**PHP Coding Standards**

**Indenting and Line Length**

Use an indent of 4 spaces, with no tabs. This helps to avoid problems with diffs, patches, SVN history and annotations.

For Emacs you should set indent-tabs-mode to nil. Here is an example mode hook that will set up Emacs (ensure that it is called when you are editing PHP files):

(defun pear/php-mode-init()

"Set some buffer-local variables."

(setq case-fold-search t)

(c-set-offset 'arglist-intro '+)

(c-set-offset 'arglist-close '0)

)

(add-hook 'php-mode-hook 'pear/php-mode-init)

Here are Vim rules for the same thing:

set expandtab

set shiftwidth=4

set softtabstop=4

set tabstop=4

It is recommended to keep lines at approximately 75-85 characters long for better code readability.

**Control Structures**

These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated of them:

<?php  
if ((condition1) || (condition2)) {  
    action1;  
} elseif ((condition3) && (condition4)) {  
    action2;  
} else {  
    defaultaction;  
}  
?>

Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

You are strongly encouraged to always use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

For switch statements:

<?php  
switch (condition) {  
case 1:  
    action1;  
    break;  
  
case 2:  
    action2;  
    break;  
  
default:  
    defaultaction;  
    break;  
}  
?>

**Function Calls**

Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

<?php  
$var = foo($bar, $baz, $quux);  
?>

As displayed above, there should be one space on either side of an equals sign used to assign the return value of a function to a variable. In the case of a block of related assignments, more space may be inserted to promote readability:

<?php  
$short         = foo($bar);  
$long\_variable = foo($baz);  
?>

To support readability, parameters in subsequent calls to the same function/method may be aligned by parameter name:

<?php  
  
$this->callSomeFunction('param1',     'second',        true);  
$this->callSomeFunction('parameter2', 'third',         false);  
$this->callSomeFunction('3',          'verrrrrrylong', true);  
?>

**Class Definitions**

Class declarations have their opening brace on a new line:

<?php  
class Foo\_Bar  
{  
  
    //... code goes here  
  
}  
?>

**Function Definitions**

Function declarations follow the "K&R style":

<?php  
function fooFunction($arg1, $arg2 = '')  
{  
    if (condition) {  
        statement;  
    }  
    return $val;  
}  
?>

Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate. Here is a slightly longer example:

<?php  
function connect(&$dsn, $persistent = false)  
{  
    if (is\_array($dsn)) {  
        $dsninfo = &$dsn;  
    } else {  
        $dsninfo = DB::parseDSN($dsn);  
    }  
  
    if (!$dsninfo || !$dsninfo['phptype']) {  
        return $this->raiseError();  
    }  
  
    return true;  
}  
?>

**Arrays**

Assignments in arrays may be aligned. When splitting array definitions onto several lines, the last value may also have a trailing comma. This is valid PHP syntax and helps to keep code diffs minimal:

<?php  
  
$some\_array = array(  
    'foo'  => 'bar',  
    'spam' => 'ham',  
);  
?>

**Header Comment Blocks**

All source code files shall contain a "page-level" docblock at the top of each file and a "class-level" docblock immediately above each class. Below are examples of such docblocks.

<?php  
  
/\* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: \*/  
  
/\*\*  
 \* Short description for file  
 \*  
 \* Long description for file (if any)...  
 \*  
 \* PHP version 5  
 \*  
 \* LICENSE: This source file is subject to version 3.01 of the PHP license  
 \* that is available through the world-wide-web at the following URI:  
 \* http://www.php.net/license/3\_01.txt.  If you did not receive a copy of  
 \* the PHP License and are unable to obtain it through the web, please  
 \* send a note to license@php.net so we can mail you a copy immediately.  
 \*  
 \* @category   CategoryName  
 \* @package    PackageName  
 \* @author     Original Author <author@example.com>  
 \* @author     Another Author <another@example.com>  
 \* @copyright  1997-2005 The PHP Group  
 \* @license    http://www.php.net/license/3\_01.txt  PHP License 3.01  
 \* @version    SVN: $Id$  
 \* @link       http://pear.php.net/package/PackageName  
 \* @see        NetOther, Net\_Sample::Net\_Sample()  
 \* @since      File available since Release 1.2.0  
 \* @deprecated File deprecated in Release 2.0.0  
 \*/  
  
