Coding Conventions and Guidelines

March 2025

Contents

Chapter 1

General Guidelines

1.1 Purpose

This document provides a comprehensive set of coding conventions and guidelines for writing clean, readable, maintainable, and standardized PHP code. These conventions are derived from widely accepted standards such as PSR-12, WordPress Coding Standards, Drupal Coding Standards, PEAR Coding Standards and other industry best practices. Adhering to these guidelines ensures consistency across projects and improves collaboration among developers.

1.2 File Structure

- **File Naming:** Use lowercase with underscores, e.g., `user_profile.php` for class `UserProfile`.
- **Tags:** Use `<?php` and omit closing `?>` for PHP-only files to avoid whitespace issues.
- **Namespaces:** Declare at the top, followed by sorted use statements, e.g., `namespace MyProject;` use `MyClass`;.
- **Declare Statements:** Include `declare(strict_types=1);` for strict typing, e.g., after the opening tag.

1.3 Coding Style

- **Indentation:** Use 4 spaces, never tabs, for consistent alignment.
- **Line Length:** Aim for under 120 characters, split at 80 for readability.
- **Whitespace:** No trailing spaces, one space after commas, spaces around binary operators (e.g., **a + b**).
- **Braces:** Always use braces, opening on the same line, closing aligned, e.g., `if (true) { … }`.
- **Quotes:** Single quotes for static strings, double for variables, e.g., ‘`static`’ vs. “`$var`”.

1.4 Naming and Syntax

- **Naming:** Variables can be camelCase (`$myVar`) or snake_case (`$my_var`), classes in PascalCase (`MyClass`), constants in uppercase (`MY_CONST`).
- **Syntax:** Use `elseif`, prefer `===` over `==`, and format arrays with trailing commas in multi-line, e.g., `array(‘key’ => ‘value’,)`.

1.5 Documentation

- Use PHPDoc for comments, e.g., `/** @param int $a Description */`, to document parameters, returns, and more.

Chapter 2

Detailed Guidelines

2.1 File Structure

The following conventions ensure consistency:

2.1.1 File Naming

Files should be named in lowercase with words separated by underscores. For classes, the file name should match the class name, aligning with PSR-4 autoloading conventions. For example, a class `UserProfile` should reside in `user_profile.php`. This practice is supported by Drupal and WordPress standards for clarity.

2.1.2 Opening and Closing Tags

Use full PHP tags `<?php` for all files. Omit the closing `?>` tag for files containing only PHP code to prevent accidental whitespace output, a recommendation from PSR-12 and Drupal (since Drupal 4.7+). This avoids issues like unwanted newlines in output.

2.1.3 Namespace and Use Statements

Declare namespaces at the top of the file, followed by use statements for importing classes, functions, or constants. Use statements should be grouped (e.g., classes, then functions, then constants) and sorted alphabetically for readability. PSR-12 specifies no leading backslash for imports, and compound namespaces deeper than two levels are not allowed. Example:

```
<?php
namespace MyProject\SubNamespace;
use MyProject\AnotherClass;
use function MyProject\someFunction;
```

2.1.4 Declare Statements

If using strict types (recommended for new projects), include `declare(strict_types=1);` right after the opening tag or after the file-level docblock. This enhances type safety, as per PSR-12 and Drupal guidelines. Example:

```
<?php
```

```
declare(strict_types=1);
// Code follows
```

2.2 Coding Style

Coding style focuses on visual layout, impacting readability and maintainability. The following rules are derived from PSR-12, with supplements from WordPress and Drupal:

2.2.1 Indentation

Use 4 spaces for each level of indentation, as mandated by PSR-12. Do not use tabs, ensuring consistency across editors. This contrasts with Drupal's 2-space indent, but PSR-12 is preferred for general use. Example:

```
function myFunction() {
    if (true) {
        // Code here
    }
}
```

2.2.2 Line Length

PSR-12 recommends no hard limit, but a soft limit of 120 characters, with lines over 80 characters split into multiple lines of ≤80 characters each. Drupal and WordPress also suggest 80 characters as a general limit, balancing readability and terminal compatibility. Example of splitting:

```
$longVariableName = 'This is a very long string that should be split for readability';
```

2.2.3 Whitespace

Ensure no trailing whitespace at the end of lines, as per PSR-12 and WordPress. Use one space after commas in argument lists, spaces around binary operators (e.g., `a + b`), and no spaces around unary operators (e.g., `++i`). Example:

```
$result = $a + $b; // Spaces around binary
$i++; // No space around unary
```

2.2.4 Braces

Use braces for all control structures, even single-line statements, as per PSR-12 and WordPress. The opening brace goes on the same line as the control structure, and the closing brace on its own line, aligned with the start. Example:

```
if ($condition) {
    // Code
} else {
    // Else code
}
```

2.2.5 Quotes

Use single quotes for strings without variables or escape sequences, and double quotes when variables need interpolation, as recommended by WordPress and PSR-12. This reduces parsing overhead and improves readability. Example:

```
$static = ‘Hello’; // Single quotes
$dynamic = “Hello, $name”; // Double quotes
```

2.3 Naming Conventions

Naming conventions ensure clarity and consistency in identifiers. These rules vary slightly across standards, but the following are widely accepted:

2.3.1 Variables

Use camelCase (e.g., `$myVariable`) or snake_case (e.g., `$my_variable`), with consistency within the project. WordPress prefers snake_case for variables, while PSR-12 is agnostic. Example:

```
$userName = ‘John’; // camelCase
$user_name = ‘John’; // snake_case
```

2.3.2 Functions

Use camelCase (e.g., `myFunction()`) or snake_case (e.g., `my_function()`), starting with a lowercase letter. WordPress uses snake_case, while PSR-12 allows flexibility. Example:

```
function calculateTotal() {} // camelCase
function calculate_total() {} // snake_case
```

2.3.3 Classes

Use PascalCase (e.g., `MyClassName`), as per PSR-12 and Drupal. This is standard for class names to distinguish them from functions. Example:

```
class UserProfile {}
```

2.3.4 Constants

Use uppercase letters with underscores (e.g., `MY_CONSTANT`), as per PSR-12 and WordPress. This is universal for distinguishing constants. Example:

```
const MAX_USERS = 100;
```

2.3.5 Files

Lowercase with underscores, matching the class name if applicable, aligning with PSR-4 and Drupal practices. Example: `user_profile.php` for class `UserProfile`.

2.4 Syntax

Syntax rules ensure code is clear and reduces errors. The following are based on PSR-12, with additions from WordPress and Drupal:

2.4.1 Control Structures

Use `elseif` instead of `else if`, as per PSR-12 and WordPress, for readability. Ensure each case in a switch statement has a break, unless falling through is intentional. Always use curly braces for blocks. Example:

```
if ($condition) {
    // Code
} elseif ($anotherCondition) {
    // Another code
} else {
    // Else code
}
```

2.4.2 Operators

Prefer `===` and `!==` for comparisons to avoid type juggling, as recommended by WordPress and PSR-12. Use the ternary operator sparingly; prefer if-else for complex conditions for readability. Example:

```
$result = ($value === true) ? 'Yes' : 'No'; // Ternary
if ($value === true) {
    $result = 'Yes';
} else {
    $result = 'No';
} // Preferred for clarity
```

2.4.3 Arrays

Use the long array syntax `array()` for compatibility, though short syntax `[]` is acceptable in PHP 5.4+, per PSR-12. In multi-line arrays, place each element on its own line with a trailing comma, as per Drupal and WordPress. Example:

```
$array = array(
    'key1' => 'value1',
    'key2' => 'value2',
);
```

2.4.4 Strings

Use single quotes for static strings, double quotes for strings with variables, and concatenate with `.`, with spaces around the operator, as per WordPress and PSR-12. Example:

```
$greeting = 'Hello'; // Single quotes
$name = "John"; // Double quotes
$fullGreeting = $greeting . ', ' . $name; // Concatenation
```

2.4.5 Functions and Methods

No space between the function/method name and the opening parenthesis, and one space after commas in argument lists, as per PSR-12. Example:

```
function myFunction($arg1, $arg2) {
    // Code
}
```

2.5 Documentation

Documentation is vital for maintaining and understanding code. The following practices are based on PHPDoc standards, supported by Drupal and WordPress:

2.5.1 Docblocks

Use PHPDoc style for docblocks, including tags like `@param`, `@return`, `@throws`, as necessary. For classes, include `@package`, `@author`, etc., if applicable. This enhances code clarity and supports tools like IDEs. Example:

```
/**
 * Calculates the total of two numbers.
 *
 * @param int $a The first number.
 * @param int $b The second number.
 * @return int The sum of $a and $b.
 */
function calculateTotal(int $a, int $b): int {
    return $a + $b;
}
```

References

- PSR-12 Extended Coding Style Guide
- WordPress PHP Coding Standards
- Drupal PHP Coding Standards
- PEAR Coding Standards