PSR-1: Basic Coding Standard

This section of the standard comprises what should be considered the standard coding elements that are required to ensure a high level of technical interoperability between shared PHP code.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1. Overview

- Files MUST use only `<?php` and `<?=` tags.
- Files MUST use only UTF-8 without BOM for PHP code.
- Files SHOULD *either* declare symbols (classes, functions, constants, etc.) *or* cause side-effects (e.g. generate output, change .ini settings, etc.) but SHOULD NOT do both.
- Namespaces and classes MUST follow an "autoloading" PSR: [PSR-0, PSR-4].
- Class names MUST be declared in `StudlyCaps`.
- Class constants MUST be declared in all upper case with underscore separators.
- Method names MUST be declared in `camelCase`.

2. Files

2.1. PHP Tags

PHP code MUST use the long `<?php ?>` tags or the short-echo `<?= ?>` tags; it MUST NOT use the other tag variations.

2.2. Character Encoding

PHP code MUST use only UTF-8 without BOM.

2.3. Side Effects

A file SHOULD declare new symbols (classes, functions, constants, etc.) and cause no other side effects, or it SHOULD execute logic with side effects, but SHOULD NOT do both.

The phrase "side effects" means execution of logic not directly related to declaring classes, functions, constants, etc., *merely from including the file*.

"Side effects" include but are not limited to: generating output, explicit use of `require` or `include`, connecting to external services, modifying ini settings, emitting errors or exceptions, modifying global or static variables, reading from or writing to a file, and so on.

The following is an example of a file with both declarations and side effects; i.e, an example of what to avoid:

```
<?php
// side effect: change ini settings
ini_set('error_reporting', E_ALL);

// side effect: loads a file
include "file.php";

// side effect: generates output
echo "<html>\n";

// declaration
function foo()
{
    // function body
}
```

The following example is of a file that contains declarations without side effects; i.e., an example of what to emulate:

```
<?php
// declaration
function foo()
{
    // function body
}

// conditional declaration is *not* a side effect
if (! function_exists('bar')) {
    function bar()
    {
        // function body
    }
}
```

3. Namespace and Class Names

Namespaces and classes MUST follow an "autoloading" PSR: [PSR-0, PSR-4].

This means each class is in a file by itself, and is in a namespace of at least one level: a top-level vendor name.

Class names MUST be declared in `StudlyCaps`.

Code written for PHP 5.3 and after MUST use formal namespaces.

For example:

```
<?php
// PHP 5.3 and later:
namespace Vendor\Model;

class Foo
{
}
```

Code written for 5.2.x and before SHOULD use the pseudo-namespacing convention of `Vendor_` prefixes on class names.

```
<?php
// PHP 5.2.x and earlier:
class Vendor_Model_Foo
{
}
```

4. Class Constants, Properties, and Methods

The term "class" refers to all classes, interfaces, and traits.

4.1. Constants

Class constants MUST be declared in all upper case with underscore separators. For example:

```
<?php
namespace Vendor\Model;

class Foo
{
    const VERSION = '1.0';
    const DATE_APPROVED = '2012-06-01';
}
```

4.2. Properties

This guide intentionally avoids any recommendation regarding the use of `$StudlyCaps`, `$camelCase`, or `$under_score` property names.

Whatever naming convention is used SHOULD be applied consistently within a reasonable scope. That scope may be vendor-level, package-level, class-level, or method-level.

4.3. Methods

Method names MUST be declared in `camelCase()`.