1 Configuring mod_perl 2.0 for Win32

1.1 Description

This document discusses how to configure mod_perl 2.0.

1.2 Configuration

Add this line to *C:/Apache2/conf/httpd.conf*:

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

Be sure that the path to your Perl binary (eg, *C:/Perl/bin*) is in your `PATH` environment variable. This can be done either by editing *C:\AutoExec.bat*, if present, or through the *Environment Variables* option of the *Advanced* tab of the *System* area of the Control Panel. Especially when running Apache as a service, you may also want to add the directive

```
LoadFile "/Path/to/your/Perl/bin/perl5x.dll"
```

to *httpd.conf*, before loading *mod_perl.so*, to load your Perl dll.

You may also want to use a start-up script to load commonly used modules; this can be done with a directive as, eg,

```
PerlRequire "C:/Apache2/conf/extra.pl"
```

where a sample start-up script *C:/Apache2/conf/extra.pl* is

```
use Apache2 ();
use ModPerl::Util ();
use Apache::RequestRec ();
use Apache::RequestIO ();
use Apache::RequestUtil ();
use Apache::Server ();
use Apache::ServerUtil ();
use Apache::Connection ();
use Apache::Log ();
use Apache::Const -compile => ':common';
use APR::Const -compile => ':common';
use APR::Table ();
use Apache::compat ();
use ModPerl::Registry ();
use CGI ();
1;
```

The `Apache2` module is used to add to `@INC` the relevant directories underneath, eg, */Perl/site/lib/Apache2/* used when building mod_perl 2.0 with an `MP_INST_APACHE2=1` option to `perl Makefile.PL` (the `PPM` packages discussed above were built this way). `Apache::compat` is used to provide backwards compatibility with mod_perl 1.0. `ModPerl::Registry`, named so as not to conflict with `Apache::Registry` of mod_perl 1.0, is used for registry scripts.

1.3 Registry scripts

Using `ModPerl::Registry` to speed up cgi scripts may be done as follows. Create a directory, for example, *C:/Apache2/perl/*, which will hold your scripts, such as

```
##  printenv -- demo CGI program which just prints its environment
##
use strict;
print "Content-type: text/html\n\n";
print "<HTML><BODY><H3>Environment variables</H3><UL>";
foreach (sort keys %ENV) {
  my $val = $ENV{$_};
  $val =~ s|\n|\\n|g;
  $val =~ s|"|\\"|g;
  print "<LI>$_ = \"${val}\"</LI>\n";
}
#sleep(10);
print "</UL></BODY></HTML>";
```

Note that Apache takes care of using the proper line endings when sending the *Content-type* header. Next, insert in *C:/Apache2/conf/httpd.conf* the following directives:

```
Alias /perl/ "/Apache2/perl/"
<Location /perl>
   SetHandler perl-script
   PerlResponseHandler ModPerl::Registry
   Options +ExecCGI
   PerlOptions +ParseHeaders
</Location>
```

whereby the script would be called as

```
http://localhost/perl/name_of_script
```

The `PerlOptions +ParseHeaders` directive is needed when the script sends the header (in mod_perl 1.0, this was given as `PerlSendHeader ON`).

As an illustration of how mod_perl 2.0 addresses the issues raised in the discussion of issues in multi-thread win32 concerning the threading limitations of mod_perl 1.0 on Win32, consider the `printenv` script above with the `sleep(10)` line uncommented. Using the Apache benchmarking tool `ab` of the Apache 2.0 Win32 distribution:

```
C:\Apache2\bin> ab -n 5 -c 5 http://localhost/perl/printenv
```

to make 5 concurrent requests, we find the following results. For mod_perl 1.0/Apache 1.3:

```
Server Software:        Apache/1.3.23
Concurrency Level:      5
Time taken for tests:   50.51972 seconds
```

while for mod_perl 2.0/Apache 2.0:

```
Server Software:        Apache/2.0.45
Concurrency Level:      5
Time taken for tests:   13.729743 seconds
```

The dramatic difference is due to the fact that in Apache 1.3/mod_perl 1.0 a given request has to finish (taking essentially 10 seconds, due to the `sleep(10)` call) before the next request is processed, whereas on Apache 2.0/mod_perl 2.0 the requests are processed as they arrive.

1.4 Hello World

As you will discover, there is much to mod_perl beyond simple speed-up of cgi scripts. Here is a simple *Hello, World* example that illustrates the use of mod_perl as a content handler. Create a file *Hello.pm* as follows:

```
package Apache::Hello;
use strict;

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

sub handler {
    my $r = shift;
    my $time = scalar localtime();
    my $package = __PACKAGE__;
    $r->content_type('text/html');
    $r->puts(<<"END");
<HTML><BODY>
<H3>Hello</H3>
Hello from <B>$package</B>! The time is $time.
</BODY></HTML>
END
    return Apache::OK;
}

1;
```

and save it in, for example, the *C:/Perl/site/lib/Apache2/Apache/* directory. Next put the following directives in *C:/Apache2/conf/httpd.conf*:

```
PerlModule Apache::Hello
<Location /hello>
  SetHandler modperl
  PerlResponseHandler Apache::Hello
</Location>
```

With this, calls to

```
http://localhost/hello
```

will use `Apache::Hello` to deliver the content.

1.5 See Also

The directions for installing mod_perl 2.0 on Win32, the mod_perl documentation, http://perl.apache.org/, http://take23.org/, http://httpd.apache.org/, http://www.activestate.com/, and the FAQs for mod_perl on Win32. Help is also available through the archives of and subscribing to the mod_perl mailing list.

1.6 Maintainers

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

- Randy Kobes <randy@theoryx5.uwinnipeg.ca>

1.7 Authors

- Randy Kobes <randy@theoryx5.uwinnipeg.ca>

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

Table of Contents: