methods

### 1.9.1 `Apache->request`

`Apache->request` has been replaced with `Apache2::RequestUtil::request()`.

`Apache2::RequestUtil->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 `Apache2::RequestUtil->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 Apache2::Const::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 Apache2::Const::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 Apache2::RequestUtil->request being available at run time. For any location that uses Apache2::RequestUtil->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 `Apache2::RequestUtil->request` available **only** during the response phase (`PerlResponseHandler`). Other phases can make `Apache2::RequestUtil->request` available, by explicitly setting it in the handler that has an access to `$r`. For example the following skeleton for an *authn* phase handler makes the `Apache2::RequestUtil->request` available in the calls made from it:

```
package MyApache2::Auth;

# PerlAuthenHandler MyApache2::Auth

use Apache2::RequestUtil ();
#...
sub handler {
    my $r = shift;
    Apache2::RequestUtil->request($r);
    # do some calls that rely on Apache2::RequestUtil->request being available
    #...
}
```

## 1.9.2 Apache->define

`Apache->define` has been replaced with `Apache2::ServerUtil::exists_config_define()`.

## 1.9.3 Apache->can\_stack\_handlers

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

## 1.9.4 Apache->untaint

`Apache->untaint` has moved to `Apache2::ModPerl::Util::untaint()` 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.

`Apache2::compat` provides the backward compatible with `mod_perl 1.0` implementation.

## 1.9.5 Apache->get\_handlers

To get handlers for the server level, `mod_perl 2.0` code should use `Apache2::ServerUtil::get_handlers()`:

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

or:

```
Apache2::ServerUtil->server->get_handlers(...);
```

Apache->get\_handlers is available via Apache2::compat.

See also Apache2::RequestUtil::get\_handlers().

## ***1.9.6 Apache->push\_handlers***

To push handlers at the server level, mod\_perl 2.0 code should use Apache2::ServerUtil::push\_handlers():

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

or:

```
Apache2::ServerUtil->server->push_handlers(...);
```

Apache->push\_handlers is available via Apache2::compat.

See also Apache2::RequestUtil::push\_handlers().

## ***1.9.7 Apache->set\_handlers***

To set handlers at the server level, mod\_perl 2.0 code should use Apache2::ServerUtil::set\_handlers():

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

or:

```
Apache2::ServerUtil->server->set_handlers(...);
```

Apache->set\_handlers is available via Apache2::compat.

To reset the list of handlers, instead of doing:

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

do:

```
$r->set_handlers(PerlAuthenHandler => []);
```

or

```
$r->set_handlers(PerlAuthenHandler => undef);
```

See also `Apache2::RequestUtil::set_handlers()`.

## ***1.9.8 Apache->httpd\_conf***

Apache->httpd\_conf is now `$s->add_config`:

```
require Apache2::ServerUtil;
Apache2::ServerUtil->server->add_config(['require valid-user']);
```

Apache->httpd\_conf is available via `Apache2::compat`.

See also `Apache2::RequestUtil::add_config()`.

## ***1.9.9 Apache->unescape\_url\_info***

Apache->unescape\_url\_info is not available in mod\_perl 2.0 API. Use `CGI::Util::unescape` instead (<http://search.cpan.org/dist/CGI.pm/CGI/Util.pm>).

It is also available via `Apache2::compat` for backwards compatibility.

## ***1.9.10 Apache::exit()***

`Apache::exit()` has been replaced with `ModPerl::Util::exit()`.

## ***1.9.11 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.

## ***1.9.12 Apache::log\_error()***

`Apache::log_error()` is not available in mod\_perl 2.0 API. You can use `Apache2::Log::log_error()`:

```
Apache2::ServerUtil->server->log_error
```

instead. See the `Apache2::Log` manpage.

### ***1.9.13 Apache->warn***

`$Apache->warn` has been removed and exists only in `Apache2::compat`. Choose another `Apache2::Log` method.

### ***1.9.14 Apache::warn***

`Apache::warn` has been removed and exists only in `Apache2::compat`. Choose another `Apache2::Log` method.

### ***1.9.15 Apache::module()***

`Apache::module()` has been replaced with the function `Apache2::Module::loaded()`, which now accepts a single argument: the module name.

