Coding standard

Version: 1.0 Rev: #003/04-03-2013

WEBINY LTD
Registered in England and Wales
No: 8222574
DEPT 400, 61 PRAED STREET
LONDON W2 1NS

1

Overview

Scope

This document will give you coding guidelines that you should follow during your work and contribution to **Webiny platform**.

By following these **guidelines** your code will use the **same notation and standard** that other developers on the project use, making your code **more readable** for them, and vice-versa.

This coding standard follows guidelines set by other respectable communities such as Zend, Symfony, PEAR and PHP FIG.

References:

- https://github.com/php-fig/fig-standards/tree/master/accepted
- http://framework.zend.com/manual/1.12/en/coding-standard.html
- http://symfony.com/doc/2.0/contributing/code/standards.html
- http://pear.php.net/manual/en/standards.php

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 (http://www.ietf.org/rfc/rfc2119.txt).

2

File structure and naming

General

The organisation of your files and their names is mandatory to follow the defined guidelines in order to be in alignment for **Webiny autoloader**.

- The file and folder structure SHOULD follow this organisation pattern: **Vendor/Package/SubPackage/FileName.php**
- Each **class** MUST have a **namespace** that follows folder structure.
- The **namespace** separator will be replaced with **DIRECTORY_SEPARATOR** upon loading the file from the system.
- Each '_' (underscore) character in the **CLASS NAME** will be replaced with **DIRECTORY_SEPARATOR.** Note that the same rule DOES NOT apply to **namespaces**.

Examples

- **\Doctrine\Common\IsolatedClassLoader** is mapped to **/path/to/project/lib/vendor/Doctrine/Common/IsolatedClassLoader.php**
- **\Symfony\Core\Request** is mapped to **/path/to/project/lib/vendor/Symfony/Core/Request.php**
- **\namespace\Package_Name\Class_Name** is mapped to **/path/to/project/lib/vendor/namespace/Package_Name/Class/Name.php**

Additional rules

- File names SHOULD BE written in **StudlyCase**;
  Example: **MyClassName.php**
- **Interfaces** SHOULD BE formed of ClassName+”Interface” keyword.
  Example: **MyClassInterface.php**
  The same rule applies to **Traits** and **Exceptions.**
- **Abstract** and **Static classes** SHOULD BE named like ClassName+”Abstract” (or “Static) keyword.
  Example: **MyClassAbstract.php**

Example autoloader function

If you follow the before specified guidelines for naming your files, this autoloader should work for your files.

```
<?php

function autoload($className)
{
    $className = ltrim($className, '\\');
    $fileName  = '';
    $namespace = '';
    if ($lastNsPos = strrpos($className, '\\')) {
        $namespace = substr($className, 0, $lastNsPos);
        $className = substr($className, $lastNsPos + 1);
        $fileName  = str_replace('\\', DIRECTORY_SEPARATOR, $namespace) . DIRECTORY_
SEPARATOR;
    }
    $fileName .= str_replace('_', DIRECTORY_SEPARATOR, $className) . '.php';

    require $fileName;
}
```

A more advanced implementation of autoloader can be explored on GitHub:
http://gist.github.com/221634

5

PHP File Formatting

Overview

- All PHP files must start with **<?php**
- The PHP closing tag (**?>**) MUST BE omitted. This prevents the accidental injection of a trailing white space into server response.
- Files MUST BE encoded in **UTF-8 without BOM.**
- **Class names** MUST BE declared in **StudlyCaps**.
- **Class constants** MUST BE declared in **uppercase** with underscore ('_') as separator.
- **Class methods** MUST BE declared in **camelCase**.
- **Functions** and **variables** SHOULD BE declared in **camelCase**. The usage of underscore is allowed but discouraged.
- There can be only **one class definition per file.**

Separation of execution and logic

Each PHP file should either declare new symbols (**classes** and **functions**) without any execution of that logic outside the symbol body, or it should only contain the execution of that logic, but without its definition.
In short, PHP files can either declare **functions** and **classes**, or can just execute those **functions** and **classes** to generate output, but SHOULD NOT do both.

Here is an example code that has both the definition of logic and its execution. This is what you **MUST avoid**.

```php
<?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
}
```

A correct approach would be more like the following example.

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

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

7

Coding style

Overview

- Code MUST use 4 spaces for indenting, **not tabs**. The reason for this is to avoid problems with diffs on versioning systems.
- **Line limit** SHOULD be **120 characters.** The preferable line length is 80 characters. This SHOULD provide a better readability of the code on different screen resolutions.
- After **namespace** definition there MUST be one blank line, and there MUST be one blank line after **use** block.
- Opening and closing brackets for **class** body MUST go into a new line. The same rule applies to **class methods** and **functions**.
- Visibility MUST be declared on **all properties**.
- Opening brackets on control structures (**if, for, foreach, while**) MUST go into the same line.
- Control structure keywords MUST have one space after them, **functions** and **methods** MUST NOT.
- **Strings** MUST be concatenated using the "." operator. A space MUST always be added before and after the "." operator to improve readability.
- When declaring **array** elements, a space MUST be added after each comma (",").
- In **associative arrays** when you have key-value pairs, where each pair is defined in a new line, you SHOULD align the '=>' operator to be always in the same place, by adding white space before the operator.

