XOOPS 3.0 Coding Standards Draft

Authors:

- Taiwen Jiang <phppp@users.sourceforge.net>

Scope

This document provides the coding standards and guidelines for developers and teams working on or with the XOOPS Project. The subjects covered are:

- PHP File Formatting
- Naming Conventions
- Coding Style
- Inline Documentation
- Errors and Exceptions

Developer Roles:

- XOOPS Developers: developers who contribute to XOOPS frameworks, including
  - Core Developers: developers who contribute to XOOPS Core SVN and whose code will be used by XOOPS application developers.
  - Framework Developers: developers who contribute to XOOPS Frameworks that could be adopted into XOOPS core in the future.
  - Front-end Developers: developers who work on XOOPS themes and module templates.
- XOOPS Application Developers: developers who build their own applications upon XOOPS platform
  - Module Developers: developers who build third-party modules using XOOPS core platform and libraries.

Goals

Good coding standards are important in any development project, particularly when multiple developers are working on the same project. Having coding standards helps to ensure that the code is of high quality, has fewer bugs, and is easily maintained.

Abstract goals we strive for:

- extreme simplicity
- tool friendliness, such as use of method signatures, constants, and patterns that support IDE tools and auto-completion of method, class, and constant names.

When considering the goals above, each situation requires an examination of the circumstances and balancing of various trade-offs.

PHP File Formatting

General

For files that contain PHP code, the closing tag ("?>") must be included where necessary.

Indentation

Use an indent of 4 spaces with no tab characters. Editors should be configured to treat tabs as four spaces in order to prevent injection of tab characters into the source code.

Maximum Line Length

The target line length is 80 characters; i.e., developers should aim keep code as close to the 80-column boundary as is practical. However, longer lines are acceptable. The maximum length of any line of PHP code is 120 characters.

Line Termination

Line termination is the standard way for Unix text files. Lines must end only with a linefeed (LF).
Linefeeds are represented as ordinal 10, or hexadecimal 0x0A.
Do not use carriage returns (CR) like Macintosh computers (0x0D).
Do not use the carriage return/linefeed combination (CRLF) as Windows computers (0x0D, 0x0A).
Lines should not contain trailing spaces. In order to facilitate this convention, most editors can be configured to strip trailing spaces, such as upon a save operation.

Naming Conventions

Overall Proposal

1. Names for all classes and functions inside XOOPS should start with Xoops;
    1. First letter for function names should always be in lowercase; in case necessary names should be separated using underscores "_" not camelCaps:
        - core functions should be started with xoops_
        - framework and library functions should be started with xoops_[framework or library identifier]_
    2. Class names should always be separated using camelCaps, which is addressed below;
2. Names for shared variables generated by XOOPS should start with $xoops following camelCaps style
    - Private (or not shared) variables are encouraged to follow the style as well
3. Third-party applications, including modules, should not start with Xoops_ but start with corresponding identifier
    - Module class names should start with [module identifer, usually directory name] following camelCaps style, e.g. NewbbPost
    - Module function names should start with [module identifer, usually directory name]_, e.g. newbb_getPostCount()
    - Module variables should start with $[module identifier], e.g. $newbbPostCount

Abstractions Used in API (Class Interfaces)

When creating an API for use by application developers, if application developers must identify abstractions using a compound name, separate the names using underscores, not camelCaps. When the developer uses a string, normalize it to lowercase. Where reasonable, add constants to support this.

Classes

System, core or framework class names should start with Xoops, like XoopsUser, XoopsCaptcha.
Module class names should start with [module identifier], like NewbbPost.

Interfaces

Interface classes must follow the same conventions as other classes (see above), but must end with "_Interface", such as: XoopsLogger_Interface

Filenames

For all files, only alphanumeric characters, underscores, and the dash character ("-") are permitted.
Spaces are prohibited.
Any file that contains any PHP code must end with the extension ".php".
File names must be in lowercase.

Directory names

For all directories, only alphanumeric characters, underscores, and the dash character ("-") are permitted. Spaces are prohibited.
Directory names must be in lowercase.

Functions and Methods

Core, system module function names should start with xoops_doSomething([...]).
Framework function names should start with xoops_[lowercase of identifier]_, like xoops_pear_doSomething([...]).
Module function names should start with [lowercase of module identifier]_, like newbb_getTopic([...]).

- Function names may only contain alphanumeric characters and underscores. Numbers are permitted in function names but are discouraged.
- Function names must always start with a lowercase letter.
- Verbosity is encouraged. Function names should be as illustrative as is practical to enhance understanding.
- For object-oriented programming, accessors for object members should always be prefixed with either "get" or "set".
- Class methods that are declared as protected or private are encouraged to follow same style as public methods although it is desired by some external frameworks to begin with a single underscore, i.e. without leading single underscore.
- Functions in the global scope, or "floating functions," are permitted but discouraged. It is recommended that these functions be wrapped in a class and declared static.
- Methods or variables declared with a "static" scope in a class generally should not be "private", but protected instead. Use "final" if the method should not be extended.

Optional Parameters

Use "NULL" as the default value instead of "FALSE", for situations like this:

```
public function foo($required, $optional = NULL)
```

when $optional does not have or need a particular default value.
However, if an optional parameter is boolean, and its logical default value should be true, or false, then using true or false is acceptable.

Variables

- System Global variables should start with $xoops, like $xoopsConfig
- Module global variables should start with $[lowercase of module identifier] following camelCaps style, like $newbbPostCounter
- Variable names for third-party frameworks/library
  - Variable names usually only contain alphanumeric characters. Underscores, numbers are permitted in variable names but are discouraged.
  - Class member variables declared as "public" may never start with an underscore.
  - For class member variables that are declared as protected or private are encouraged to follow same style as public variables, i.e. without leading single underscore.
  - Like function names, variable names must always start with a lowercase letter and follow the "camelCaps" capitalization convention.
  - Verbosity is encouraged. Variable names should always be as verbose as practical. Terse variable names such as "$i" and "$n" are discouraged for anything other than the smallest loop contexts. If a loop contains more than 20 lines of code, variables for such indices or counters need to have more descriptive names.

Constants

Variables

System constants start with XOOPS_: XOOPS_URL
Framework constants should start with XOOPS_[identifier]_: XOOPS_PEAR_CONSTANT
Module constants should start with [module identifier]_: NEWBB_CONSTANT

Language definitions

Always start with underscore: _XOOPS_LANGUAGE_CONSTANT, _NEWBB_LANGUAGE_CONSTANT

- Constants may contain both alphanumeric characters and the underscore. Numbers are permitted in constant names.
- Constant names must always have all letters capitalized.
- To enhance readability, words in constant names must be separated by underscore characters. For example, "XOOPS_EMBED_SUPPRESS_EMBED_EXCEPTION" is permitted but "XOOPS_EMBED_SUPPRESSEMBEDEXCEPTION" is not.
- Constants must be defined as class members by using the "const" construct. Defining constants in the global scope with "define" is permitted but discouraged.

Booleans and the NULL Value

Like PHP's documentation, XOOPS uses uppercase for both boolean values and the "NULL" value.

Module template names

To be added

Coding Style

PHP Code Demarcation

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

```
<?php

?>
```

Short tags are never allowed.

Strings

String Literals

Both "double quotes" and apostrophe or "single quote" are permitted for a string that is literal (contains no variable substitutions), "single quote" is encouraged:

```
$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 encouraged for SQL statements:

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

The above syntax is preferred over escaping apostrophes.

Variable Substitution

Variable substitution is encouraged using brace.

Encouraged:

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

OK but not encouraged:

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

For consistency, this form is not permitted:

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

String Concatenation

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

```
$project = 'Xoops' . ' ' . 'Project';
```

When concatenating strings with the "." operator, it is permitted 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:

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

Arrays

Numerically Indexed Arrays

Negative numbers are not permitted as array indices.
An indexed array may be started with any non-negative number, however this is discouraged and it is recommended that all arrays have a base index of 0.
When declaring indexed arrays with the array construct, a trailing space must be added after each comma delimiter to improve readability:

```
$sampleArray = array(1, 2, 3, 'XOOPS', 'Project');
```

It is also 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 aligns as shown below:

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

Associative Arrays

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

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

Classes

Class Declarations

Classes must be named by following the naming conventions.
The brace is always written on the line underneath the class name ("one true brace" form).
Every class must have a documentation block that conforms to the phpDocumentor standard.
Any code within a class must be indented the standard indent of four spaces.
It is encouraged to have only one class per PHP file.
Placing additional code in a class file is permitted but discouraged. In these files, two blank lines must separate the class from any additional PHP code in the file.
This is an example of an acceptable class declaration:

```
/**
 * Class Docblock Here
 */
class XoopsClass
{
    // entire content of class
    // must be indented four spaces
}
```

Class Member Variables

Member variables must be named by following the variable naming conventions.
Any variables declared in a class must be listed at the top of the class, prior to declaring any functions.
The var construct is not permitted. Member variables always declare their visibility by using one of the private, protected, or public constructs. Accessing member variables directly by making them public is permitted, in certain cases it is in favor of accessor methods having the set get prefixes. and

Functions and Methods

Function and Method Declaration

Functions and class methods must be named by following the naming conventions.
Methods must always declare their visibility by using one of the private, protected, or public constructs.
Following the more common usage in the PHP developer community, static methods should declare their

visibility first:

```
public      static foo()    { ... }
private     static bar()    { ... }
protected   static goo()    { ... }
```

As for classes, the opening brace for a function or method is always written on the line underneath the function or method name ("one true brace" form). There is no space between the function or method name and the opening parenthesis for the arguments.
This is an example of acceptable class method declarations:

```
/**
 * Class Docblock Here
 */
class XoopsFoo
{
    /**
     * Method Docblock Here
     */
    public function sampleMethod($a)
    {
        // entire content of function
        // must be indented four spaces
    }

    /**
     * Method Docblock Here
     */
    protected function _anotherMethod()
    {
        // ...
    }
}
```

The return value must not be enclosed in parentheses. This can hinder readability and can also break code if a function or method is later changed to return by reference.

```
function foo()
{
    // WRONG
    return($this->bar);

    // RIGHT
    return $this->bar;
}
```

The use of type hinting is encouraged where possible with respect to the component design. For example,

```
class XoopsComponent
{
    public function foo(SomeInterface $object)
    {}
    public function bar(array $options)
    {}
}
```

Function and Method Usage

Functions in the global scope are strongly discouraged.
Function arguments are separated by a single trailing space after the comma delimiter. This is an example of an acceptable function call for a function that takes three arguments:

```
threeArguments(1, 2, 3);
```

Call-time pass by-reference is prohibited. Pass-by-reference is permitted in the function declaration only. Arguments to be passed by reference must be defined in the function declaration.
For functions whose arguments permit arrays, the function call may include the "array" construct and can be split into multiple lines to improve readability. In these cases, the standards for writing arrays still apply:

```
threeArguments(array(1, 2, 3), 2, 3);

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

Control Statements

If / Else / Elseif

Control statements based on the "if", "else", and "elseif" constructs must have a single space before the opening parenthesis of the conditional, and a single space between the closing parenthesis and opening brace.
Within the conditional statements between the parentheses, operators must be separated by spaces for readability. Inner parentheses are encouraged to improve logical grouping of larger conditionals.
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 four spaces.

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

For "if" statements that include "elseif" or "else", the formatting must be as in these examples:

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

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

PHP allows for these statements to be written without braces in some circumstances. The coding standard makes no differentiation and all "if", "elseif", or "else" statements must use braces.

Switch

Control statements written with the "switch" construct must have a single space before the opening parenthesis of the conditional statement, and also a single space between the closing parenthesis and the opening brace.
All "case" items within the "switch" statement must not be indented. Content under each "case" statement must be indented an four spaces.

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

case 2:
   break;

default:
   break;
}
```

The construct "default" may never be omitted from a "switch" statement.

Inline Documentation

Documentation Format

All documentation blocks ("docblocks") must be compatible with the phpDocumentor format. For more information, visit http://phpdoc.org.
All source code file written for the XOOPS platform or that operates with the platform must contain a "file-level" docblock at the top of each file and a "class-level" docblock immediately above each class.
The sharp, '#', character should not be used to start comments.w

Files

Every file that contains PHP code must have a header block 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
 *
 * You may not change or alter any portion of this comment or credits
 * of supporting developers from this source code or any supporting source code
 * which is considered copyrighted (c) material of the original comment or credit authors.
 *
 * @copyright   The XOOPS Project http://sourceforge.net/projects/xoops/
 * @license       http://www.fsf.org/copyleft/gpl.html GNU public license
 * @author        Author  Name <author email, or website>
 * @version       $Id$
 * @since         File available since Release 3.0.0
 */
```

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   The XOOPS Project http://sourceforge.net/projects/xoops/
 * @license       http://www.fsf.org/copyleft/gpl.html GNU public license
 * @author        Author  Name <author email, or website>
 * @version       $Id$
 * @since         File available since Release 3.0.0
 */
```

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

```
/**
 * Does something interesting
 *
 * @param   Place      $where Where something interesting takes place
 * @param   integer    $repeat How many times something interesting should happen
 * @return  Status
 */
public function xoops_doSomethingInteresting(Place $where, $repeat = 1)
{
   // implementation...
}
```

Require / Include

If a component uses another component, then the using component is responsible for loading the other component. If the use is conditional, then the loading should also be conditional.
If the file(s) for the other component should always load successfully, regardless of input, then use PHP's `require_once` statement.
Use `XoopsLoad::load()` to load configured classes.
The `include`, `include_once`, `require`, and `require_once` statements should not use parentheses.

Errors and Exceptions

The XOOPS Project codebase must be `E_STRICT` compliant. XOOPS code should not emit PHP warning

(E_WARNING, E_USER_WARNING), notice (E_NOTICE, E_USER_NOTICE), or strict (E_STRICT) messages when error_reporting is set to E_ALL | E_STRICT.
See http://www.php.net/errorfunc for information on E_STRICT.

XOOPS code should not emit PHP errors, if it is reasonably possible. Instead, log each error with meaningful messages using trigger_error function then XOOPS will use custom error handler for subsequent processing.

Terms:

1. camelCaps - When a string consists of more than one word, the first letter of each new word must be capitalized. This is commonly called the "camelCaps" method.

Starting from Zend framework PHP Coding Standard (draft)
Original doc at Google Docs