1 How to use mod_perl's Method Handlers

1.1 Description

Described here are a few examples and hints on how to use method handlers with mod_perl.

This document assumes familiarity with at least the *perltoot* manpage and "normal" usage of the `Perl*Handlers`.

It isn't strictly mod_perl related, more like "what I use objects for in my mod_perl environment".

1.2 Synopsis

If a **Perl*Handler** is prototyped with '`$$`', this handler will be invoked as method, being passed a class name or blessed object as its first argument and the blessed *request_rec* as the second argument, e.g.

```
package My;
@ISA = qw(BaseClass);

sub handler ($$) {
    my ($class, $r) = @_;
    ...;
}

package BaseClass;

sub method ($$) {
    my ($class, $r) = @_;
    ...;
}

__END__
```

Configuration:

```
PerlHandler My
```

or

```
PerlHandler My->handler
```

Since the handler is invoked as a method, it may inherit from other classes:

```
PerlHandler My->method
```

In this case, the '`My`' class inherits this method from '`BaseClass`'.

To build in this feature, configure with:

```
% perl Makefile.PL PERL_METHOD_HANDLERS=1 [PERL_FOO_HOOK=1,etc]
```

1.3 Why?

The short version: For pretty much the same reasons we're using OO perl everywhere else. :-) See the *perltoot* manpage.

The slightly longer version would include some about code reusage and more clean interface between modules.

1.4 Simple example

Let's start with a simple example.

In httpd.conf:

```
<Location /obj-handler>
SetHandler perl-script
PerlHandler $My::Obj->method
</Location>
```

In startup.pl or another PerlRequire'd file:

```
package This::Class;

$My::Obj = bless {};

sub method ($$) {
    my ($obj, $r) = @_;
    $r->send_http_header("text/plain");
    print "$obj isa ", ref($obj);
    0;
}
```

which displays:

```
This::Class=HASH(0x8411edc) isa This::Class
```

1.5 A little more advanced

That wasn't really useful, so let's try something little more advanced.

I've a little module which creates a graphical 'datebar' for a client. It's reading a lot of small gifs with numbers and weekdays, and keeping them in memory in GD.pm's native format, ready to be copied together and served as gifs.

Now I wanted to use it at another site too, but with a different look. Obviously something to do with a object. Hence I changed the module to a object, and can now do a

```
$Client1::Datebar = new Datebar(
        -imagepath => '/home/client1/datebar/',
        -size      => [131,18],
        -elements  => 'wday mday mon year hour min',
);

$Client2::Datebar = new Datebar
        -imagepath => '/home/client2/datebar/',
        -size      => [90,14],
        -elements  => 'wday hour min',
);
```

And then use `$Client1::Datebar` and `$Client2::Datebar` as PerlHandlers in my Apache configuration. Remember to pass them in literal quotes ('') and not "" which will be interpolated!

I've a webinterface system to our content-database. I've created objects to handle the administration of articles, banners, images and other content. It's then very easy (a few lines of code) to enable certain modules for each client, depending on their needs.

Another area where I use objects with great success in my modperl configurations is database abstraction. All our clients using the webinterface to handle f.x. articles will use a simple module to handle everything related to the database. Each client have

```
$Client::Article = new WebAjour::Article(-host => 'www.client.com');
```

in a module what will be run at server startup.

I can then use some simple methods from the $Client::Article object in my embperl documents, like:

```
[- $c = $Client::Article->GetCursor(-layout=>'Frontpage') -]
[$ while($c->Fetch) $]
  <h2>[+ $c->f('header') +]</h2>
  [+ $c->f('textfield') +]
[$ endwhile $]
```

Very very useful!

1.6 Traps

mod_perl expects object handlers to be in the form of a string, which it will thaw for you. That means that something like

```
$r->push_handlers(PerlHandler => '$self->perl_handler_method');
```

This doesn't work as you might expect, since Perl isn't able to see $self once it goes to PerlHandler.

The best solution to this is to use an anonymous subroutine and pass it $r yourself, like this:

```
 $r->push_handlers(PerlHandler =>
     sub {
         my $r = shift;
         $self->perl_handler_method($r);
     }
 );
```

1.7 Author

This document is written by Ask Bjoern Hansen <ask@netcetera.dk> or <ask@apache.org>. Corrections and suggestions are most welcome. In particular would more examples be appreciated, most of my own code is way too integrated with our system, which isn't suitable for public release.

Some codesnippets is from Doug MacEachern.

1.8 See Also

The *Apache*, the *perltoot* manpages (also available at `http://www.perl.com/CPAN/doc/FMTEYEWTK/perltoot.html`)

1.9 Maintainers

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

- modperl docs list

1.10 Authors

- Ask Bjoern Hansen, <ask (at) netcetera.dk> or <ask (at) apache.org>.

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

Table of Contents: