1 ModPerl::Registry - Run unaltered CGI scripts persistently under mod_perl

1 ModPerl::Registry - Run unaltered CGI scripts persistently under mod_perl

1.1 Synopsis

```
# httpd.conf
PerlModule ModPerl::Registry
Alias /perl/ /home/httpd/perl/
<Location /perl>
    SetHandler perl-script
    PerlResponseHandler ModPerl::Registry
    #PerlOptions +ParseHeaders
    #PerlOptions -GlobalRequest
    Options +ExecCGI
</Location>
```

1.2 Description

URIs in the form of `http://example.com/perl/test.pl` will be compiled as the body of a Perl subroutine and executed. Each child process will compile the subroutine once and store it in memory. It will recompile it whenever the file (e.g. *test.pl* in our example) is updated on disk. Think of it as an object oriented server with each script implementing a class loaded at runtime.

The file looks much like a "normal" script, but it is compiled into a subroutine.

For example:

```
my $r = Apache->request;
$r->content_type("text/html");
$r->send_http_header;
$r->print("mod_perl rules!");
```

XXX: STOPPED here. Below is the old Apache::Registry document which I haven’t worked through yet.

META: document that for now we don’t chdir() into the script’s dir, because it affects the whole process under threads.

This module emulates the CGI environment, allowing programmers to write scripts that run under CGI or mod_perl without change. Existing CGI scripts may require some changes, simply because a CGI script has a very short lifetime of one HTTP request, allowing you to get away with "quick and dirty" scripting. Using mod_perl and ModPerl::Registry requires you to be more careful, but it also gives new meaning to the word "quick"!

Be sure to read all mod_perl related documentation for more details, including instructions for setting up an environment that looks exactly like CGI:

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

Note that each httpd process or "child" must compile each script once, so the first request to one server may seem slow, but each request there after will be faster. If your scripts are large and/or make use of many Perl modules, this difference should be noticeable to the human eye.

1.3 Security

ModPerl::Registry::handler will preform the same checks as mod_cgi before running the script.

1.4 Environment

The Apache function ‘exit’ overrides the Perl core built-in function.

The environment variable **GATEWAY_INTERFACE** is set to `CGI-Perl/1.1`.

1.5 Commandline Switches In First Line

Normally when a Perl script is run from the command line or under CGI, arguments on the ‘#!’ line are passed to the perl interpreter for processing.

`ModPerl::Registry` currently only honors the **-w** switch and will enable the `warnings` pragma in such case.

Another common switch used with CGI scripts is **-T** to turn on taint checking. This can only be enabled when the server starts with the configuration directive:

```
PerlSwitches -T
```

However, if taint checking is not enabled, but the **-T** switch is seen, `ModPerl::Registry` will write a warning to the *error_log* file.

1.6 Debugging

You may set the debug level with the $ModPerl::Registry::Debug bitmask

```
1 => log recompile in errorlog
2 => ModPerl::Debug::dump in case of $@
4 => trace pedantically
```

1.7 Caveats

ModPerl::Registry makes things look just the CGI environment, however, you must understand that this *is not CGI*. Each httpd child will compile your script into memory and keep it there, whereas CGI will run it once, cleaning out the entire process space. Many times you have heard "always use `-w`, always use `-w` and ’use strict’". This is more important here than anywhere else!

Your scripts cannot contain the __END__ or __DATA__ token to terminate compilation. (META: works in 2.0).

1.8 Authors

Andreas J. Koenig, Doug MacEachern and Stas Bekman.

1.9 See Also

`ModPerl::RegistryCooker`, `ModPerl::RegistryBB`, `ModPerl::PerlRun`, Apache(3), mod_perl(3)

Table of Contents: