1 mod_perl 2.0 Server Configuration

1.1 Description

This chapter provides an in-depth mod_perl 2.0 configuration details.

1.2 mod_perl configuration directives

Similar to mod_perl 1.0, in order to use mod_perl 2.0 a few configuration settings should be added to *httpd.conf*. They are quite similar to 1.0 settings but some directives were renamed and new directives were added.

1.3 Enabling mod_perl

To enable mod_perl built as DSO add to *httpd.conf*:

```
LoadModule perl_module modules/mod_perl.so
```

This setting specifies the location of the mod_perl module relative to the `ServerRoot` setting, therefore you should put it somewhere after `ServerRoot` is specified.

If mod_perl has been statically linked it's automatically enabled.

For Win32 specific details, see the documentation on Win32 configuration.

1.4 Accessing the mod_perl 2.0 Modules

In order to prevent from inadvertently loading mod_perl 1.0 modules mod_perl 2.0 Perl modules are installed into dedicated directories under *Apache2/*. The `Apache2` module prepends the locations of the mod_perl 2.0 libraries to `@INC`, which are the same as the core `@INC`, but with *Apache2/* appended. This module has to be loaded just after mod_perl has been enabled. This can be accomplished with:

```
use Apache2 ();
```

in the startup file. Only if you don't use a startup file you can add:

```
PerlModule Apache2
```

to *httpd.conf*, due to the order the `PerlRequire` and `PerlModule` directives are processed.

1.5 Startup File

Next usually a startup file with Perl code is loaded:

```
PerlRequire "/home/httpd/httpd-2.0/perl/startup.pl"
```

It’s used to adjust Perl modules search paths in `@INC`, pre-load commonly used modules, pre-compile constants, etc. Here is a typical *startup.pl* for mod_perl 2.0:

```
file:startup.pl
---------------
use Apache2 ();

use lib qw(/home/httpd/perl);

# enable if the mod_perl 1.0 compatibility is needed
# use Apache::compat ();

# preload all mp2 modules
# use ModPerl::MethodLookup;
# ModPerl::MethodLookup::preload_all_modules();

use ModPerl::Util (); #for CORE::GLOBAL::exit

use Apache::RequestRec ();
use Apache::RequestIO ();
use Apache::RequestUtil ();

use Apache::Server ();
use Apache::ServerUtil ();
use Apache::Connection ();
use Apache::Log ();

use APR::Table ();

use ModPerl::Registry ();

use Apache::Const -compile => ’:common’;
use APR::Const -compile => ’:common’;

1;
```

In this file the `Apache2` modules is loaded, so the 2.0 modules will be found. Afterwards `@INC` in adjusted to include non-standard directories with Perl modules:

```
use lib qw(/home/httpd/perl);
```

If you need to use the backwards compatibility layer load:

```
use Apache::compat ();
```

Next we preload the commonly used mod_perl 2.0 modules and precompile common constants.

Finally as usual the *startup.pl* file must be terminated with `1;`.

1.6 Server Configuration Directives

1.6.1 `PerlRequire`

```
META: to be written
```

1.6.2 `PerlModule`

```
META: to be written
```

1.6.3 `PerlLoadModule`

```
META: to be written
discused somewhere in docs::2.0::user::config::custom
```

1.6.4 `PerlSetVar`

```
META: to be written
```

1.6.5 `PerlAddVar`

```
META: to be written
```

1.6.6 `PerlSetEnv`

```
META: to be written
```

1.6.7 `PerlPassEnv`

```
META: to be written
```

1.6.8 `<Perl>` Sections

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

Please refer to the Apache::PerlSections manpage for more information.

META: a dedicated chapter with examples?

1.6.9 `PerlSwitches`

Now you can pass any Perl's command line switches in *httpd.conf* using the `PerlSwitches` directive. For example to enable warnings and Taint checking add:

```
PerlSwitches -wT
```

As an alternative to using `use lib` in *startup.pl* to adjust `@INC`, now you can use the command line switch `-I` to do that:

```
PerlSwitches -I/home/stas/modperl
```

You could also use `-Mlib=/home/stas/modperl` which is the exact equivalent as `use lib`, but it's broken on certain platforms/version (e.g. Darwin/5.6.0). `use lib` is removing duplicated entries, whereas `-I` does not.

1.6.10 SetHandler

mod_perl 2.0 provides two types of `SetHandler` handlers: `modperl` and `perl-script`. The `SetHandler` directive is only relevant for response phase handlers. It doesn't affect other phases.

1.6.10.1 modperl

Configured as:

```
SetHandler modperl
```

The bare mod_perl handler type, which just calls the `Perl*Handler`'s callback function. If you don't need the features provided by the *perl-script* handler, with the `modperl` handler, you can gain even more performance. (This handler isn't available in mod_perl 1.0.)

Unless the `Perl*Handler` callback, running under the `modperl` handler, is configured with:

```
PerlOptions +SetupEnv
```

or calls:

```
$r->subprocess_env;
```

in a void context (which has the same effect as `PerlOptions +SetupEnv` for the handler that called it), only the following environment variables are accessible via `%ENV`:

- `MOD_PERL` (always)
- `PATH` and `TZ` (if you had them defined in the shell or *httpd.conf*)

Therefore if you don't want to add the overhead of populating `%ENV`, when you simply want to pass some configuration variables from *httpd.conf*, consider using `PerlSetVar` and `PerlAddVar` instead of `PerlSetEnv` and `PerlPassEnv`. In your code you can retrieve the values using the `dir_config()` method. For example if you set in *httpd.conf*:

```
<Location /print_env2>
    SetHandler modperl
    PerlResponseHandler Apache::VarTest
    PerlSetVar VarTest VarTestValue
</Location>
```

this value can be retrieved inside `Apache::VarTest::handler()` with:

```
$r->dir_config('VarTest');
```

Alternatively use the Apache core directives `SetEnv` and `PassEnv`, which always populate `r->subprocess_env`, but this doesn't happen until the Apache *fixups* phase, which could be too late for your needs.

1.6.10.2 `perl-script`

Configured as:

```
SetHandler perl-script
```

Most mod_perl handlers use the *perl-script* handler. Among other things it does:

- `PerlOptions +GlobalRequest` is in effect only during the PerlResponseHandler phase unless:

  ```
  PerlOptions -GlobalRequest
  ```

  is specified.

- `PerlOptions +SetupEnv` is in effect unless:

  ```
  PerlOptions -SetupEnv
  ```

  is specified.

- `STDIN` and `STDOUT` get tied to the request object `$r`, which makes possible to read from `STDIN` and print directly to `STDOUT` via `CORE::print()`, instead of implicit calls like `$r->puts()`.

- Several special global Perl variables are saved before the handler is called and restored afterwards (similar to mod_perl 1.0). This includes: `%ENV`, `@INC`, `$/`, `STDOUT`'s `$|` and `END` blocks array (`PL_endav`).

1.6.10.3 Examples

Let's demonstrate the differences between the `modperl` and the `perl-script` core handlers in the following example, which represents a simple mod_perl response handler which prints out the environment variables as seen by it:

```
file:MyApache/PrintEnv1.pm
-----------------------
package MyApache::PrintEnv1;
use strict;

use Apache::RequestRec (); # for $r->content_type
use Apache::RequestIO ();  # for print
use Apache::Const -compile => ':common';

sub handler {
```

```
      my $r = shift;

      $r->content_type('text/plain');
      for (sort keys %ENV){
          print "$_ => $ENV{$_}\n";
      }

      return Apache::OK;
  }

  1;
```

This is the required configuration:

```
  PerlModule MyApache::PrintEnv1
  <Location /print_env1>
      SetHandler perl-script
      PerlResponseHandler MyApache::PrintEnv1
  </Location>
```

Now issue a request to *http://localhost/print_env1* and you should see all the environment variables printed out.

Here is the same response handler, adjusted to work with the `modperl` core handler:

```
  file:MyApache/PrintEnv2.pm
  ------------------------
  package MyApache::PrintEnv2;
  use strict;

  use Apache::RequestRec (); # for $r->content_type
  use Apache::RequestIO ();  # for $r->print

  use Apache::Const -compile => ':common';

  sub handler {
      my $r = shift;

      $r->content_type('text/plain');
      $r->subprocess_env;
      for (sort keys %ENV){
          $r->print("$_ => $ENV{$_}\n");
      }

      return Apache::OK;
  }

  1;
```

The configuration now will look as:

```
  PerlModule MyApache::PrintEnv2
  <Location /print_env2>
      SetHandler modperl
      PerlResponseHandler MyApache::PrintEnv2
  </Location>
```

`MyApache::PrintEnv2` cannot use `print()` and therefore uses `$r->print()` to generate a response. Under the `modperl` core handler `%ENV` is not populated by default, therefore `subprocess_env()` is called in a void context. Alternatively we could configure this section to do:

```
    PerlOptions +SetupEnv
```

If you issue a request to *http://localhost/print_env2*, you should see all the environment variables printed out as with *http://localhost/print_env1*.

1.6.11 *PerlOptions*

The directive `PerlOptions` provides fine-grained configuration for what were compile-time only options in the first mod_perl generation. It also provides control over what class of `PerlInterpreter` is used for a `<VirtualHost>` or location configured with `<Location>`, `<Directory>`, etc.

Options are enabled by prepending `+` and disabled with `-`. The options include:

1.6.11.1 Enable

On by default, can be used to disable mod_perl for a given `VirtualHost`. For example:

```
  <VirtualHost ...>
      PerlOptions -Enable
  </VirtualHost>
```

1.6.11.2 Clone

Share the parent Perl interpreter, but give the `VirtualHost` its own interpreter pool. For example should you wish to fine tune interpreter pools for a given virtual host:

```
  <VirtualHost ...>
      PerlOptions +Clone
      PerlInterpStart 2
      PerlInterpMax 2
  </VirtualHost>
```

This might be worthwhile in the case where certain hosts have their own sets of large-ish modules, used only in each host. By tuning each host to have its own pool, that host will continue to reuse the Perl allocations in their specific modules.

When cloning a Perl interpreter, to inherit base Perl interpreter's `PerlSwitches` use:

```
  <VirtualHost ...>
      ...
      PerlSwitches +inherit
  </VirtualHost>
```

1.6.11.3 `Parent`

Create a new parent Perl interpreter for the given `VirtualHost` and give it its own interpreter pool (implies the `Clone` option).

A common problem with mod_perl 1.0 was the shared namespace between all code within the process. Consider two developers using the same server and each wants to run a different version of a module with the same name. This example will create two *parent* Perl interpreters, one for each `<VirtualHost>`, each with its own namespace and pointing to a different paths in `@INC`:

META: is -Mlib portable? (problems with -Mlib on Darwin/5.6.0?)

```
<VirtualHost ...>
    ServerName dev1
    PerlOptions +Parent
    PerlSwitches -Mlib=/home/dev1/lib/perl
    PerlModule Apache2
</VirtualHost>

<VirtualHost ...>
    ServerName dev2
    PerlOptions +Parent
    PerlSwitches -Mlib=/home/dev2/lib/perl
    PerlModule Apache2
</VirtualHost>
```

Remember that `+Parent` gives you a completely new Perl interpreters pool, so all your modifications to `@INC` and preloading of the modules should be done again. Consider using PerlOptions +Clone if you want to inherit from the parent Perl interpreter.

Or even for a given location, for something like "dirty" cgi scripts:

```
<Location /cgi-bin>
    PerlOptions +Parent
    PerlInterpMaxRequests 1
    PerlInterpStart 1
    PerlInterpMax 1
    PerlResponseHandler ModPerl::Registry
</Location>
```

will use a fresh interpreter with its own namespace to handle each request.

1.6.11.4 `Perl*Handler`

Disable `Perl*Handler`s, all compiled-in handlers are enabled by default. The option name is derived from the `Perl*Handler` name, by stripping the `Perl` and `Handler` parts of the word. So `PerlLogHandler` becomes `Log` which can be used to disable `PerlLogHandler`:

```
PerlOptions -Log
```

Suppose one of the hosts does not want to allow users to configure `PerlAuthenHandler`, `PerlAuthzHandler`, `PerlAccessHandler` and <Perl> sections:

```
<VirtualHost ...>
    PerlOptions -Authen -Authz -Access -Sections
</VirtualHost>
```

Or maybe everything but the response handler:

```
<VirtualHost ...>
    PerlOptions None +Response
</VirtualHost>
```

1.6.11.5 `AutoLoad`

Resolve `Perl*Handlers` at startup time, which includes loading the modules from disk if not already loaded.

In mod_perl 1.0, configured `Perl*Handlers` which are not a fully qualified subroutine names are resolved at request time, loading the handler module from disk if needed. In mod_perl 2.0, configured `Perl*Handlers` are resolved at startup time. By default, modules are not auto-loaded during startup-time resolution. It is possible to enable this feature with:

```
PerlOptions +Autoload
```

Consider this configuration:

```
PerlResponseHandler Apache::Magick
```

In this case, `Apache::Magick` is the package name, and the subroutine name will default to *handler*. If the `Apache::Magick` module is not already loaded, `PerlOptions +Autoload` will attempt to pull it in at startup time. With this option enabled you don’t have to explicitly load the handler modules. For example you don’t need to add:

```
PerlModule Apache::Magick
```

in our example.

1.6.11.6 `GlobalRequest`

Setup the global `request_rec` for use with `Apache->request`.

This setting is enabled by default during the PerlResponseHandler phase for sections configured as:

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

And can be disabled with:

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

Notice that if you need the global request object during other phases, you will need to explicitly enable it in the configuration file.

You can also set that global object from the handler code, like so:

```
sub handler {
    my $r = shift;
    Apache->request($r);
    ...
}
```

The `+GlobalRequest` setting is needed for example if you use older versions of `CGI.pm` to process the incoming request. Starting from version 2.93, `CGI.pm` optionally accepts `$r` as an argument to `new()`, like so:

```
sub handler {
    my $r = shift;
    my $q = CGI->new($r);
    ...
}
```

Remember that inside registry scripts you can always get `$r` at the beginning of the script, since it gets wrapped inside a subroutine and accepts `$r` as the first and the only argument. For example:

```
#!/usr/bin/perl
use CGI;
my $r = shift;
my $q = CGI->new($r);
...
```

of course you won't be able to run this under mod_cgi, so you may need to do:

```
#!/usr/bin/perl
use CGI;
my $q = $ENV{MOD_PERL} ? CGI->new(shift @_) : CGI->new();
...
```

in order to have the script running under mod_perl and mod_cgi.

1.6.11.7 `ParseHeaders`

Scan output for HTTP headers, same functionality as mod_perl 1.0's `PerlSendHeader`, but more robust. This option is usually needs to be enabled for registry scripts which send the HTTP header with:

```
print "Content-type: text/html\n\n";
```

1.6.11.8 `MergeHandlers`

Turn on merging of `Perl*Handler` arrays. For example with a setting:

```
PerlFixupHandler Apache::FixupA

<Location /inside>
    PerlFixupHandler Apache::FixupB
</Location>
```

a request for */inside* only runs `Apache::FixupB` (mod_perl 1.0 behavior). But with this configuration:

```
PerlFixupHandler Apache::FixupA

<Location /inside>
    PerlOptions +MergeHandlers
    PerlFixupHandler Apache::FixupB
</Location>
```

a request for */inside* will run both `Apache::FixupA` and `Apache::FixupB` handlers.

1.6.11.9 `SetupEnv`

Set up environment variables for each request ala mod_cgi.

When this option is enabled, *mod_perl* fiddles with the environment to make it appear as if the code is called under the mod_cgi handler. For example, the `$ENV{QUERY_STRING}` environment variable is initialized with the contents of *Apache::args()*, and the value returned by *Apache::server_hostname()* is put into `$ENV{SERVER_NAME}`.

But `%ENV` population is expensive. Those who have moved to the Perl Apache API no longer need this extra `%ENV` population, and can gain by disabling it. A code using the `CGI.pm` module require `PerlOptions +SetupEnv` because that module relies on a properly populated CGI environment table.

This option is enabled by default for sections configured as:

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

Since this option adds an overhead to each request, if you don’t need this functionality you can turn it off for a certain section:

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

or globally:

```
PerlOptions -SetupEnv
<Location ...>
    ...
</Location>
```

and then it’ll affect the whole server. It can still be enabled for sections that need this functionality.

When this option is disabled you can still read environment variables set by you. For example when you use the following configuration:

```
PerlOptions -SetupEnv
PerlModule Modperl::Registry
<Location /perl>
  PerlSetEnv TEST hi
  SetHandler perl-script
  PerlResponseHandler ModPerl::Registry
  Options +ExecCGI
</Location>
```

and you issue a request for this script:

```
setupenvoff.pl
--------------
use Data::Dumper;
my $r = Apache->request();
$r->send_http_header('text/plain');
print Dumper(\%ENV);
```

you should see something like this:

```
$VAR1 = {
          'GATEWAY_INTERFACE' => 'CGI-Perl/1.1',
          'MOD_PERL' => 'mod_perl/2.0.1',
          'PATH' => 'bin:/usr/bin',
          'TEST' => 'hi'
        };
```

Notice that we have got the value of the environment variable *TEST*.

1.7 Server Life Cycle Handlers Directives

See Server life cycle.

1.7.1 `PerlOpenLogsHandler`

See `PerlOpenLogsHandler`.

1.7.2 `PerlPostConfigHandler`

See `PerlPostConfigHandler`.

1.7.3 `PerlChildInitHandler`

See `PerlChildInitHandler`.

1.7.4 `PerlChildExitHandler`

See `PerlChildExitHandler`.