Example

Here is an example that incorporates the before defined guidelines.

```php
<?php
namespace Vendor\Package;
// notice blank line
use FooInterface;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;
// notice blank line
class Foo extends Bar implements FooInterface // extend and implement go into a
single line
{ // brackets go into a new line
    private function _sampleFunction($a, $b = null)
    {// brackets go into a new line
        if ($a === $b) {// brackets go into the same line
            bar();
        } elseif ($a > $b) {// brackets go into the same line
            $foo->bar($arg1);
        } else {// brackets go into the same line
            BazClass::bar($arg2, $arg3);
        }
    }

    final public static function bar()
    {
        // method body
    }
}// brackets go into a new line
// put one blank line at the end of your PHP file
```

Please take note on spaces, position of brackets and cases used in element and symbol naming.

General

1. Files

All PHP files MUST use the Unix LF (linefeed) line ending.

All PHP files MUST end with a single blank line.

The closing **?>** tag MUST be omitted from files containing only PHP.

2. Lines

There MUST NOT be a hard limit on line length.

The soft limit on line length MUST be **120 characters**; automated style checkers MUST warn but MUST NOT error at the soft limit.

Lines SHOULD NOT be longer than **80 characters**; lines longer than that SHOULD be split into multiple subsequent lines of no more than 80 characters each.

There MUST NOT be trailing white space at the end of non-blank lines.

Blank lines MAY be added to improve readability and to indicate related blocks of code.

There MUST NOT be more than one statement per line.

3. Indenting

Code MUST use an indent of **4 spaces**, and **MUST NOT use tabs** for indenting.

4. Keywords and True/False/Null

PHP keywords MUST be in lower case.

The PHP constants **true**, **false**, and **null** MUST be in **lower case**.

5. Namespace and Use Declarations

When present, there MUST be one blank line after the **namespace** declaration.

When present, all **use** declarations MUST go after the **namespace** declaration.

There MUST be one **use** keyword per declaration.

There MUST be one blank line after the **use** block.

For example:

```
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

// ... additional PHP code ...
```

6. Classes, Properties and Methods

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

6.1 Extends and Implements

The **extends** and **implements** keywords MUST be declared on the same line as the **class** name.

The opening brackets for the **class** MUST go on its own line; the closing brackets for the **class** MUST go on the next line after the body.

Lists of **implements** MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one **interface** per line.

```php
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements
    \ArrayAccess,
    \Countable,
    \Serializable
{
    // constants, properties, methods
}
```

6.2 Properties

Visibility MUST be declared on all **properties**.

The **var** keyword MUST NOT be used to declare a **property**.

There MUST NOT be more than one **property** declared per statement.

Property names SHOULD be prefixed with a single underscore to indicate **protected** or **private** visibility.

A **property** declaration looks like the following.

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public $foo = null;
    private $_foo = null;
    protected $_boo = null;
}
```

6.3 Methods

Visibility MUST be declared on all **methods**.

Method names SHOULD be prefixed with a single underscore to indicate **protected** or **private** visibility.

Method names MUST NOT be declared with a space after the **method** name. The opening brackets MUST go on its own line, and the closing brackets MUST go on the next line following the body. There MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis.

```php
<?php
namespace Vendor\Package;

class ClassName
{
	public function myMethod()
	{
		// method body
	}

	private function _doSomethingPrivate()
	{
		// method body
	}

	protected function _doSomethingProtected()
	{
		// method body
	}
}
```

6.4 Method Arguments

In the **argument** list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

Method arguments with default values MUST go at the end of the **argument** list.

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public function foo($arg1, &$arg2, $arg3 = [])
    {
        // method body
    }
}
```

Argument lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one **argument** per line.

When the **argument** list is split across multiple lines, the closing bracket and opening bracket MUST be placed together on their own line with one space between them.

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public function aVeryLongMethodName(
        ClassTypeHint $arg1,
        &$arg2,
        array $arg3 = []
    ) {
        // method body
    }
}
```

6.5 abstract, final and static

When present, the **abstract** and **final** declarations MUST precede the visibility declaration.

When present, the **static** declaration MUST come after the visibility declaration.

```php
<?php
namespace Vendor\Package;

abstract class ClassName
{
    protected static $foo;

    abstract protected function zim();

    final public static function bar()
    {
        // method body
    }
}
```

6.6 Method and Function Calls

When making a **method** or **function** call, there MUST NOT be a space between the **method** or **function** name and the opening parenthesis, there MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis. In the **argument** list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

```
<?php
bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);
```

Argument lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one **argument** per line.

