PHP-Fusion Coding Coding Style

PHP-Fusion Version: 7.02.00

Table of Contents

Overview

License

Scope

This document provides guidelines for code formatting and documentation to individuals and teams contributing to PHP-Fusion. Many developers using PHP-Fusion have also found these coding standards useful because their code's style remains consistent with all PHP-Fusion code. It is also worth noting that it requires significant effort to fully specify coding standards.

Note

Sometimes developers consider the establishment of a standard more important than what that standard actually suggests at the most detailed level of design. The guidelines in PHP-Fusion's coding standards capture practices that have worked well on the PHP-Fusion project. You may modify these standards or use them as is in accordance with the terms on Zend Framework's License Page.

Topics covered in PHP-Fusion's coding standards include:

1. Naming Conventions
2. Variables
3. Classes and Functions
4. Control Statements
5. Comments

Coding standards are important in any development project, but they are particularly important when many developers are working on the same project. Coding standards help ensure that the code is high quality, has fewer bugs, and can be easily maintained.

Naming Convention

Classes

Class names may only contain alphanumeric characters. Numbers are permitted in class names but are discouraged in most cases. Underscores are not permitted.

If a class name is comprised of more than one word, the first letter of each new word, including the first work, must be capitalised. Successive capitalised letters are not allowed, e.g. a class "MYClass" is not allowed while "MyClass" is acceptable.

Folders

Only alphanumeric characters and underscores are permitted. Spaces are strictly prohibited. New words are separated by underscores and the first letter of each new word, including the first work, must be lower case.

Filenames

Only alphanumeric characters and underscores are permitted. Spaces are strictly prohibited. New words are separated by underscores and the first letter of each new word, including the first work, must be lower case, with the notable exception of class files.

Any file that contains PHP code should end with the extension ".php". The following examples show acceptable filenames for PHP-Fusion classes:

- includes/MyClass.class.php
- infusions/my_infusion/MyClass.class.php

File names must map to the class names as described above with the correct capital letters for the class, without the .class at the end. Only one class is permitted per class file. Class files can only be placed in the includes folder in PHP-Fusion root or inside an infusion.

Following example show other acceptable file names:

- myfile.php
- infusions/my_infusion/myfile.php

Functions

Function names may only contain alphanumeric characters. Underscores are not permitted. Numbers are permitted in function names but are discouraged in most cases.

Function names must always start with a lowercase letter. When a function name consists of more than one word, the first letter of each new word must be capitalised. This is commonly called "camelCase" formatting.

Verbosity is generally encouraged. Function names should be as verbose as is practical to fully describe their purpose and behaviour.

These are examples of acceptable names for functions:

- filterInput()
- getElementById()
- widgetFactory()

For object-oriented programming, accessors for instance or static variables should always be prefixed with "is ", "get" or "set". In implementing design patterns, such as the singleton or factory patterns, the name of the method should contain the pattern name where practical to more thoroughly describe behaviour.

For methods on objects that are declared with the "private" or "protected" modifier, the first character of the method name must be an underscore. This is the only acceptable application of an underscore in a method name. Methods declared "public" should never contain an underscore.

Functions in the global scope (a.k.a "floating functions") are permitted but discouraged in most cases. Consider wrapping these functions in a static class.

Variables
Variable names may only contain alphanumeric characters. Underscores are not permitted. Numbers are permitted in variable names but are discouraged in most cases.

For instance variables that are declared with the "private" or "protected" modifier, the first character of the variable name must be a single underscore. This is the only acceptable application of an underscore in a variable name. Member variables declared "public" should never start with an underscore.

As with function names (see section 3.3) variable names must always start with a lowercase letter and follow the "camelCaps" capitalisation convention.

Verbosity is generally encouraged. Variables should always be as verbose as practical to describe the data that the developer intends to store in them. Terse variable names such as "$i", $ii, $n and "$m" are discouraged for all but the smallest loop contexts. If a loop contains more than 20 lines of code, the index variables should have more descriptive names.

Variables names like $foo, $bar or $test are not permitted.

Constants
Constants may contain both alphanumeric characters and underscores. Numbers are permitted in constant names.

All letters used in a constant name must be capitalised, while all words in a constant name must be separated by underscore characters. For example:

MY_CONSTANT is permitted but MYCONSTANT is not.

Variables

PHP Code Demarcation
PHP code must always be delimited by the full-form, standard PHP tags:

```
1. <?php
2.
3. ?>
```

Short tags or not adding ?> at the end of a PHP file are never allowed.