1.8 Protocol Handlers Directives

See Protocol handlers.

1.8.1 `PerlPreConnectionHandler`

See `PerlPreConnectionHandler`.

1.8.2 `PerlProcessConnectionHandler`

See `PerlProcessConnectionHandler`.

1.9 Filter Handlers Directives

mod_perl filters are described in the filter handlers tutorial, `Apache::Filter` and `Apache::FilterRec` manpages.

The following filter handler configuration directives are available:

1.9.1 `PerlInputFilterHandler`

See `PerlInputFilterHandler`.

1.9.2 `PerlOutputFilterHandler`

See `PerlOutputFilterHandler`.

1.9.3 *`PerlSetInputFilter`*

See `PerlSetInputFilter`.

1.9.4 *`PerlSetOutputFilter`*

See `PerlSetInputFilter`.

1.10 HTTP Protocol Handlers Directives

See HTTP protocol handlers.

1.10.1 *`PerlPostReadRequestHandler`*

See `PerlPostReadRequestHandler`.

1.10.2 *`PerlTransHandler`*

See `PerlTransHandler`.

1.10.3 *`PerlMapToStorageHandler`*

See `PerlMapToStorageHandler`.

1.10.4 *`PerlInitHandler`*

See `PerlInitHandler`.

1.10.5 *`PerlHeaderParserHandler`*

See `PerlHeaderParserHandler`.

1.10.6 *`PerlAccessHandler`*

See `PerlAccessHandler`.

1.10.7 *`PerlAuthenHandler`*

See `PerlAuthenHandler`.

1.10.8 *`PerlAuthzHandler`*

See `PerlAuthzHandler`.

1.10.9 *`PerlTypeHandler`*

See `PerlTypeHandler`.

1.10.10 *`PerlFixupHandler`*

See `PerlFixupHandler`.

1.10.11 *`PerlResponseHandler`*

See `PerlResponseHandler`.

1.10.12 *`PerlLogHandler`*

See `PerlLogHandler`.

1.10.13 *`PerlCleanupHandler`*

See `PerlCleanupHandler`.

1.11 Threads Mode Specific Directives

These directives are enabled only in a threaded mod_perl+Apache combo:

1.11.1 *`PerlInterpStart`*

The number of interpreters to clone at startup time.

Default value: 3

1.11.2 *`PerlInterpMax`*

If all running interpreters are in use, mod_perl will clone new interpreters to handle the request, up until this number of interpreters is reached. when `PerlInterpMax` is reached, mod_perl will block (via COND_WAIT()) until one becomes available (signaled via COND_SIGNAL()).

Default value: 5

1.11.3 *`PerlInterpMinSpare`*

The minimum number of available interpreters this parameter will clone interpreters up to `PerlInterpMax`, before a request comes in.

Default value: 3

1.11.4 *`PerlInterpMaxSpare`*

mod_perl will throttle down the number of interpreters to this number as those in use become available.

Default value: 3

1.11.5 *`PerlInterpMaxRequests`*

The maximum number of requests an interpreter should serve, the interpreter is destroyed when the number is reached and replaced with a fresh clone.

Default value: 2000

1.11.6 *`PerlInterpScope`*

As mentioned, when a request in a threaded mpm is handled by mod_perl, an interpreter must be pulled from the interpreter pool. The interpreter is then only available to the thread that selected it, until it is released back into the interpreter pool. By default, an interpreter will be held for the lifetime of the request, equivalent to this configuration:

```
  PerlInterpScope request
```

For example, if a `PerlAccessHandler` is configured, an interpreter will be selected before it is run and not released until after the logging phase.

Interpreters will be shared across sub-requests by default, however, it is possible to configure the interpreter scope to be per-sub-request on a per-directory basis:

```
  PerlInterpScope subrequest
```

With this configuration, an autoindex generated page, for example, would select an interpreter for each item in the listing that is configured with a Perl*Handler.

It is also possible to configure the scope to be per-handler:

```
  PerlInterpScope handler
```

For example if `PerlAccessHandler` is configured, an interpreter will be selected before running the handler, and put back immediately afterwards, before Apache moves onto the next phase. If a `PerlFixupHandler` is configured further down the chain, another interpreter will be selected and again put back afterwards, before `PerlResponseHandler` is run.

For protocol handlers, the interpreter is held for the lifetime of the connection. However, a C protocol module might hook into mod_perl (e.g. mod_ftp) and provide a `request_rec` record. In this case, the default scope is that of the request. Should a mod_perl handler want to maintain state for the lifetime of an ftp connection, it is possible to do so on a per-virtualhost basis:

```
PerlInterpScope connection
```

Default value: `request`

1.12 Debug Directives

1.12.1 `PerlTrace`

The `PerlTrace` is used for tracing the mod_perl execution. This directive is enabled when mod_perl is compiled with the `MP_TRACE=1` option.

To enable tracing, add to *httpd.conf*:

```
PerlTrace [level]
```

where `level` is either:

```
all
```

which sets maximum logging and debugging levels;

a combination of one or more option letters from the following list:

```
a Apache API interaction
c configuration for directive handlers
d directive processing
f filters
e environment variables
g Perl runtime interaction
h handlers
i interpreter pool management
m memory allocations
o I/O
s Perl sections
t benchmark-ish timings
```

Tracing options add to the previous setting and don’t override it. So for example:

```
 PerlTrace c
 ...
 PerlTrace f
```

will set tracing level first to ’c’ and later to ’cf’. If you wish to override settings, unset any previous setting by assigning 0 (zero), like so:

```
  PerlTrace c
  ...
  PerlTrace 0
  PerlTrace f
```

now the tracing level is set only to ’f’. You can’t mix the number 0 with letters, it must be alone.

When `PerlTrace` is not specified, the tracing level will be set to the value of the `$ENV{MOD_PERL_TRACE}` environment variable.

1.13 mod_perl Directives Argument Types and Allowed Location

The following table shows where in the configuration files mod_perl configuration directives are allowed to appear, what kind and how many arguments they expect:

General directives:

Directive	Arguments	Scope
PerlSwitches	ITERATE	SRV
PerlRequire	ITERATE	SRV
PerlModule	ITERATE	SRV
PerlLoadModule	RAW_ARGS	SRV
PerlOptions	ITERATE	DIR
PerlSetVar	TAKE2	DIR
PerlAddVar	ITERATE2	DIR
PerlSetEnv	TAKE2	DIR
PerlPassEnv	TAKE1	SRV
<Perl> Sections	RAW_ARGS	SRV
PerlTrace	TAKE1	SRV

Handler assignment directives:

Directive	Arguments	Scope
PerlOpenLogsHandler	ITERATE	SRV
PerlPostConfigHandler	ITERATE	SRV
PerlChildInitHandler	ITERATE	SRV
PerlChildExitHandler	ITERATE	SRV
PerlPreConnectionHandler	ITERATE	SRV
PerlProcessConnectionHandler	ITERATE	SRV
PerlPostReadRequestHandler	ITERATE	SRV
PerlTransHandler	ITERATE	SRV
PerlMapToStorageHandler	ITERATE	SRV
PerlInitHandler	ITERATE	DIR
PerlHeaderParserHandler	ITERATE	DIR
PerlAccessHandler	ITERATE	DIR
PerlAuthenHandler	ITERATE	DIR
PerlAuthzHandler	ITERATE	DIR
PerlTypeHandler	ITERATE	DIR

```
  PerlFixupHandler             ITERATE    DIR
  PerlResponseHandler          ITERATE    DIR
  PerlLogHandler               ITERATE    DIR
  PerlCleanupHandler           ITERATE    DIR

  PerlInputFilterHandler       ITERATE    DIR
  PerlOutputFilterHandler      ITERATE    DIR
  PerlSetInputFilter           ITERATE    DIR
  PerlSetOutputFilter          ITERATE    DIR
```

Perl Interpreter management directives:

```
    Directive                 Arguments  Scope
  --------------------------------------------
  PerlInterpStart              TAKE1      SRV
  PerlInterpMax                TAKE1      SRV
  PerlInterpMinSpare           TAKE1      SRV
  PerlInterpMaxSpare           TAKE1      SRV
  PerlInterpMaxRequests        TAKE1      SRV
  PerlInterpScope              TAKE1      DIR
```

mod_perl 1.0 back-compatibility directives:

```
    Directive                 Arguments  Scope
  --------------------------------------------
  PerlHandler                   ITERATE   DIR
  PerlSendHeader                FLAG      DIR
  PerlSetupEnv                  FLAG      DIR
  PerlTaintCheck                FLAG      SRV
  PerlWarn                      FLAG      SRV
```

The *Arguments* column represents the type of arguments directives accepts, where:

- **ITERATE**

  Expects a list of arguments.

- **ITERATE2**

  Expects one argument, followed by at least one or more arguments.

- **TAKE1**

  Expects one argument only.

- **TAKE2**

  Expects two arguments only.

- **FLAG**

  One of `On` or `Off` (case insensitive).

- **RAW_ARGS**

  The function parses the command line by itself.

The *Scope* column shows the location the directives are allowed to appear in:

- **SRV**

  Global configuration and `<VirtualHost>` (mnemonic: *SeRVer*). These directives are defined as `RSRC_CONF` in the source code.

- **DIR**

  `<Directory>`, `<Location>`, `<Files>` and all their regular expression variants (mnemonic: *DIRectory*). These directives can also appear in *.htaccess* files. These directives are defined as `OR_ALL` in the source code.

  These directives can also appear in the global server configuration and `<VirtualHost>`.

Apache specifies other allowed location types which are currently not used by the core mod_perl directives and their definition can be found in *include/httpd_config.h* (hint: search for `RSRC_CONF`).

Also see Single Phase's Multiple Handlers Behavior.

1.14 Server Startup Options Retrieval

Inside *httpd.conf* one can do conditional configuration based on the define options passed at the server startup. For example:

```
<IfDefine PERLDB>
    <Perl>
        use Apache::DB ();
        Apache::DB->init;
    </Perl>

    <Location />
        PerlFixupHandler Apache::DB
    </Location>
</IfDefine>
```

So only when the server is started as:

```
% httpd C<-DPERLDB> ...
```

The configuration inside `IfDefine` will have an effect. If you want to have some configuration section to have an effect if a certain define wasn't defined use `!`, for example here is the opposite of the previous example:

```
<IfDefine !PERLDB>
    # ...
</IfDefine>
```

If you need to access any of the startup defines in the Perl code you use `Apache::exists_config_define`. For example in a startup file you can say:

```
use Apache::ServerUtil ();
if (Apache::exists_config_define("PERLDB")) {
    require Apache::DB;
    Apache::DB->init;
}
```

For example to check whether the server has been started in a single mode use:

```
if (Apache::exists_config_define("ONE_PROCESS")) {
    print "Running in a single mode";
}
```

1.14.1 `MODPERL2` Define Option

When running under mod_perl 2.0 a special configuration "define" symbol `MODPERL2` is enabled internally, as if the server had been started with `-DMODPERL2`. For example this can be used to write a configuration file which needs to do something different whether it's running under mod_perl 1.0 or 2.0:

```
<IfDefine MODPERL2>
    # 2.0 configuration
</IfDefine>
<IfDefine !MODPERL2>
    # else
</IfDefine>
```

From within Perl code this can be tested with `Apache::exists_config_define()`, for example:

```
if (Apache::exists_config_define("MODPERL2")) {
    # some 2.0 specific code
}
```

1.15 Perl Interface to the Apache Configuration Tree

For now refer to the Apache::Directive manpage and the test *t/response/TestApache/conftree.pm* in the mod_perl source distribution.

META: need help to write the tutorial section on this with examples.

1.16 Adjusting `@INC`

You can always adjust contents of `@INC` before the server starts. There are several ways to do that.

- ***startup.pl***

  In the startup file you can use the `lib` pragma like so:

  ```
  use lib qw(/home/httpd/project1/lib /tmp/lib);
  use lib qw(/home/httpd/project2/lib);
  ```

- ***httpd.conf***

  In *httpd.conf* you can use the `PerlSwitches` directive to pass arguments to perl as you do from the command line, e.g.:

  ```
  PerlSwitches -I/home/httpd/project1/lib -I/tmp/lib
  PerlSwitches -I/home/httpd/project2/lib
  ```

1.16.1 `PERL5LIB` and `PERLLIB` Environment Variables

The effect of the `PERL5LIB` and `PERLLIB` environment variables on `@INC` is described in the *perlrun* manpage. mod_perl 2.0 doesn't do anything special about them.

It's important to remind that both `PERL5LIB` and `PERLLIB` are ignored when the taint mode (`PerlSwitches -T`) is in effect. Since you want to make sure that your mod_perl server is running under the taint mode, you can't use the `PERL5LIB` and `PERLLIB` environment variables.

However there is the *perl5lib* module on CPAN, which, if loaded, bypasses perl's security and will affect `@INC`. Use it only if you know what you are doing.

