



# Coding Standard

Code formatting and documentation to individuals and teams contributing to OpenBiz Framework.

Version 2.0  
Agus Suhartono  
2009-09-08

# Index

Index .....	1
Overview .....	3
Scope.....	3
Goals .....	3
PHP File Formatting.....	4
Indentation .....	4
Maximum Line Length.....	4
Naming Conventions.....	5
Classes .....	5
Filenames .....	5
Functions and Methods.....	5
Variables .....	6
Constants.....	7
Coding Style.....	8
PHP Code Demarcation.....	8
Strings.....	8
String Literals .....	8
String Literals Containing Apostrophes .....	8
Variable Substitution .....	9
String Concatenation .....	9
Arrays .....	9
Numerically Indexed Arrays .....	9
Associative Arrays .....	10
Classes .....	10
Class Declaration.....	10
Class Member Variables .....	11
Functions and Methods.....	11
Function and Method Declaration .....	11
Function and Method Usage.....	14
Control Statements .....	14
If/Else/Elseif .....	14

Switch .....	16
Inline Documentation .....	17
Documentation Format .....	17
Files .....	17
Classes .....	17
Functions .....	18
Implementation: .....	19
For current source code .....	19
Phase 1: (Safe) .....	19
Phase 2: (Safe with some risk) .....	19
Phase 3: (Safe with some risk) .....	19

# Overview

## Scope

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

*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 the OpenBiz Framework coding standards capture practices that have worked well on the OpenBiz project.*

Topics covered in the OpenBiz coding standards include:

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

## Goals

Status: Use 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.

# PHP File Formatting

## Indentation

Indentation should consist of 4 spaces. Tabs are not allowed.

## Maximum Line Length

The target line length is 80 characters. That is to say, OpenBiz Framework developers should strive keep each line of their code under 80 characters where possible and practical. However, longer lines are acceptable in some circumstances. The maximum length of any line of PHP code is 120 characters

# Naming Conventions

## Classes

All class name start with "OBiz\_". Class names may only contain alphanumeric characters. Underscores and numbers are permitted in class names but are discouraged in most cases.

## Filenames

For all other files, only alphanumeric characters, underscores, and the dash character ("-") are permitted. Spaces are strictly prohibited.

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

- BizSystem.php
- openbiz/bin/Expression.php
- openbiz/bin/sysheader.inc
- openbiz/bin/easy/HTMLMenu.php
- openbiz/bin/easy/HTMLTabs.php
- openbiz/bin/easy/EasyForm.php

File names must map to class names as described above.

## 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.

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

These are examples of acceptable names for functions:

```
getObjectFactory()  
getSessionContext()  
getTypeManager()
```

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

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 "public" modifier must starting with `$m_`, and 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.

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" 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.

## 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, while all words in a constant name must be separated by underscore characters.

For example, `EMBED_SUPPRESS_EMBED_EXCEPTION` is permitted but `EMBED_SUPPRESSEMBEDECEPTION` 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.

# Coding Style

## PHP Code Demarcation

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

```
<?php  
  
?>
```

Short tags are never allowed.

## 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:

```
$a = 'Example String';
```

### String Literals Containing Apostrophes

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:

```
$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.

## Variable Substitution

Variable substitution is permitted using either of these forms:

```
$greeting = "Hello $name, welcome back!";  
$greeting = "Hello {$name}, welcome back!";
```

For consistency, this form is not permitted:

```
$greeting = "Hello ${name}, welcome back!";
```

## String Concatenation

Strings must be concatenated using the "." operator. A space must always be added before and after the "." operator to improve readability:

```
$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 white space such that the "." operator is aligned under the "=" operator:

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

## Arrays

### 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:

```
$sampleArray = array(1, 2, 3, 'Zend', 'Studio');
```

It is permitted to declare multi-line 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:

```
$sampleArray = array(1, 2, 3, 'Zend', 'Studio',  
                    $a, $b, $c,  
                    56.44, $d, 500);
```

## Associative Arrays

When declaring associative arrays with the `array` construct, breaking the statement into multiple lines is encouraged. In this case, each successive line must be padded with white space such that both the keys and the values are aligned:

```
$sampleArray = array('firstKey' => 'firstValue',  
                    'secondKey' => 'secondValue');
```

## Classes

### Class Declaration

Classes must be named according to OpenBiz Framework's naming conventions.

The brace should always be written on the line underneath the class name.

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.

The following is an example of an acceptable class declaration:

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

## Class Member Variables

Member variables must be named according to OpenBiz Framework'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 favor of accessor methods (`set/get`).

## Functions and Methods

### Function and Method Declaration

Functions must be named according to the OpenBiz Framework 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 line underneath 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:

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

*NOTE:*

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

```
/**
 * Documentation Block Here
 */
class Foo
{
    /**
     * Documentation Block Here
```

```
*/  
public function bar(&$baz)  
    {  
}  
}
```

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

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

```
/**  
 * Documentation Block Here  
 */  
class Foo  
{  
    /**  
     * WRONG  
     */  
    public function bar()  
    {  
        return($this->bar);  
    }  
  
    /**  
     * RIGHT  
     */  
    public function bar()  
    {  
        return $this->bar;  
    }  
}
```

## 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:

```
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:

```
threeArguments(array(1, 2, 3), 2, 3);  
  
threeArguments(array(1, 2, 3, 'Zend', 'Studio',  
                    $a, $b, $c,  
                    56.44, $d, 500), 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 **new** 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.

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

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:

```
if ($a != 2)
{
    $a = 2;
}
else
{
    $a = 7;
}

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

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.

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

## 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 four spaces. Content under each "case" statement must be indented using an additional four spaces.

```
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.*

## Inline Documentation

### Documentation Format

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.

### Files

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
 */
```

Use directory name as package name

## 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

It is not necessary to use the "@access" tag because the access level is already known from the "public", "private", or "protected" modifier used to declare the function.

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

@throws exceptionclass [description]

## Implementation:

For a new source code:

All the conventions should be implemented

### For current source code

Applied to 3 phase

#### Phase 1: (Safe)

- Indentation
- Inline documentation
- Coding style unless name convention
- Name convention for private scope of variables and methods

#### Phase 2: (Safe with some risk)

- Name convention for method/function name
- Name convention for protected scope of variable

#### Phase 3: (Safe with some risk)

- Name convention for global scope of variable
- Name convention for class (add OBiz\_)