```
<?php
$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
);
```

7. Control Structures

The general style rules for control structures are as follows:

- There MUST be one space after the control structure keyword.
- There MUST NOT be a space after the opening parenthesis.
- There MUST NOT be a space before the closing parenthesis.
- There MUST be one space between the closing parenthesis and the opening bracket.
- The structure body MUST be indented once.
- The closing bracket MUST be on the next line after the body.

The body of each structure MUST be enclosed by brackets. This standardizes how the structures look, and reduces the likelihood of introducing errors as new lines get added to the body.

7.1 if, elseif, else

An **if** structure looks like the following.
Note the placement of parentheses, spaces, and brackets; and that **else** and **elseif** are on the same line as the closing brackets from the earlier body.

```
<?php
if ($expr1) {
    // if body
} elseif ($expr2) {
    // elseif body
} else {
    // else body;
}
```

The keyword **elseif** SHOULD be used instead of **else if** so that all control keywords look like single words.

7.2 switch, case

A **switch** structure looks like the following.
Note the placement of parentheses, spaces, and brackets. The **case** statement MUST be indented once from **switch**, and the **break** keyword (or other terminating keyword) MUST be indented at the same level as the case body. There MUST be a comment such as // no break when fall-through is intentional in a non-empty case body.

```php
<?php
switch ($expr) {
    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;
}
```

7.3 while, do while

A **while** statement looks like the following.
Note the placement of parentheses, spaces, and brackets.

```php
<?php
// while
while ($expr) {
    // structure body
}

// do while
do {
    // structure body;
} while ($expr);
```

7.4 for

A **for** statement looks like the following.
Note the placement of parentheses, spaces, and brackets.

```
<?php
for ($i = 0; $i < 10; $i++) {
    // for body
}
```

7.5 foreach

A **foreach** statement looks like the following.
Note the placement of parentheses, spaces, and brackets.

```
<?php
foreach ($iterable as $key => $value) {
    // foreach body
}
```

7.6 try, catch

A **try catch** block looks like the following.
Note the placement of parentheses, spaces, and brackets.

```
<?php
try {
    // try body
} catch (FirstExceptionType $e) {
    // catch body
} catch (OtherExceptionType $e) {
    // catch body
}
```

8. Closures

Closures MUST be declared with a space after the **function** keyword, and a space before and after the **use** keyword.

The opening brackets MUST go on the same line, and the closing brace MUST go on the next line following the body.

There MUST NOT be a space after the opening parenthesis of the **argument** list or **variable** list, and there MUST NOT be a space before the closing parenthesis of the **argument** list or **variable** list.

In the **argument** list and **variable** list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

Closure arguments with default values MUST go at the end of the **argument** list.

A **closure** declaration looks like the following.
Note the placement of parentheses, commas, spaces, and brackets:

```
<?php
$closureWithArgs = function ($arg1, $arg2) {
    // body
};

$closureWithArgsAndVars = function ($arg1, $arg2) use ($var1, $var2) {
    // body
};
```

Argument lists and variable lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one **argument** or **variable** per line.

When the ending list (whether or **arguments** or **variables**) is split across multiple lines, the closing parenthesis and opening brackets MUST be placed together on their own line with one space between them.

The following are examples of **closures** with and without **argument** lists and **variable** lists split across multiple lines.

```php
<?php
$longArgs_noVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) {
    // body
};

$noArgs_longVars = function () use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // body
};

$longArgs_longVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // body
};

$longArgs_shortVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use ($var1) {
    // body
};

$shortArgs_longVars = function ($arg) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // body
};
```

Note that the formatting rules also apply when the **closure** is used directly in a **function** or **method** call as an **argument**.

```php
<?php
$foo->bar(
    $arg1,
    function ($arg2) use ($var1) {
        // body
    },
    $arg3
);
```

Documenting your code

Overview

All documentation blocks (“docblocks”) must be compatible with the **phpDocumentor** format. Describing the phpDocumentor format is beyond the scope of this document. For more information, 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. Examples of such docblocks can be found below.

```php
<?php
/**
 * Webiny Framework (http://www.webiny.com/framework)
 *
 * @link      http://www.webiny.com/wf-snv for the canonical source repository
 * @copyright Copyright (c) 2009-2013 Webiny LTD. (http://www.webiny.com)
 * @license   http://www.webiny.com/framework/license
 */

namespace WF\Tools\Redirect;

/**
 * Short description.
 *
 * Long description (optional).
 *
 * @package   WF\Tools\Redirect;
 */

class Redirect
{
      // class body
}
```

Class Methods and 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** or **method** MAY throw an **exception**, use **@throws** for all known **exception** classes:

```php
<?php
/**
 * (...)
 **/
class Redirect
{
	/**
	 * Constructor.
	 * Set standard object value.
	 *
	 * @param array $value		Array that will hold some information.
	 * @throws WebinyException
	 */
	public function __construct($value)
	{

	}
}
```

E: info@webiny.com
W: http://www.webiny.com