1 mod_perl Coding Style Guide

1.1 Description

This document explains the coding style used in the core mod_perl development and which should be followed by all core developers.

1.2 Coding Style Guide

We try hard to code mod_perl using an identical style. Because everyone in the team should be able to read and understand the code as quickly and easily as possible. Some will have to adjust their habits for the benefit of all.

- **C code**

  mod_perl's C code follows the Apache style guide: http://dev.apache.org/styleguide.html

- **XS code**

  C code inside XS modules also follows the Apache style guide.

- **Perl code**

  mod_perl's Perl code also follows the Apache style guide, in terms of indentation, braces, etc. Style issues not covered by Apache style of guide should be looked up in the *perlstyle* manpage.

Here are the rough guidelines with more stress on the Perl coding style.

- **Indentation and Tabs**

  Do use 4 characters indentation.

  Do NOT use tabs.

  Here is how to setup your editor to do the right thing:

  - **x?emacs: cperl-mode**

    ```
    .xemacs/custom.el:
    ------------------
    (custom-set-variables
       '(cperl-indent-level 4)
       '(cperl-continued-statement-offset 4)
       '(cperl-tab-always-indent t)
       '(indent-tabs-mode nil)
    )
    ```

  - **vim**

```
.vimrc:
-------
set expandtab " replaces any tab keypress with the appropriate number of spaces
set tabstop=4 " sets tabs to 4 spaces
```

- **Block Braces**

  Do use a style similar to K&R style, not the same. The following example is the best guide:

  Do:

```
sub foo {
    my($self, $cond, $baz, $taz) = @_;

    if ($cond) {
        bar();
    }
    else {
        $self->foo("one", 2, "...");
    }

    return $self;
}
```

  Don't:

```
sub foo
{
    my ($self,$bar,$baz,$taz)=@_;
    if( $cond )
    {
        &bar();
    } else { $self->foo ("one",2,"..."); }
    return $self;
}
```

- **Lists and Arrays**

  Whenever you create a list or an array, always add a comma after the last item. The reason for doing this is that it's highly probable that new items will be appended to the end of the list in the future. If the comma is missing and this isn't noticed, there will be an error.

  Do:

```
my @list = (
    "item1",
    "item2",
    "item3",
);
```

  Don't:

```
my @list = (
    "item1",
    "item2",
    "item3"
);
```

- **Last Statement in the Block**

  The same goes for `;` in the last statement of the block. Almost always add it even if it's not required, so when you add a new statement you don't have to remember to add `;` on a previous line.

  Do:

```
sub foo {
    statement1;
    statement2;
    statement3;
}
```

  Don't:

```
sub foo {
    statement1;
    statement2;
    statement3
}
```

1.3 Function and Variable Prefixes Convention

- **modperl_**

  The prefix for mod_perl C API functions.

- **MP_**

  The prefix for mod_perl C macros.

- **mpxs_**

  The prefix for mod_perl XS utility functions.

- **mp_xs_**

  The prefix for mod_perl *generated* XS utility functions.

- **MPXS_**

  The prefix for mod_perl XSUBs with an XS() prototype.

1.4 Coding Guidelines

The following are the Perl coding guidelines:

1.4.1 Global Variables

- **avoid globals in general**
- **avoid $&, $', $`**

  See `Devel::SawAmpersand`'s *README* that explains the evilness. Under mod_perl everybody suffers when one is seen anywhere since the interpreter is never shutdown.

1.4.2 Modules

- **Exporting/Importing**

  Avoid too much exporting/importing (glob aliases eat up memory)

  When you do wish to import from a module try to use an explicit list or tag whenever possible, e.g.:

  ```
  use POSIX qw(strftime);
  ```

  When you do not wish to import from a module, always use an empty list to avoid any import, e.g.:

  ```
  use IO::File ();
  ```

  (explain how to use Apache::Status to find imported/exported functions)

1.4.3 Methods

- **indirect vs direct method calls**

  Avoid indirect method calls, e.g.

  Do:

  ```
  CGI::Cookie->new
  ```

  Don't:

  ```
  new CGI::Cookie
  ```

1.4.4 Inheritance

- **Avoid inheriting from certain modules**

Exporter. To avoid inheriting **AutoLoader::AUTOLOAD**

Do:

```
*import = \&Exporter::import;
```

Don't:

```
@MyClass::ISA = qw(Exporter);
```

1.4.5 Symbol tables

- **%main::**

  stay away from `main::` to avoid namespace clashes

1.4.6 Use of $_ in loops

Avoid using `$_` in loops unless it's a short loop and you don't call any subs from within the loop. If the loop started as short and then started to grow make sure to remove the use of `$_`:

Do:

```
for my $idx (1..100) {
    ....more than few lines...
    foo($idx);
    ....
}
```

Don't:

```
for (1..100) {
    ....more than a few statements...
    foo();
    ....
}
```

Because foo() might change `$_` if foo()'s author didn't localize `$_`.

This is OK:

```
for (1..100) {
    .... a few statements with no subs called
    # do something with $_
    ....
}
```

1.5 Maintainers

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

Stas Bekman <stas *at* stason.org>

1.6 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: