Programming Style Guide

This document describes the programming style and best practices used in Blesta. Programmers wishing to certify their plugin, module, or gateway, with the Marketplace must conform to these guidelines regardless of whether or not their source code is made available to end users.

Blesta primarily follows a PSR-2 coding style guide.

Table of Contents

PHP File Formatting

Tags

PHP code must always begin with the long form opening tag. If a file contains only PHP code, the end tag may optionally be omitted from the end of the file.

```
<?php

?>
```

Indentation

Tabs should always be used for indentation. We live in a world where IDEs can be configured to display tabs at variable lengths, so **configure your tab width to be represented as 4 spaces**.

Line Length

Lines should be designed such that a reader can avoid horizontal scrolling, except in cases where scrolling offers improved readability over word-wrapping. As the resolution and width of monitors has increased the standard width of 80 characters (previously the width of a punch card) no longer holds any relevance. Use your best judgement to determine where lines should wrap. Blesta uses a 120-character limit for *.php files and best judgement for *.pdt files.

EOL

Every line should terminate with a Unix-style line-feed (LF) character only. That is '\n', also known as hex value 0x0A. Do not use CRLF (Windows systems) or CR (Mac systems).

Character Encoding

All PHP files should be encoded using UTF-8, with no byte-order-mark (BOM).

Programming Style

Variables

Use logical and descriptive names for your variables. Variables used in iterating loops, such as **$i**, **$j**, **$k**, etc., are exceptions to this rule. Variables should contain only lowercase letters, underscores, and numbers, and should start with a letter. Values **true**, **false**, and **null** should always be lowercase.

Incorrect

```
$foo = 365; // not descriptive
$days_in_non_leap_year = 365; // too long
$_year_days = 365; // starts with an underscore
```

Correct

```
$year_days = 365;
$leap_year_days = 366;
```

Strings

Literals

Strings should be surrounded with single-quotes ('), except where double-quotes (") are necessary in php.

```
$str = 'This is a long string';
$chr = 'a';
$double_str = "Double-quoted string to include a new line\n";
```

Concatenation

Run-on Strings

A string whose length requires it to wrap multiple lines may be concatenated with the concatenation character (.) prepended at the beginning of the next line to be concatenated, indented accordingly. A single string with new lines within the string is also acceptable.

```
$str = 'This is a string that needs to'
    . ' wrap onto another line.';

$str2 = 'This is a string that
        spans multiple lines, but
        the new-line white-space
        is not relevant';
```

Variable Substitution

When a variable needs to be added to a string the string should be broken open and concatenated together to improve readability.

```
$name = 'Hefo Quent Esbit';
$str = 'Hello ' . $name . '! How are you?';
```

Arrays

Numerically Index Arrays

Each array element should be separated by a comma and a space character. If wrapping an array across multiple lines, subsequent lines should be indented.

Correct

```
$shapes = ['circle', 'square', 'triangle', 'hexagon'];
$colors = [
    'red',
        'pink',
    'green',
        'purple'
];
```

Associative Arrays

Associative arrays should have a space before and after the **=>** assignment operator, just like all operators should. Strings should be quoted using single quotes ('). Short arrays can contain all elements on a single line, but it's recommend to set each index on its own line.

Correct

```
$blocks = [
    'circle' => 'red',
    'square' => 'pink',
    'triangle' => 'green',
    'hexagon' => 'purple'
];
```

Multidimensional Arrays

Each array depth should be indented one level. The ending parenthesis of the array should be indented such that it aligns with the indent level of its defining index.

Correct

```
$polygons = [
    'triangles' => [
        'obtuse',
        'acute',
        'right'
    ],
    'quadrilaterals' => [
        'square',
        'rhombus',
        'rectangle'
    ]
];
```

Constants

Constants should always be capitalized.

Incorrect

```
define('myConstant', 'some value');
```

Correct

```
define('MY_CONSTANT', 'some value');
```

Statements

Blesta uses PSR-2 control structure styles. Each statement should begin on its own line. The exception to this rule is do-while.

Incorrect

```
if ($foo > $bar)
{
     echo 'foo is greater';
} else
{
    echo 'bar is greater';
}
```

Correct

```
if ($foo > $bar) {
    echo 'foo is greater';
} else {
    echo 'bar is greater';
}

$i = 5;
do {
    echo $i--;
} while ($i > 0);
```

Classes

Declaration

Class names should be CamelCased. Only a single class should appear in a file. Class file names should correlate to the class name, be lowercase, and use underscores to separate words. Class constructors should be named using the __construct() magic method.

Incorrect

my_class.php

```
<?php
// Class name starts with a lower case letter
class myClass
{
    // Constructor should be named __construct()
    public function myClass()
        {
                // Do stuff
        }
}
```

Correct

my_class.php

```
<?php
class MyClass
{
    public function __construct()
        {
                // Do stuff
        }
}
```

Member Variables

All member variables should be declared at the top of the class, before any methods are declared. Variables must always declare their visibility using **public**, **private**, or **protected**.

Incorrect

```
<?php
class MyClass {
    // Can not use var, must use public, private, or protected
    var $foo;

    public function __construct()
        {
                // Do stuff
        }

    // Must be declared at the top of the class
    private $bar = "baz";
}
```

Correct

```
<?php
class MyClass
{
    public $foo;
    private $bar = 'baz';

    public function __construct()
        {
                // Do stuff
        }
}
```

Methods

Method names should begin with a lowercase letter and be camelCased. Type hinting should be used whenever possible, but must conform to PHP 5.4 syntax (i.e. only objects, arrays, and callables may be defined using type hints).

Incorrect

```
public function CalcDistance($time, $velocity)
{
    return $time * $velocity;
}
```

Correct

```
public function calcDistance($time, $velocity)
{
    return $time * $velocity;
}
```

Comments

There are three types of comments that may be used in PHP. They are block comments (/* */), inline comments (//), and shell comments (#).

Block Comments

Block comments are used for doc commenting and commenting out large sections of code. Blesta uses PHPDoc syntax for all doc comments. Block comments may not be used as a substitute for inline comments.

Incorrect

```
/*
This is a comment.
*/
if ($foo > $bar) {
    return $bar
}
```

Correct

```
/**
 * This is a doc block comment
 *
 * @param string $bar The value
 */
function foo($bar)
{

}
```

Inline Comments

Inline comments should be used to clarify code. You are discouraged from using inline comments at the end of a line as this creates run-on lines and your comment is unlikely to be read.

Incorrect

```
/*
Calculate the circumference
*/
$c = 2 * pi() * $r;
```

Correct, but discouraged

```
$c = 2 * pi() * $r; // Calculate the circumference
```

Correct

```
// Calculate the circumference
$circumference = 2 * pi() * $r;
```

Shell Comments

These comments should only be used to identify TODO comments.

Incorrect

```
# Calculate the circumference
$c = 2 * pi() * $r;
```

Correct

```
<?php
class MyClass
{
    #
    # TODO: Finish this class.
    #
}
```