String Literals
When a string is literal (contains no variable substitutions), "double quotes" should always be used to demarcate the string:

```
1. $a = "Example String";
```

Variable Substitution
Variable substitution is permitted using this form:

```
1. $greeting = "Hello " .$name. ", welcome back!";
```

For consistency, this form is not permitted:

```
1. $greeting = "Hello $name, welcome back!";
```

String Concatenation
Strings must be concatenated using the "." operator. A space must never be added before or after the ".":

```
1. $company = "PHP- Fusion"."Solutions";
```

It is not necessary concatenating strings with the "." operator when breaking a the statement into multiple line. In these cases, each successive line should be padded with white space such that the start of the new line is aligned under or passed the "=" operator:

```
1. $sql = "SELECT user_id, user_name FROM ".DB_USERS."
2.       WHERE user_level = '103'
3.       ORDER BY user_name ASC";
```

Numerically Indexed Arrays
Negative numbers are not permitted as indices.

An indexed array may start with any non-negative number, however all base indices besides 0 are discouraged.

When declaring indexed arrays with the Array function, a trailing space must be added after each comma delimiter to improve readability:

```
1. $sampleArray = array(1, 2, 3, "PHP-Fusion", "Solutions");
```

It is permitted to declare multi-line indexed arrays using the "array" construct. The initial array item must begin on the following line. It should be padded at one indentation level greater than the line containing the array declaration, and all successive lines should have the same indentation; the closing paren should be on a line by itself at the same indentation level as the line containing the array declaration:

```
1.  $sampleArray = array(
2.          1, 2, 3, "PHP-Fusion", "Solutions",
3.          $a, $b, $c,
4.          56.44, $d, 500
5.  );
```

We require using a trailing comma for the last item in the array; this minimises the impact of adding new items on successive lines, and helps to ensure no parse errors occur due to a missing comma.

Associative Arrays

When declaring associative arrays with the Array construct, the initial array item must begin on the following line. It should be padded at one indentation level greater than the line containing the array declaration, and all successive lines should have the same indentation; the closing parent should be on a line by itself at the same indentation level as the line containing the array declaration. For readability, the various "=>" assignment operators should be padded such that they align.

```
1.  $sampleArray = array(
2.          "firstKey"  => "firstValue",
3.          "secondKey" => "secondValue",
4.  );
```

We require using a trailing comma for the last item in the array; this minimises the impact of adding new items on successive lines, and helps to ensure no parse errors occur due to a missing comma.

Classes and functions

Class Declaration

Classes must be named according to PHP-Fusion's naming conventions. The brace should always be written on the same line as the class name. All code in a class must be indented with one tab and only one class is permitted in each PHP file.

Placing additional code in class files is permitted but discouraged. In such files, two blank lines must separate the class from any additional PHP code in the class file, the only exception is additional classes extending the main class.

The following is an example of an acceptable class declaration:

```
1. class SampleClass {
2.       // all contents of class
3.       // must be indented by one tab
4. }
```

Classes that extend other classes or which implement interfaces should declare their dependencies on the same line when possible.

```
1. class SampleClass extends FooAbstract {
2. }
```

Class Member Variables

Member variables must be named according to PHP-Fusion's variable naming conventions. Any variables declared in a class must be listed at the top of the class, above the declaration of any methods.

The var construct is not permitted. Member variables always declare their visibility by using one of the private, protected, or public modifiers. Giving access to member variables directly by declaring them as public is permitted but discouraged in favour of accessor methods (is, set & get).

Function and Method Declaration

Functions must be named according to PHP-Fusion's function naming conventions. Methods inside classes must always declare their visibility by using one of the private, protected, or public modifiers. As with classes, the brace should always be written on the same line as the function name. Space between the function name and the opening parenthesis for the arguments is not permitted.

Functions in the global scope are strongly discouraged.

The following is an example of an acceptable function declaration in a class:

```
1. class MyClass {
2.       // Method description here
3.       public function myFunction() {
4.            // all contents of function
5.            // must be indented by one tab
6.       }
7. }
```

In cases where the argument list exceeds the maximum line length, you may introduce line breaks. Additional arguments to the function or method must be indented one additional level beyond the function or method name. A line break should then occur before the closing argument parent, which should then be placed on the same line as the opening brace of the function or method with one space separating the two, and at the same indentation level as the function or method declaration. The following is an example of one such situation:

```
1.  class MyClass {
2.  // Method description here
3.       public function myFunction($arg1, $arg2, $arg3,
4.                                  $arg4, $arg5, $arg6) {
5.            // all contents of function
6.            // must be indented one tab
7.       }
8.  }
```

The return value must not be enclosed in parentheses. This can hinder readability, in additional to breaking code if a method is later changed to return by reference.

```
1.  class MyClass {
2.       // WRONG
3.       public function myFunction() {
4.            return($this->value);
5.       }
6.
7.       // RIGHT
8.       public function myFunction() {
9.            return $this->value;
10.      }
11. }
```

Function and Method Usage

Function arguments should be separated by a single trailing space after the comma delimiter. The following is an example of an acceptable invocation of a function that takes three arguments:

```
1.  threeArguments(1, 2, 3);
```

Call-time pass-by-reference is strictly prohibited. See the function declarations section for the proper way to pass function arguments by-reference.

