



Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::PerlSections.3pm'

\$ man Apache2::PerlSections.3pm

libapache2-mod-perl2-2.0.12::dolibapache2-mod-perl2-2.0.12::docs::api::Apache2::PerlSections(3pm)

NAME

Apache2::PerlSections - write Apache configuration files in Perl

Synopsis

```
<Perl>
@PerlModule = qw(Mail::Send Devel::Peek);
#run the server as whoever starts it
$User = getpwuid(>) || >;
$Group = getgrgid() || );
$ServerAdmin = $User;
</Perl>
```

Description

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

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

Block sections such as "<Location>.."</Location>" are represented in a %Location hash,

e.g.:

```
<Perl>
$Location{"/~doug/"} = {
  AuthUserFile => '/tmp/htpasswd',
  AuthType     => 'Basic',
```

```

AuthName    => 'test',
DirectoryIndex => [qw(index.html index.htm)],
Limit      => {
    "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>
```

API

"Apache2::PerlSections" provides the following functions and/or methods:

"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

@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"

Configuration Variables

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

`$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 variables 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.

PerlSections Dumping

"`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.

"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

Advanced API

mod_perl 2.0 now introduces the same general concept of handlers to "<Perl>" sections.

Apache2::PerlSections simply being the default handler for them.

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

```
<Perl handler="My::PerlSection::Handler" somearg="test1">
```

```
$foo = 1;
```

```
$bar = 2;
```

```
</Perl>
```

And in My/PerlSection/Handler.pm:

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

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

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 routine 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.

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
```

Bugs

<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>
```

<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.

See Also

[mod_perl 2.0 documentation.](#)

Copyright

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

Authors

The mod_perl development team and numerous contributors.

perl v5.34.0

libapache2-mod-perl2-2.0.12::docs::api::Apache2::PerlSections(3pm)