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

1.1 Synopsis

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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

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

1.2 Description

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

1.3 API

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

1.3.1 `add_config`

Dynamically add Apache configuration at request processing runtime:

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

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

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

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

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

1.3.2 `check_cmd_context`

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

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

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

  the context to check against.

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

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

- **since: 2.0.00**

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

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

1.3.3 cmd

This module's command information

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

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

1.3.4 directive

This command's directive object in the configuration tree

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

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

  The current directive node in the configuration tree

- **since: 2.0.00**

1.3.5 info

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

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

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

  The string passed in `cmd_data`

- **since: 2.0.00**

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

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

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

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

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

1.3.6 `method_is_limited`

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

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

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

  The name of the method to check for

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

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

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

1.3.7 `override`

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

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

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

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

- **since: 2.0.00**

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

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

1.3.8 override_opts

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

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

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

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

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

- **since: 2.0.3**

1.3.9 path

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

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

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

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

- **since: 2.0.00**

For example for a container block:

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

'/foo' will be returned.

1.3.10 pool

Pool associated with this command

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

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

1.3.11 server

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

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

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

1.3.12 temp_pool

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

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

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

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

1.4 Unsupported API

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

1.4.1 context

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

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

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

  Returns the commands' per-dir config structures

- **since: 2.0.00**

1.5 See Also

mod_perl 2.0 documentation.

1.6 Copyright

1.7 Authors

The mod_perl development team and numerous contributors.

Table of Contents: