Zend Framework Coding Standard for PHP

This document provides guidelines for code formatting and documentation. 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.

Topics covered in the ZF coding standards include:

- **PHP File Formatting**
- **Naming Conventions**
- **Coding Style**
- **Inline Documentation**

1. PHP File Formatting

- For files that contain only PHP code, the closing tag ("?>") is never permitted. It is not required by PHP, and omitting it prevents the accidental injection of trailing whitespace into the response.
- Indentation should consist of 4 spaces. Tabs are **not** allowed.
- Keep each line of the code under 80 characters.
- Lines must end with a single linefeed (LF) character. Do not use carriage returns (CR) or the carriage return/linefeed combination (CRLF)

2. Naming Conventions

2.1 Classes

- Zend Framework standardizes on a class naming convention whereby the names of the classes directly map to the directories in which they are stored.

  e.g The class *"Zend_Db_Table"* should be stored in *"Zend/Db/Table"* where *"Zend/"* is the root level directory of the ZF standard library.
  All Zend Framework classes are stored hierarchically under these root directories.

- Class names may only contain alphanumeric characters. Numbers are permitted in class names but are discouraged in most cases. Underscores are only permitted in place of the path separator.

  e.g. The filename *"Zend/Db/Table.php"* must map to the class name *"Zend_Db_Table"*.

- If a class name is comprised of more than one word, the first letter of each new word must be capitalized. Successive capitalized letters are not allowed.
  e.g. a class *"Zend_PDF"* is not allowed while *"Zend_Pdf"* is acceptable.

2.2 Filenames

- Use only alphanumeric characters, but underscores and dash character ("-") are permitted also.
- Spaces are strictly prohibited.
- Any file that contains PHP code should end with the extension ".php", with the notable exception of **view scripts**.
  e.g. *Zend/Db.php*
  Zend/Controller/Front.php
  Zend/View/Helper/FormRadio.php

- File names must map to class names as described above.

2.3 Functions and Methods

- 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 capitalized. This is commonly called **"camelCase"** formatting.

- Function names should be as verbose as is practical to fully describe their purpose and behavior.
  e.g. *filterInput()*
  getElementById()

- For object-oriented programming, accessors for instance or static variables should always be prefixed with "get" or "set".

- For methods on objects that are declared with the "private" or "protected" modifier, the first character of the variable 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.

2.4 Variables

- Variables naming convention is exactly same as the function names.
- As with function names variable names must always start with a lowercase letter and follow the ***"camelCaps"*** capitalization convention.

- Terse variable names such as "$i" and "$n" 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.

2.5 Constants

- Constants may contain both alphanumeric characters and underscores. Numbers are permitted in constant names.
- All letters used in a constant name must be capitalized.
- Words in constant names must be separated by underscore characters.
  e.g. *EMBED_SUPPRESS_EMBED_EXCEPTION* is permitted but *EMBED_SUPPRESSEMBEDEXCEPTION* is not.
- Constants must be defined as class members with the "const" modifier. Defining constants in the global scope with the "define" function is permitted but strongly discouraged.

3. Coding Style

3.1 PHP Code Demarcation

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

```
<?php
?>
```

- For files containing only PHP code, the closing tag must always be omitted

3.2 Strings

- String Literals

- When a string is literal (contains no variable substitutions), the apostrophe or "single quote" should always be used to demarcate the string:
  e.g. *$a = 'Example String';*

- When a literal string itself contains apostrophes, it is permitted to demarcate the string with quotation marks or "double quotes". This is especially useful for SQL statements:

```
e.g. $sql = "SELECT `id`, `name`
             from `people`
            WHERE `name`='Fred' OR `name`='Susan'";
```

This syntax is preferred over escaping apostrophes as it is much easier to read.

- String Concatenation

- Strings must be concatenated using the "." operator.
- A space must always be added before and after the"." operator to improve readability:
  e.g. *$company = 'Zend' . ' '. 'Technologies';*

- When concatenating strings with the "." operator, it is encouraged to break the statement into multiple lines to improve readability. In these cases, each successive line should be padded with whitespace such that the "." operator is aligned under the "=" operator:

e.g. *$sql = "SELECT `id`, `name` FROM `people` "*
. "WHERE `name` = 'Susan' "
. "ORDER BY `name` ASC ";

3.3 Arrays

-Numerically Indexed Arrays

- Negative numbers are not permitted as indices.
- 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.
  e.g. *$sampleArray = array(1, 2, 3, 'Zend', 'Studio');*
- It is permitted to declare multiline indexed arrays using the "array" construct. In this case, each successive line must be padded with spaces such that beginning of each line is aligned:
  e.g. *$sampleArray = array(1, 2, 3, 'Zend', 'Studio',*
  $a, $b, $c,
  56.44, $d, 500);

3.4 Class Declaration

- Classes must be named according to Zend Framework's naming conventions.
- The brace should always be written on the line underneath the class name (the "one true brace" form).
- Every class must have a documentation block that conforms to the PHPDocumentor standard.
- All code in a class must be indented with **four spaces**.
- 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.
  e.g.

```
/**
* Documentation Block Here
*/
class SampleClass
{
// all contents of class
// must be indented four spaces
}
Function and Method Declaration
/**
* Documentation Block Here
*/
```

```
class Foo
{
/**
* Documentation Block Here
*/
public function bar()
{
// all contents of function
// must be indented four spaces
}
}
```

- Pass-by-reference is the only parameter passing mechanism permitted in a method declaration.

- Call-time pass-by-reference is strictly prohibited.
- 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.

  e.g.

```
public function bar()
{
return $this->bar;
}
```

3.5 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 four spaces.

e.g.

```
if ($a != 2) {
    $a = 2;
}
if ($a != 2) {
    $a = 2;
} elseif ($a == 3) {
    $a = 4;
} else {
```

```
    $a = 7;
}
```

- Use of the "elseif" construct is permitted but strongly discouraged in favor of the "else if" combination.

-Switch

- All content within the "switch" statement must be indented using four spaces.
- Content under each "case" statement must be indented using an additional four spaces.

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

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

4. Inline Documentation

4.1 Documentation Format

- All documentation blocks *("docblocks")* must be compatible with the phpDocumentor format*.(visit: http://phpdoc.org/)*
- All class files must contain a *"file-level*" *docblock* at the top of each file and a *"class-level" docblock* immediately above each class.

-File

- Every file that contains PHP code must have a *docblock* at the top of the file that contains these *phpDocumentor* tags at a minimum:

```
/**
* Short description for file
*
* Long description for file (if any)...
*
* LICENSE: Some license information
*
* @copyright 2008 Zend Technologies
* @license http://framework.zend.com/license BSD License
* @version $Id:$
* @link http://framework.zend.com/package/PackageName
```

```
 * @since File available since Release 1.5.0
 */
```

-Classes

- Every class must have a *docblock* that contains these *phpDocumentor* tags at a minimum:

```
/**
* Short description for class
*
* Long description for class (if any)...
*
* @copyright 2008 Zend Technologies
* @license http://framework.zend.com/license BSD License
* @version Release: @package_version@
* @link http://framework.zend.com/package/PackageName
* @since Class available since Release 1.5.0
* @deprecated Class deprecated in Release 2.0.0
*/
```

- Functions

- Every function, including object methods, must have a *docblock* that contains at a minimum:
  - A description of the function
  - All of the arguments
  - All of the possible return values

- If a function/method may throw an exception, use @throws for all known exception classes:

  @throws exceptionclass [description]