In passing arrays as arguments to a function, the function call may include the "array" hint and may be split into multiple lines to improve readability. In such cases, the normal guidelines for writing arrays still apply:

```
1.  threeArguments(array(1, 2, 3), 2, 3);
```

```
1.  threeArguments(array(
2.       1, 2, 3, 'PHP-Fusion', 'Solutions',
3.       $a, $b, $c,
4.       56.44, $d, 500
5.  ), 2, 3);
```

Control Statements

if / else / elseif

Control statements based on the if and elseif constructs must have a single space before the opening parenthesis of the conditional and a single space after the closing parenthesis.

Within the conditional statements between the parentheses, operators must be separated by spaces for readability. Inner parentheses are encouraged to improve logical grouping for larger conditional expressions.

The opening brace is written on the same line as the conditional statement. The closing brace is always written on its own line. Any content within the braces must be indented using one tab.

```
1.  if ($a != 2) {
2.          $a = 2;
3.  }
```

If the conditional statement causes the line length to exceed the maximum line length and has several clauses, you may break the conditional into multiple lines. In such a case, break the line prior to a logic operator, and pad the line such that it aligns under the first character of the conditional clause. The closing parent in the conditional will then be placed on a line with the opening brace, with one space separating the two, at an indentation level equivalent to the opening control statement.

```
1.  if (($a == $b)
2.          && ($b == $c)
3.          || (MyClass::myFunction() == $d)
4.  ) {
5.          $a = $d;
6.  }
```

The intention of this latter declaration format is to prevent issues when adding or removing clauses from the conditional during later revisions.

For "if" statements that include "elseif" or "else", the formatting conventions are similar to the "if" construct. The following examples demonstrate proper formatting for "if" statements with "else" and/or "elseif" constructs:

```
1.  if ($a != 2) {
2.          $a = 2;
3.  } else {
4.          $a = 7;
5.  }
```

```
1.  if ($a != 2) {
2.          $a = 2;
3.  } elseif ($a == 3) {
4.          $a = 4;
5.  } else {
6.          $a = 7;
7.  }
```

```
if ((($a == $b)
      && ($b == $c)
      || (MyClass::myFunction() == $d)
) {
      $a = $d;
} elseif (($a != $b)
      || ($b != $c)
) {
      $a = $c;
} else {
      $a = $b;
}
```

PHP allows statements to be written without braces in some circumstances. This coding standard makes no differentiation - All "if", "elseif" or "else" statements must use braces.

Switch
Control statements written with the "switch" statement must have a single space before the opening parenthesis of the conditional statement and after the closing parenthesis.

All content within the "switch" statement must be indented using one tab. Content under each "case" statement must be indented using an additional one tab.

```
switch ($numPeople) {
      case 1:
            break;
      case 2:
            break;
      default:
            break;
}
```

The construct default should never be omitted from a switch statement.

Note: It is sometimes useful to write a case statement which falls through to the next case by not including a break or return within that case. To distinguish these cases from bugs, any case statement where break or return are omitted should contain a comment indicating that the break was intentionally omitted.

For
Control statements written with the "for" statement must have a single space before the opening parenthesis of the conditional statement and after the closing parenthesis.

The opening brace is written on the same line as the "for" statement. The closing brace is always written on its own line. Any content within the braces must be indented using one tab.

```
for  ($i = 0; $i < 5; $i++) {
        // content goes here
}
```

Comments

Header comment

All files must at the line below <?php add the following header comment:

```
/*---------------------------------------------------------+
| PHP-Fusion Content Management System
| Copyright (C) 2002 - 2010 Nick Jones
| http://www.php-fusion.co.uk/
+---------------------------------------------------------+
| Filename: filename.php
| Author: My Name
+---------------------------------------------------------+
| This program is released as free software under the
| Affero GPL license. You can redistribute it and/or
| modify it under the terms of this license which you
| can read by viewing the included agpl.txt or online
| at www.gnu.org/licenses/agpl.html. Removal of this
| copyright header is strictly prohibited without
| written permission from the original author(s).
+---------------------------------------------------------*/
```

After this you are allowed to add any code.

Inline comments

All inline comments are added using two slashes "//" and one space after the slashes:

```
// My comment goes here
```