## **1.10 Apache:: Variables**

### ***1.10.1 \$Apache::\_\_T***

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

## **1.11 Apache::Module:: Methods**

### ***1.11.1 Apache::Module->top\_module***

`Apache::Module->top_module` has been replaced with the function `Apache2::Module::top_module()`.

### ***1.11.2 Apache::Module->get\_config***

`Apache::Module->get_config` has been replaced with the function `Apache2::Module::get_config()`.

## **1.12 Apache::ModuleConfig:: Methods**

### ***1.12.1 Apache::ModuleConfig->get***

`Apache::ModuleConfig->get` has been replaced with the function `Apache2::Module::get_config()`.

## 1.13 Apache::Server:: Methods and Variables

### 1.13.1 *\$Apache::Server::CWD*

`$Apache::Server::CWD` is deprecated and exists only in `Apache2::compat`.

### 1.13.2 *\$Apache::Server::AddPerlVersion*

`$Apache::Server::AddPerlVersion` is deprecated and exists only in `Apache2::compat`.

### 1.13.3 *\$Apache::Server::Starting and \$Apache::Server::ReStarting*

`$Apache::Server::Starting` and `$Apache::Server::ReStarting` were replaced by `Apache2::ServerUtil::restart_count()`. Though both exist in `Apache2::compat`.

### 1.13.4 *Apache::Server->warn*

`Apache::Server->warn` has been removed and exists only in `Apache2::compat`. Choose another `Apache2::Log` method.

## 1.14 Server Object Methods

### 1.14.1 *\$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, a callback function as a second and data variable as the optional third argument. If that data argument was provided it is then passed to the callback function when the time comes for the pool object to get destroyed.

```
use Apache2::ServerUtil ();
sub cleanup_callback {
    my $data = shift;
    # your code comes here
    return Apache2::Const::OK;
}
$s->pool->cleanup_register(\&cleanup_callback, $data);
```

See also `PerlChildExitHandler`.

In order to register a cleanup handler to be run only once when the main server (not each child process) shuts down, you can register a cleanup handler with `server_shutdown_cleanup_register()`.

### 1.14.2 *\$s->uid*

See the next entry.

### 1.14.3 *\$s->gid*

apache-1.3 had `server_rec` records for `server_uid` and `server_gid`. httpd-2.0 doesn't have them, because in httpd-2.0 the directives `User` and `Group` are platform specific. And only UNIX supports it: [http://httpd.apache.org/docs-2.0/mod/mpm\\_common.html#user](http://httpd.apache.org/docs-2.0/mod/mpm_common.html#user)

It's possible to emulate mod\_perl 1.0 API doing:

```
sub Apache2::Server::uid { $< }
sub Apache2::Server::gid { $( }
```

but the problem is that if the server is started as *root*, but its child processes are run under a different username, e.g. *nobody*, at the startup the above function will report the `uid` and `gid` values of *root* and not *nobody*, i.e. at startup it won't be possible to know what the `User` and `Group` settings are in *httpd.conf*.

META: though we can probably access the parsed config tree and try to fish these values from there. The real problem is that these values won't be available on all platforms and therefore we should probably not support them and let developers figure out how to code around it (e.g. by using `$<` and `$(`).

## 1.15 Request Object Methods

### 1.15.1 *\$r->print*

```
$r->print($foo);
```

or

```
print $foo;
```

no longer accepts a reference to a scalar as it did in mod\_perl 1.0. This optimisation is not needed in the mod\_perl 2.0's implementation of `print`.

### 1.15.2 *\$r->cgi\_env*

See the next item

### 1.15.3 *\$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.

## ***1.15.4 `$r->current_callback`***

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

`Apache2::compat` implements `$r->current_callback` for backwards compatibility.

## ***1.15.5 `$r->cleanup_for_exec`***

`$r->cleanup_for_exec` wasn't a part of the `mpl` core API, but lived in a 3rd party module `Apache2::SubProcess`. That module's functionality is now a part of `mod_perl 2.0` API. But `Apache 2.0` doesn't need this function any longer.

`Apache2::compat` implements `$r->cleanup_for_exec` for backwards compatibility as a NOOP.

See also the `Apache2::SubProcess` manpage.

## ***1.15.6 `$r->get_remote_host`***

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

```
use Apache2::Connection;
$r->connection->get_remote_host();
```

`$r->get_remote_host` is available through `Apache2::compat`.

## ***1.15.7 `$r->content`***

See the next item.

## ***1.15.8 `$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 `Apache2::Request`.
- if one wishes to simply read POST data, there is the more modern filter API, along with continued support for `read(STDIN, ...)` and `$r->read($buf, $r->headers_in->{'content-length'})`

You could use `CGI.pm` or the code in `Apache2::compat` (it's slower).

However, now that `Apache2::Request` has been ported to mod\_perl 2.0 you can use it instead and reap the benefits of the fast C implementations of these functions. For documentation on its uses, please see:

<http://httpd.apache.org/apreq>

### ***1.15.9 \$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`.

### ***1.15.10 \$r->is\_main***

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

Refer to the `Apache2::RequestRec` manpage.

### ***1.15.11 \$r->filename***

When a new `$r->filename` is assigned Apache 2.0 doesn't update the `finfo` structure like it did in Apache 1.3. If the old behavior is desired `Apache2::compat`'s overriding can be used. Otherwise one should explicitly update the `finfo` struct when desired as explained in the `filename` API entry.

### ***1.15.12 \$r->finfo***

As Apache 2.0 doesn't provide an access to the `stat` structure, but hides it in the opaque object `$r->finfo` now returns an `APR::Finfo` object. You can then invoke the `APR::Finfo` accessor methods on it.

It's also possible to adjust the mod\_perl 1.0 code using `Apache2::compat`'s overriding. For example:

```
use Apache2::compat;
Apache2::compat::override_mp2_api('Apache2::RequestRec::finfo');
my $is_writable = -w $r->finfo;
Apache2::compat::restore_mp2_api('Apache2::RequestRec::finfo');
```

which internally does just the following:

```
stat $r->filename and return \*_;
```

So maybe it's easier to just change the code to use this directly, so the above example can be adjusted to be:

```
my $is_writable = -w $r->filename;
```

with the performance penalty of an extra `stat()` system call. If you don't want this extra call, you'd have to write:

```
use APR::Finfo;
use Apache2::RequestRec;
use APR::Const -compile => qw(WWRITE);
my $is_writable = $r->finfo->protection & APR::WWRITE,
```

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

### **1.15.13 `$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.

It's also possible to adjust the `mod_perl 1.0` code using `Apache2::compat`'s overriding:

```
use Apache2::compat;
Apache2::compat::override_mp2_api('Apache2::RequestRec::notes');
$r->notes($key => $val);
$val = $r->notes($key);
Apache2::compat::restore_mp2_api('Apache2::RequestRec::notes');
```

See the `Apache2::RequestRec` manpage.

### **1.15.14 `$r->header_in`**

See `$r->err_header_out`.

### **1.15.15 `$r->header_out`**

See `$r->err_header_out`.

### **1.15.16 `$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 Apache2::RequestRec manpage.

### ***1.15.17 \$r->register\_cleanup***

Similarly to \$s->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.

### ***1.15.18 \$r->post\_connection***

\$r->post\_connection has been replaced with:

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

See the APR::Pool manpage.

### ***1.15.19 \$r->request***

Use Apache2::RequestUtil->request.

### ***1.15.20 \$r->send\_fd***

mod\_perl 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:

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

There is also a compatibility implementation of send\_fd in pure perl in Apache2::compat.

XXX: later we may provide a direct access to the real `send_fd`. That will be possible if we figure out how to portably convert `PerlIO/FILE` into `apr_file_t` (with help of `apr_os_file_put`, which expects a native file-handle, so I'm not sure whether this will work on win32).

### 1.15.21 *\$r->send\_http\_header*

This method is not needed in 2.0, though available in `Apache2::compat`. 2.0 handlers only need to set the *Content-type* via `$r->content_type($type)`.

