PHP Coding Styles

Code Demarcation

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

```
<?php

?>
```

Short tags are never allowed.

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 white space into the response.

Indention

Indentation should consist of 1 tab per indentation level. Spaces are not allowed.

Line Length

The target line length is 120 characters. Longer lines are acceptable as long as readability is maintained.

Line Termination

Line termination follows the Unix text file convention. Lines must end with a single linefeed (LF) character. Linefeed characters are represented as ordinal 10, or hexadecimal 0×0A.

Note: Do not use carriage returns (CR) as is the convention in Apple OS’s (0×0D) or the carriage return – linefeed combination (CRLF) as is standard for the Windows OS (0×0D, 0×0A).

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 = 'HUBzero' . ' ' . 'content management system';
```

When concatenating long 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 `users` "
     . "WHERE `name` = 'Jim' "
     . "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, 'HUBzero');
```

It is permitted to declare multi-line indexed arrays using the “array” construct. In this case, each successive line must be indented to the same level as first line and then padded with spaces such that beginning of each line is aligned:

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

Alternately, the initial array item may begin on the following line. If so, 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:

```
$sampleArray = array(
    1, 2, 3, 'HUBzero',
    $a, $b, $c,
```

```
    56.44, $d, 500,
);
```

When using this latter declaration, we encourage using a trailing comma for the last item in the array; this minimizes 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, 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');
```

Alternately, the initial array item may begin on the following line. If so, 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. For readability, the various “=>” assignment operators should be padded such that they align.

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

When using this latter declaration, we encourage using a trailing comma for the last item in the array; this minimizes the impact of adding new items on successive lines, and helps to ensure no parse errors occur due to a missing comma.

Classes

- Classes must be named according to HUBzero’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 a single tab.
- Only one class is preferred in each PHP file. Additional classes are permitted but strongly discouraged.
- Placing additional code in class files is permitted but discouraged.

The following is an example of an acceptable class declaration:

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

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

```
class SampleClass extends FooAbstract implements BarInterface
{
}
```

If as a result of such declarations, readability suffers due to line length, break the line before the “extends” and/or “implements” keywords, and pad those lines by one indentation level.

```
class SampleClass
    extends FooAbstract
    implements BarInterface
{
}
```

If the class implements multiple interfaces and the declaration covers multiple lines, break after each comma separating the interfaces, and indent the interface names such that they align.

```
class SampleClass
```

```
    implements BarInterface,
               BazInterface
{
}
```

Class Member Variables

Member variables must be named according to HUBzero'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 permitted but discouraged. Member variables should 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

Declaration

Functions must be named according to HUBzero'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 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
    }
}
```

In cases where the argument list affects readability, you may introduce line breaks. Additional arguments to the function or method must be indented one additional level beyond the function or method declaration. The following is an example of one such situation:

```
/**
* Documentation Block Here
*/
class Foo
{
    /**
     * Documentation Block Here
     */
    public function bar($arg1, $arg2, $arg3,
        $arg4, $arg5, $arg6)
    {
        // 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 additional 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, 'HUBzero',
                     $a, $b, $c,
                     56.44, $d, 500), 2, 3);

threeArguments(array(
    1, 2, 3, 'HUBzero',
    $a, $b, $c,
    56.44, $d, 500
), 2, 3);
```

Control Statements

If/Else/Elseif

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

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 line after the conditional statement. The closing brace is always written on its own line. Any content within the braces must be indented using 1 tab.

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

If the conditional statement causes the line length to affect readability 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 paren 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.

```
if (($a == $b)
    && ($b == $c)
    || (Foo::CONST == $d))
{
    $a = $d;
}
```

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

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

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

if (($a == $b)
    && ($b == $c)
    || (Foo::CONST == $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 is not permitted; all if, else if 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 one indention level. Content under each case statement must be indented using an additional indention level.

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

    case 2:
        break;

    default:
        break;
}
```

The construct default should not 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

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: [1]

All class files must contain a “file-level” docblock at the top of each file and a “class-level” docblock immediately above each class.

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:

```
/**
 * @package     hubzero-cms
 * @author      Joe Smith <joesmith@hubzero.org>
 * @copyright   Copyright 2005-2011 Purdue University. All rights rese
rved.
 * @license     http://www.gnu.org/licenses/lgpl-3.0.html LGPLv3
 *
 * Copyright 2005-2011 Purdue University. All rights reserved.
 *
 * This file is part of: The HUBzero(R) Platform for Scientific Collab
oration
 *
 * The HUBzero(R) Platform for Scientific Collaboration (HUBzero) is f
ree
 * software: you can redistribute it and/or modify it under the terms
of
 * the GNU Lesser General Public License as published by the Free Soft
ware
 * Foundation, either version 3 of the License, or (at your option) an
y
 * later version.
 *
 * HUBzero is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Lesser General Public License for more details.
```

```
 *
 * You should have received a copy of the GNU Lesser General Public Li
cense
 * along with this program.  If not, see <http://www.gnu.org/licenses/
>.
 *
 * HUBzero is a registered trademark of Purdue University.
 */
```

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)...
 *
 * @package     hubzero-cms
 * @subpackage  com_members
 * @copyright   Copyright 2005-2011 Purdue University. All rights rese
rved.
 * @license     http://www.gnu.org/licenses/lgpl-3.0.html LGPLv3
 * @version     Release: @package_version@
 * @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

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 or method may throw an exception, use @throws for all known exception classes:

```
@throws exceptionclass [description]
```

SQL Queries

SQL keywords are to be written in uppercase, while all other identifiers (which the exception of quoted text) is to be in lowercase.

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