



The Handbook

Codex

[http://codex.wordpress.org/WordPress Coding Standards](http://codex.wordpress.org/WordPress_Coding_Standards)

Version Date

9 August 2005

WordPress Coding Standards

Some legacy parts of the WordPress code structure for PHP markup are inconsistent in their style. WordPress is working to gradually improve this by helping users maintain a consistent style so the code can remain clean and easy to read at a glance.

Keep the following points in mind when writing code for WordPress, whether for core programming code, [Plugins](#), or [WordPress Themes](#). The guidelines are similar to [Pear standards](#) (<http://pear.php.net/manual/en/standards.php>) in many ways, but differs in some key respects.

Single and double quotes

When possible, single and double quotes should be used when appropriate. If you're not evaluating anything in the string, use single quotes. You should almost never have to escape HTML quotes in a string, because you can just alternate your quoting style, like so:

```
echo "<a href='$link' title='$linktitle'>$linkname</a>";
echo '<a href="/static/link" title="Yeah yeah!">Link name</a>';
```

The only exception to this is JavaScript, which sometimes requires double or single quotes. Text that goes into attributes should be run through Texturize so that single or double quotes do not end the attribute value and invalidate the XHTML.

Indentation

Your indentation should always reflect logical structure. Use **real tabs** and **not spaces**, as this allows the most flexibility across clients.

Brace Style

Brace styles, when used correctly, makes for easier reading. An example from the Pear site:

```
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
} // end blah
```

Furthermore if you have a really long block put a short comment at the end so people can tell at glance what that ending brace ends. Don't clutter the code though, only do this if the logic block is longer than about 35 rows.

include_once vs. require_once

Learn the difference between

[include_once](#) (<http://us3.php.net/manual/en/function.include-once.php>) and [require_once](#) (<http://us3.php.net/manual/en/function.require-once.php>), and use each as appropriate.

Regular expressions

Perl compatible regular expressions (PCRE, `preg_` functions) should be used in preference to their POSIX counterparts.

No Shorthand PHP

Never use shorthand PHP start tags. Always use `<?php ... ?>`.

Spaces After Comma

Always put spaces after commas and on both sides of logical operators, like `=`, `==`, `!=`, etc.

Formatting SQL statements

When formatting SQL statements you may break it into several lines and indent if it is sufficiently complex to warrant it. Most statements work well as one line though. Always capitalize the SQLy parts of the statement.

Variables, functions, and operators

If you don't use a variable more than once, don't create it. This includes queries. Always use the [wpdb Class](#) of functions when interacting with the database.

[Ternary](#) operators are fine, but always have them test for true, not false. Otherwise it just gets confusing. Good example:

```
$alert = ('1.5.1' == $version) ? 'This version is 1.5.1' : 'This version is NOT 1.5.1';
```

Another important point in the above example, when doing logical comparisons always put the variable on the right side, like above. If you forget an equal sign it'll throw a parse error instead of just evaluating true and executing the statement. It really takes no extra time to do, so if this saves one bug it's worth it.