### 1.15.22 *\$r->server\_root\_relative*

This method was replaced with `Apache2::ServerUtil::server_root_relative()` function and its first argument is a *pool* object. For example:

```
# during request
$conf_dir = Apache2::server_root_relative($r->pool, 'conf');
# during startup
$conf_dir = Apache2::server_root_relative($s->pool, 'conf');
```

Note that the old form

```
my $conf_dir = Apache->server_root_relative('conf');
```

is no longer valid - `server_root_relative()` must be explicitly passed a pool.

The old functionality is available with `Apache2::compat`.

### 1.15.23 *\$r->hard\_timeout*

See `$r->kill_timeout`.

### 1.15.24 *\$r->reset\_timeout*

See `$r->kill_timeout`.

### 1.15.25 *\$r->soft\_timeout*

See `$r->kill_timeout`.

### 1.15.26 *\$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`. `Apache2::compat` implements these functions for backwards compatibility as NOOPs.

### 1.15.27 *\$r->set\_byterange*

See *\$r->each\_byterange*.

### 1.15.28 *\$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*.

## 1.16 Apache::Connection

### 1.16.1 *\$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*.

*Apache2::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.

### 1.16.2 *\$connection->user*

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

### 1.16.3 *\$connection->local\_addr*

See *\$connection->remote\_addr*

### 1.16.4 *\$connection->remote\_addr*

*\$c->local\_addr* and *\$c->remote\_addr* return an *APR::SockAddr* 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 $c = $r->connection;
my ($serverport, $serverip) = sockaddr_in($c->local_addr);
my ($remoteport, $remoteip) = sockaddr_in($c->remote_addr);
```

now it'll be written as:

```
require APR::SockAddr;
my $c = $r->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;
```

It's also possible to adjust the code using Apache2::compat's overriding:

```
use Socket 'sockaddr_in';
use Apache2::compat;

Apache2::compat::override_mp2_api('Apache2::Connection::local_addr');
my ($serverport, $serverip) = sockaddr_in($r->connection->local_addr);
Apache2::compat::restore_mp2_api('Apache2::Connection::local_addr');

Apache2::compat::override_mp2_api('Apache::Connection::remote_addr');
my ($remoteport, $remoteip) = sockaddr_in($r->connection->remote_addr);
Apache2::compat::restore_mp2_api('Apache::Connection::remote_addr');
```

## 1.17 Apache::File

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

### 1.17.1 new(), open() and close()

The methods new(), open() and close() were removed. See the back compatibility implementation in the module Apache2::compat.

Because of that some of the idioms have changes too. If previously you were writing:

```
my $fh = Apache::File->new($r->filename)
    or return Apache::DECLINED;
# Slurp the file (hopefully it's not too big).
my $content = do { local $/; <$fh> };
close $fh;
```

Now, you would write that using Apache2::RequestUtil::slurp\_filename():

```
use Apache2::RequestUtil ();
my $content = ${ $r->slurp_filename() };
```

### 1.17.2 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 `Apache2::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.

## 1.18 Apache::Util

A few `Apache2::Util` functions have changed their interface.

### 1.18.1 *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.

### 1.18.2 *Apache::Util::escape\_uri()*

`Apache::Util::escape_uri()` has been replaced with `Apache2::Util::escape_path()` and requires a pool object as a second argument. For example:

```
$escaped_path = Apache2::Util::escape_path($path, $r->pool);
```

### 1.18.3 *Apache::Util::unescape\_uri()*

`Apache::Util::unescape_uri()` has been replaced with `Apache2::URI::unescape_url()`.

### 1.18.4 *Apache::Util::escape\_html()*

`Apache::Util::escape_html` is not available in `mod_perl 2.0`. Use `HTML::Entities` instead (<http://search.cpan.org/dist/HTML-Parser/lib/HTML/Entities.pm>).

It's also available via `Apache2::compat` for backwards compatibility.

### 1.18.5 *Apache::Util::parsedate()*

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

### 1.18.6 *Apache::Util::ht\_time()*

`Apache2::Util::ht_time()` now requires a pool object as a first argument.

For example:

```
use Apache2::Util ();
$fmt = '%a, %d %b %Y %H:%M:%S %Z';
$gmt = 1;
$fmt_time = Apache2::Util::ht_time($r->pool, time(), $fmt, $gmt);
```

See the Apache2::Util manpage.

It's also possible to adjust the mod\_perl 1.0 code using Apache2::compat's overriding.

For example:

```
use Apache2::compat;
Apache2::compat::override_mp2_api('Apache2::Util::ht_time');
$fmt_time = Apache2::Util::ht_time(time(), $fmt, $gmt);
Apache2::compat::restore_mp2_api('Apache2::Util::ht_time');
```

### ***1.18.7 Apache::Util::validate\_password()***

Apache::Util::validate\_password() has been replaced with APR::Util::password\_validate(). For example:

```
my $ok = Apache2::Util::password_validate("stas", "ZeO.RAc3iYvpA");
```

## **1.19 Apache::URI**

### ***1.19.1 Apache::URI->parse(\$r, [\$uri])***

parse() and its associated 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.

### ***1.19.2 unparse()***

Other than moving to the package APR::URI, 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
```

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

It's also possible to adjust the behavior to be mod\_perl 1.0 compatible using `Apache2::compat`'s overriding, in which case `unparse()` will transparently set *scheme* to *http*.

```
# request http://localhost.localdomain:8529/TestAPI::uri
Apache2::compat::override_mp2_api('APR::URI::unparse');
my $parsed = $r->parsed_uri;
# set hostname, but not the scheme
$parsed->hostname($r->get_server_name);
$parsed->port($r->get_server_port);
print $parsed->unparse;
Apache2::compat::restore_mp2_api('APR::URI::unparse');
```

prints:

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

## 1.20 Miscellaneous

### 1.20.1 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.

## 1.20.2 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 Stacked Handlers for a complete description of each hook and its run-type.

## 1.21 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 mod_perl 2.0;
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

## 1.22 Apache::Table

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

## 1.23 Apache::SIG

Apache::SIG currently exists only Apache2::compat and it does nothing.

## 1.24 Apache::StatINC

Apache::StatINC has been replaced by Apache2::Reload, which works for both mod\_perl generations. To migrate to Apache2::Reload simply replace:

```
PerlInitHandler Apache::StatINC
```

with:

```
PerlInitHandler Apache2::Reload
```

However Apache2::Reload provides an extra functionality, covered in the module's manpage.

## 1.25 Maintainers

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

- Stas Bekman [<http://stason.org/>]

## 1.26 Authors

- Stas Bekman [<http://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	PerlScript . . . . .	2
1.2.3	PerlSendHeader . . . . .	2
1.2.4	PerlSetupEnv . . . . .	3
1.2.5	PerlTaintCheck . . . . .	3
1.2.6	PerlWarn . . . . .	3
1.2.7	PerlFreshRestart . . . . .	3
1.2.8	\$Apache::Server::StrictPerlSections . . . . .	3
1.2.9	\$Apache::Server::SaveConfig . . . . .	3
1.2.10	Apache Configuration Customization . . . . .	4
1.2.11	@INC Manipulation . . . . .	4
1.3	Server Startup . . . . .	4
1.4	Code Porting . . . . .	4
1.5	Apache::Registry, Apache::PerlRun and Friends . . . . .	5
1.5.1	ModPerl::RegistryLoader . . . . .	5
1.6	Apache::Constants . . . . .	5
1.6.1	mod_perl 1.0 and 2.0 Constants Coexistence . . . . .	6
1.6.2	Deprecated Constants . . . . .	7
1.6.3	SERVER_VERSION() . . . . .	7
1.6.4	export() . . . . .	7
1.7	Issues with Environment Variables . . . . .	7
1.8	Special Environment Variables . . . . .	8
1.8.1	\$ENV{GATEWAY_INTERFACE} . . . . .	8
1.9	Apache::Methods . . . . .	8
1.9.1	Apache->request . . . . .	8
1.9.2	Apache->define . . . . .	10
1.9.3	Apache->can_stack_handlers . . . . .	10
1.9.4	Apache->untaint . . . . .	10
1.9.5	Apache->get_handlers . . . . .	10
1.9.6	Apache->push_handlers . . . . .	11
1.9.7	Apache->set_handlers . . . . .	11
1.9.8	Apache->httpd_conf . . . . .	12
1.9.9	Apache->unescape_url_info . . . . .	12
1.9.10	Apache::exit() . . . . .	12
1.9.11	Apache::gensym() . . . . .	12
1.9.12	Apache::log_error() . . . . .	12
1.9.13	Apache->warn . . . . .	13
1.9.14	Apache::warn . . . . .	13
1.9.15	Apache::module() . . . . .	13
1.10	Apache::Variables . . . . .	13
1.10.1	\$Apache::__T . . . . .	13

