Coding Standards

As we are using mostly PHP for our project, we are following PHP coding standard PSR-2.

PSR-2 is an extension of the PSR-1 coding standard. Some examples of its contents are:

1. **We must follow PSR-1 coding standards:**

**PSR-1 Basic Coding Standard**

The more basic parts of PHP coding standards are defined in PSR-1. For example:

Only <?php or <?= are allowed for PHP tags

Files must be in UTF-8 without BOM(Byte Order Mark)

Namespaces and class names must follow the standards in PSR-0 and PSR-4

Class names must be defined in UpperCamelCase

Class variables must be defined in UPPER\_SNAKE\_CASE

Method names must be defined in camelCase

Standard functions in PHP are defined in snake\_case, but in PSR-1, method names must be defined in camelCase. There are no explicit rules for variable and property names, so you can use whichever style you like, but it is noted that they should be consistent. For example, defining normal properties in camelCase and static properties in UpperCamelCase like below:

*class Something*

*{*

*public $normalPropterty;*

*public static $StaticProperty;*

*}*

1. **4 spaces must be used for indents. Using tabs is not allowed**
2. **There is no limit to line length, but it should be under 120 characters, and best if under 80**
3. **We must put a newline before curly braces for classes and methods**
4. **Methods and properties must be defined with abstract/final first, followed with public/protected, and finally static.**
5. **We must not put a newline before curly braces in conditional statements**
6. **We must not put any spaces before ( and ) in conditional statements**

**Defining Classes**

We must put a newline before { in class definitions. Also, extends and implements must be written on the same line as the class name.

*class ClassName extends ParentClassName implements Interface1, Interface2*

*{*

*// Class definition*

*}*

If there are too many interfaces for one line, we should put a newline after implements and write one interface per line like below.

*class ClassName extends ParentClassName implements*

*Interface1,*

*Interface2,*

*Interface3,*

*Interface4*

*{*

*// Class definition*

*}*

**Defining Properties**

In PSR-2, we must not omit public/protected/private modifiers. In PHP, properties become public if these are omitted, but because it is hard to tell if one purposely omitted these modifiers or they just forgot, we should always explicitly write public. The static keyword comes next. We must not use var when defining properties because we can't add any modifiers to var.

*class ClassName*

*{*

*public $property1;*

*private $property2;*

*public static $staticProperty;*

*}*

*class ClassName*

*{*

*private $property1, $property2;*

*}*

**Methods**

Like properties, we must have either one of public/protected/private and abstract/final comes after them if used. static is the last modifier. We must not put any spaces before and after braces, and we must put a newline before curly braces. Also, we must not put any whitespaces before commas in arguments, and we must put one whitespace after them.

*class ClassName*

*{*

*abstract protected function abstractDoSomething();*

*final public static function doSomething($arg1, $arg2, $arg3)*

*{*

*// ...*

*}*

*}*

If there are too many arguments, we can put a newline after ( and write one argument per line. In this case, we can't write multiple variables on one line. Also, we should write ) and { on the same line, separated by a whitespace.

*class ClassName*

*{*

*public function doSomething(*

*TypeHint $arg1,*

*$arg2,*

*$arg3,*

*$arg4*

*) {*

*// ...*

*}*

*}*

Please note that you must not put a newline before { in closures.

$closure = function ($a, $b) use ($c) {

// Body

};

**Conditional Statements**

For conditional statements,

We must put one whitespace before (

We must not put any whitespaces after (

We must not put any whitespaces before )

We must put one whitespace after )

Also, use elseif rather than else if.

*if ($condition1) {*

*// ...*

*} elseif ($condition2) {*

*// ...*

*} else {*

*// ...*

*}*

We have to be careful, else if and elseif are not the complete same things. elseif is one statement by itself, but else if on the other hand is interpreted as an if statement in the else of the first if.

*if ($condition1) {*

*// ...*

*} else if ($condition2) {*

*// ...*

*} else {*

*// ...*

*}*

*The syntax above is actually interpreted like below:*

*if ($condition1) {*

*// ...*

*} else {*

*if ($condition2) {*

*// ...*

*} else {*

*// ...*

*}*

*}*

For switch statements, case statements must be indented once from switch, and bodies for the cases must be indented once from case. When not breaking after any kind of operations in case, we must write a comment.

*switch ($condition) {*

*case 0:*

*echo 'First case, with a break';*

*break;*

*case 1:*

*echo 'Second case, which falls through';*

*// no break*

*case 2:*

*case 3:*

*case 4:*

*echo 'Third case, return instead of break';*

*return;*

*default:*

*echo 'Default case';*

*break;*

*}*

Documentation

We are using phpDox. phpDox is the documentation generator for PHP projects.

This includes, but is not limited to, API documentation. By default, phpDox uses PHP-Parser to collect information about a PHP codebase.

The information collected is stored in XML documents.These XML documents can be enriched with information from external sources, like XML logfiles of PHP\_CodeSniffer, PHP Mess Detector (PHPMD) or PHPUnit.

This information is rendered to HTML, for instance, by applying XSL transformations to the XML data.