/\*  
\* Place includes, constant defines and $\_GLOBAL settings here.  
\* Make sure they have appropriate docblocks to avoid phpDocumentor  
\* construing they are documented by the page-level docblock.  
\*/  
  
/\*\*  
 \* Short description for class  
 \*  
 \* Long description for class (if any)...  
 \*  
 \* @category   CategoryName  
 \* @package    PackageName  
 \* @author     Original Author <author@example.com>  
 \* @author     Another Author <another@example.com>  
 \* @copyright  1997-2005 The PHP Group  
 \* @license    http://www.php.net/license/3\_01.txt  PHP License 3.01  
 \* @version    Release: @package\_version@  
 \* @link       http://pear.php.net/package/PackageName  
 \* @see        NetOther, Net\_Sample::Net\_Sample()  
 \* @since      Class available since Release 1.2.0  
 \* @deprecated Class deprecated in Release 2.0.0  
 \*/  
class Foo\_Bar  
{  
}  
  
?>

C style comments (/\* \*/) and standard C++ comments (//) are both fine. Use of Perl/shell style comments (#) is discouraged.

**Including Code**

Anywhere you are unconditionally including a class file, use **require\_once**. Anywhere you are conditionally including a class file (for example, factory methods), use **include\_once**. Either of these will ensure that class files are included only once. They share the same file list, so you don't need to worry about mixing them - a file included with **require\_once** will not be included again by **include\_once**.

**include\_once** and **require\_once** are statements, not functions. Parentheses should not surround the subject filename.

[Comments(Previous)](https://pear.php.net/manual/en/standards.comments.php)[PHP Code Tags(Next)](https://pear.php.net/manual/en/standards.tags.php)

Last updated: Sat, 16 Feb 2019 — [Download Documentation](https://pear.php.net/manual/)

# PHP Code Tags

**Always** use <?php ?> to delimit PHP code, not the <? ?> shorthand. It is also the most portable way to include PHP code on differing operating systems and setups.

# Naming Conventions

## Global Variables and Functions

If your package needs to define global variables, their names should start with a single underscore followed by the package name and another underscore.

Global functions should be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps"). In addition, they should have the package name as a prefix, to avoid name collisions between packages. The initial letter of the name (after the prefix) is lowercase, and each letter that starts a new "word" is capitalized. An example:

|  |
| --- |
| XML\_RPC\_serializeData() |

## Classes

Classes should be given descriptive names. Avoid using abbreviations where possible. Class names should always begin with an uppercase letter. Examples of good class names are:

|  |  |  |
| --- | --- | --- |
| Log | Net\_Finger | HTML\_Upload\_Error |

## Class Variables and Methods

Class variables (a.k.a properties) and methods should be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps"). Some examples (these would be "public" members):

|  |  |  |  |
| --- | --- | --- | --- |
| $counter | connect() | getData() | buildSomeWidget() |

Private class members are preceded by a single underscore. For example:

|  |  |  |
| --- | --- | --- |
| $\_status | \_sort() | \_initTree() |

The following applies to PHP5.

Protected class members are not preceded by a single underscore. For example:

|  |  |
| --- | --- |
| protected $somevar | protected function init()e() |

## Constants

Constants should always be all-uppercase, with underscores to separate words. Prefix constant names with the uppercased name of the class/package they are used in. Some examples:

|  |  |
| --- | --- |
| DB\_DATASOURCENAME | SERVICES\_AMAZON\_S3NSEKEY |

The true, false and null constants are excepted from the all-uppercase rule, and must always be lowercase.