1.11	Apache::Module:: Methods . . . . .	13
1.11.1	Apache::Module->top_module . . . . .	13
1.11.2	Apache::Module->get_config . . . . .	13
1.12	Apache::ModuleConfig:: Methods . . . . .	13
1.12.1	Apache::ModuleConfig->get . . . . .	13
1.13	Apache::Server:: Methods and Variables . . . . .	14
1.13.1	\$Apache::Server::CWD . . . . .	14
1.13.2	\$Apache::Server::AddPerlVersion . . . . .	14
1.13.3	\$Apache::Server::Starting and \$Apache::Server::ReStarting . . . . .	14
1.13.4	Apache::Server->warn . . . . .	14
1.14	Server Object Methods . . . . .	14
1.14.1	\$s->register_cleanup . . . . .	14
1.14.2	\$s->uid . . . . .	15
1.14.3	\$s->gid . . . . .	15
1.15	Request Object Methods . . . . .	15
1.15.1	\$r->print . . . . .	15
1.15.2	\$r->cgi_env . . . . .	15
1.15.3	\$r->cgi_var . . . . .	15
1.15.4	\$r->current_callback . . . . .	16
1.15.5	\$r->cleanup_for_exec . . . . .	16
1.15.6	\$r->get_remote_host . . . . .	16
1.15.7	\$r->content . . . . .	16
1.15.8	\$r->args in an Array Context . . . . .	16
1.15.9	\$r->chdir_file . . . . .	17
1.15.10	\$r->is_main . . . . .	17
1.15.11	\$r->filename . . . . .	17
1.15.12	\$r->finfo . . . . .	17
1.15.13	\$r->notes . . . . .	18
1.15.14	\$r->header_in . . . . .	18
1.15.15	\$r->header_out . . . . .	18
1.15.16	\$r->err_header_out . . . . .	18
1.15.17	\$r->register_cleanup . . . . .	19
1.15.18	\$r->post_connection . . . . .	19
1.15.19	\$r->request . . . . .	19
1.15.20	\$r->send_fd . . . . .	19
1.15.21	\$r->send_http_header . . . . .	20
1.15.22	\$r->server_root_relative . . . . .	20
1.15.23	\$r->hard_timeout . . . . .	20
1.15.24	\$r->reset_timeout . . . . .	20
1.15.25	\$r->soft_timeout . . . . .	20
1.15.26	\$r->kill_timeout . . . . .	20
1.15.27	\$r->set_byterange . . . . .	21
1.15.28	\$r->each_byterange . . . . .	21
1.16	Apache::Connection . . . . .	21
1.16.1	\$connection->auth_type . . . . .	21
1.16.2	\$connection->user . . . . .	21
1.16.3	\$connection->local_addr . . . . .	21

1.16.4	\$connection->remote_addr	21
1.17	Apache::File	22
1.17.1	new(), open() and close()	22
1.17.2	tmpfile()	22
1.18	Apache::Util	23
1.18.1	Apache::Util::size_string()	23
1.18.2	Apache::Util::escape_uri()	23
1.18.3	Apache::Util::unescape_uri()	23
1.18.4	Apache::Util::escape_html()	23
1.18.5	Apache::Util::parsedate()	23
1.18.6	Apache::Util::ht_time()	23
1.18.7	Apache::Util::validate_password()	24
1.19	Apache::URI	24
1.19.1	Apache::URI->parse(\$r, [\$uri])	24
1.19.2	unparse()	24
1.20	Miscellaneous	25
1.20.1	Method Handlers	25
1.20.2	Stacked Handlers	26
1.21	Apache::src	26
1.22	Apache::Table	27
1.23	Apache::SIG	27
1.24	Apache::StatINC	27
1.25	Maintainers	27
1.26	Authors	27