1.16.2 Modifying `@INC` on a Per-VirtualHost

If Perl used with mod_perl was built with ithreads support one can specify different `@INC` values for different VirtualHosts, using a combination of `PerlOptions +Parent` and `PerlSwitches`. For example:

```
<VirtualHost ...>
    ServerName dev1
    PerlOptions +Parent
    PerlSwitches -I/home/dev1/lib/perl
    PerlModule Apache2
</VirtualHost>

<VirtualHost ...>
    ServerName dev2
    PerlOptions +Parent
    PerlSwitches -I/home/dev2/lib/perl
    PerlModule Apache2
</VirtualHost>
```

1.17 General Issues

1.18 Maintainers

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

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

1.19 Authors

- Doug MacEachern <dougm (at) covalent.net>
- Stas Bekman <stas (at) stason.org>

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

Table of Contents:

1 mod_perl 2.0 Server Configuration . . . 1
1.1 Description . . . 2
1.2 mod_perl configuration directives . . . 2
1.3 Enabling mod_perl . . . 2
1.4 Accessing the mod_perl 2.0 Modules . . . 2
1.5 Startup File . . . 2
1.6 Server Configuration Directives . . . 4
1.6.1 `PerlRequire` . . . 4
1.6.2 `PerlModule` . . . 4
1.6.3 `PerlLoadModule` . . . 4
1.6.4 `PerlSetVar` . . . 4
1.6.5 `PerlAddVar` . . . 4
1.6.6 `PerlSetEnv` . . . 4
1.6.7 `PerlPassEnv` . . . 4
1.6.8 `<Perl>` Sections . . . 4
1.6.9 `PerlSwitches` . . . 4
1.6.10 `SetHandler` . . . 5
1.6.10.1 `modperl` . . . 5
1.6.10.2 `perl-script` . . . 6
1.6.10.3 Examples . . . 6
1.6.11 `PerlOptions` . . . 8
1.6.11.1 `Enable` . . . 8
1.6.11.2 `Clone` . . . 8
1.6.11.3 `Parent` . . . 9
1.6.11.4 `Perl*Handler` . . . 9
1.6.11.5 `AutoLoad` . . . 10
1.6.11.6 `GlobalRequest` . . . 10
1.6.11.7 `ParseHeaders` . . . 11
1.6.11.8 `MergeHandlers` . . . 12
1.6.11.9 `SetupEnv` . . . 12
1.7 Server Life Cycle Handlers Directives . . . 13
1.7.1 `PerlOpenLogsHandler` . . . 13
1.7.2 `PerlPostConfigHandler` . . . 14
1.7.3 `PerlChildInitHandler` . . . 14
1.7.4 `PerlChildExitHandler` . . . 14
1.8 Protocol Handlers Directives . . . 14
1.8.1 `PerlPreConnectionHandler` . . . 14
1.8.2 `PerlProcessConnectionHandler` . . . 14
1.9 Filter Handlers Directives . . . 14
1.9.1 `PerlInputFilterHandler` . . . 14
1.9.2 `PerlOutputFilterHandler` . . . 14
1.9.3 `PerlSetInputFilter` . . . 15
1.9.4 `PerlSetOutputFilter` . . . 15
1.10 HTTP Protocol Handlers Directives . . . 15

1.10.1 `PerlPostReadRequestHandler` 15
1.10.2 `PerlTransHandler` 15
1.10.3 `PerlMapToStorageHandler` 15
1.10.4 `PerlInitHandler` 15
1.10.5 `PerlHeaderParserHandler` 15
1.10.6 `PerlAccessHandler` 15
1.10.7 `PerlAuthenHandler` 15
1.10.8 `PerlAuthzHandler` 16
1.10.9 `PerlTypeHandler` 16
1.10.10 `PerlFixupHandler` 16
1.10.11 `PerlResponseHandler` 16
1.10.12 `PerlLogHandler` 16
1.10.13 `PerlCleanupHandler` 16
1.11 Threads Mode Specific Directives 16
1.11.1 `PerlInterpStart` 16
1.11.2 `PerlInterpMax` 16
1.11.3 `PerlInterpMinSpare` 17
1.11.4 `PerlInterpMaxSpare` 17
1.11.5 `PerlInterpMaxRequests` 17
1.11.6 `PerlInterpScope` 17
1.12 Debug Directives 18
1.12.1 `PerlTrace` 18
1.13 mod_perl Directives Argument Types and Allowed Location 19
1.14 Server Startup Options Retrieval 21
1.14.1 `MODPERL2` Define Option 22
1.15 Perl Interface to the Apache Configuration Tree 22
1.16 Adjusting `@INC` 22
1.16.1 `PERL5LIB` and `PERLLIB` Environment Variables 23
1.16.2 Modifying `@INC` on a Per-VirtualHost 23
1.17 General Issues 24
1.18 Maintainers 24
1.19 Authors 24