

Useful Links

Official Site <http://www.php-fig.org>

Accepted Standards

PSR-0 Autoloading Standard

<http://www.php-fig.org/psr/psr-0/>

PSR-1 Basic Coding Standard

<http://www.php-fig.org/psr/psr-1/>

PSR-2 Coding Style Guide

<http://www.php-fig.org/psr/psr-2/>

PSR-3 Logger Interface

<http://www.php-fig.org/psr/psr-3/>

PSR-4 Improved Autoloading

<http://www.php-fig.org/psr/psr-4/>

PSR-0 - Overview

A fully-qualified namespace and class must have the following structure `<Vendor Name>\(<Namespace>\)*<Class Name>`

Each namespace must have a top-level namespace ("Vendor Name").

Each namespace can have as many sub-namespaces as it wishes.

Each namespace separator is converted to a `DIRECTORY_SEPARATOR` when loading from the file system.

Each `_` character in the CLASS NAME is converted to a `DIRECTORY_SEPARATOR`. The `_` character has no special meaning in the namespace.

The fully-qualified namespace and class is suffixed with `.php` when loading from the file system.

Alphabetic characters in vendor names, namespaces, and class names may be of any combination of lower case and upper case.

PSR-1 - Overview

Files **MUST** use only `<?php` and `<?=>` tags.

Files **MUST** use only UTF-8 without BOM for PHP code.

Files **SHOULD** either declare symbols (classes, functions, constants, etc.) or cause side-effects (e.g. generate output, change `.ini` settings, etc.) but **SHOULD NOT** do both.

Namespaces and classes **MUST** follow PSR-0.

Class names **MUST** be declared in StudlyCaps.

Class constants **MUST** be declared in all upper case with underscore separators.

Method names **MUST** be declared in camelCase.

PSR-2 - Overview

Code **MUST** follow PSR-1.

Code **MUST** use 4 spaces for indenting, not tabs.

There **MUST NOT** be a hard limit on line length; the soft limit **MUST** be 120 characters; lines **SHOULD** be 80 characters or less.

There **MUST** be one blank line after the namespace declaration, and there **MUST** be one blank line after the block of use declarations.

Opening braces for classes **MUST** go on the next line, and closing braces **MUST** go on the next line after the body.

Opening braces for methods **MUST** go on the next line, and closing braces **MUST** go on the next line after the body.

Visibility **MUST** be declared on all properties and methods; `abstract` and `final` **MUST** be declared before the visibility; `static` **MUST** be declared after the visibility.

Control structure keywords **MUST** have one space after them; method and function calls **MUST NOT**.

PSR-2 - Overview (cont)

Opening braces for control structures **MUST** go on the same line, and closing braces **MUST** go on the next line after the body.

Opening parentheses for control structures **MUST NOT** have a space after them, and closing parentheses for control structures **MUST NOT** have a space before.

PSR-2 - General

Code **MUST** follow all rules outlined in PSR-1.

All PHP files **MUST** use the Unix LF (linefeed) line ending.

All PHP files **MUST** end with a single blank line.

The closing `?>` tag **MUST** be omitted from files containing only PHP.

There **MUST NOT** be a hard limit on line length.

The soft limit on line length **MUST** be 120 characters; automated style checkers **MUST** warn but **MUST NOT** error at the soft limit.

Lines **SHOULD NOT** be longer than 80 characters; lines longer than that **SHOULD** be split into multiple subsequent lines of no more than 80 characters each.

There **MUST NOT** be trailing whitespace at the end of non-blank lines.

Blank lines **MAY** be added to improve readability and to indicate related blocks of code.

There **MUST NOT** be more than one statement per line.

Code **MUST** use an indent of 4 spaces, and **MUST NOT** use tabs for indenting.

PHP keywords **MUST** be in lower case.

The PHP constants `true`, `false`, and `null` **MUST** be in lower case.



By **Dave Child** (DaveChild)
cheatography.com/davechild/
www.addedbytes.com

Published 21st February, 2014.
 Last updated 1st June, 2014.
 Page 1 of 4.

Sponsored by **Readability-Score.com**
 Measure your website readability!
<https://readability-score.com>

PSR-2 - Namespace and Use Declarations

When present, there **MUST** be one blank line after the namespace declaration.

When present, all use declarations **MUST** go after the namespace declaration.

There **MUST** be one use keyword per declaration.

There **MUST** be one blank line after the use block.

PSR-2 - Classes, Properties, and Methods

The extends and implements keywords **MUST** be declared on the same line as the class name.

The opening brace for the class **MUST** go on its own line; the closing brace for the class **MUST** go on the next line after the body.

Lists of implements **MAY** be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list **MUST** be on the next line, and there **MUST** be only one interface per line.

Visibility **MUST** be declared on all properties.

The var keyword **MUST NOT** be used to declare a property.

There **MUST NOT** be more than one property declared per statement.

Property names **SHOULD NOT** be prefixed with a single underscore to indicate protected or private visibility.

Visibility **MUST** be declared on all methods.

Method names **SHOULD NOT** be prefixed with a single underscore to indicate protected or private visibility.

Method names **MUST NOT** be declared with a space after the method name.

The opening brace of a method **MUST** go on its own line, and the closing brace **MUST** go on the next line following the body.

PSR-2 - Classes, Properties, and Methods (cont)

There **MUST NOT** be a space after the opening parenthesis of a method, and there **MUST NOT** be a space before the closing parenthesis.

In the argument list, there **MUST NOT** be a space before each comma, and there **MUST** be one space after each comma.

Method arguments with default values **MUST** go at the end of the argument list.

Argument lists **MAY** be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list **MUST** be on the next line, and there **MUST** be only one argument per line.

When the argument list is split across multiple lines, the closing parenthesis and opening brace **MUST** be placed together on their own line with one space between them.

When present, the abstract and final declarations **MUST** precede the visibility declaration.

When present, the static declaration **MUST** come after the visibility declaration.

When making a method or function call, there **MUST NOT** be a space between the method or function name and the opening parenthesis, there **MUST NOT** be a space after the opening parenthesis, and there **MUST NOT** be a space before the closing parenthesis.

In the argument list, there **MUST NOT** be a space before each comma, and there **MUST** be one space after each comma.

PSR-2 - Control Structures

There **MUST** be one space after the control structure keyword

There **MUST NOT** be a space after the opening parenthesis

There **MUST NOT** be a space before the closing parenthesis

There **MUST** be one space between the closing parenthesis and the opening brace

The structure body **MUST** be indented once

The closing brace **MUST** be on the next line after the body

The body of each structure **MUST** be enclosed by braces.

The keyword elseif **SHOULD** be used instead of else if.

The case statement **MUST** be indented once from switch, and the break keyword (or other terminating keyword) **MUST** be indented at the same level as the case body.

There **MUST** be a comment such as // no break when fall-through is intentional in a non-empty case body.

PSR-2 - Closures

Closures **MUST** be declared with a space after the function keyword, and a space before and after the use keyword.

The opening brace **MUST** go on the same line, and the closing brace **MUST** go on the next line following the body.

There **MUST NOT** be a space after the opening parenthesis of the argument list or variable list, and there **MUST NOT** be a space before the closing parenthesis of the argument list or variable list.



By **Dave Child** (DaveChild)
cheatography.com/davechild/
www.addedbytes.com

Published 21st February, 2014.
Last updated 1st June, 2014.
Page 2 of 4.

Sponsored by **Readability-Score.com**
Measure your website readability!
<https://readability-score.com>

PSR-2 - Closures (cont)

In the argument list and variable list, there **MUST NOT** be a space before each comma, and there **MUST** be one space after each comma.

Closure arguments with default values **MUST** go at the end of the argument list.

Argument lists and variable lists **MAY** be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list **MUST** be on the next line, and there **MUST** be only one argument or variable per line.

When the ending list (whether or arguments or variables) is split across multiple lines, the closing parenthesis and opening brace **MUST** be placed together on their own line with one space between them.

PSR-3 - Log Levels (RFC 5424)

debug	error
info	critical
notice	alert
warning	emergency

PSR-3 - Basics

The `LoggerInterface` exposes eight methods matching log level names (see Log Levels block)

A ninth method, `log`, accepts a log level as first argument. Calling this method with one of the log level constants **MUST** have the same result as calling the level-specific method.

Calling the log method with a level not defined by this specification **MUST** throw a `Psr\Log\InvalidArgumentException` if the implementation does not know about the level.

Users **SHOULD NOT** use a custom level without knowing for sure the current implementation supports it.

PSR-3 - Message

Every method accepts a string as the message, or an object with a `__toString()` method.

The message **MAY** contain placeholders which implementors **MAY** replace with values from the context array.

Placeholder names **MUST** correspond to keys in the context array.

Placeholder names **MUST** be delimited with a single opening brace `{` and a single closing brace `}`. There **MUST NOT** be any whitespace between the delimiters and the placeholder name.

Placeholder names **SHOULD** be composed only of the characters A-Z, a-z, 0-9, underscore `_`, and period `..`. The use of other characters is reserved for future modifications of the placeholders specification.

Implementors **MAY** use placeholders to implement various escaping strategies and translate logs for display. Users **SHOULD NOT** pre-escape placeholder values since they can not know in which context the data will be displayed.

PSR-3 - Context

Every method accepts an array as context data. This is meant to hold any extraneous information that does not fit well in a string. The array can contain anything.

Implementors **MUST** ensure they treat context data with as much lenience as possible.

A given value in the context **MUST NOT** throw an exception nor raise any php error, warning or notice.

If an Exception object is passed in the context data, it **MUST** be in the 'exception' key.

PSR-3 - Context (cont)

Logging exceptions is a common pattern and this allows implementors to extract a stack trace from the exception when the log backend supports it.

Implementors **MUST** still verify that the 'exception' key is actually an Exception before using it as such, as it **MAY** contain anything.

PSR-3 - Helper Classes and Interfaces

The `Psr\Log\AbstractLogger` class lets you implement the `LoggerInterface` very easily by extending it and implementing the generic log method. The other eight methods are forwarding the message and context to it.

The `Psr\Log\LoggerTrait` only requires you to implement the generic log method. Note that since traits can not implement interfaces, in this case you still have to implement `LoggerInterface`.

The `Psr\Log\NullLogger` is provided together with the interface. It **MAY** be used by users of the interface to provide a fall-back "black hole" implementation if no logger is given to them. However conditional logging may be a better approach if context data creation is expensive.

The `Psr\Log\LoggerAwareInterface` only contains a `setLogger(LoggerInterface $logger)` method and can be used by frameworks to auto-wire arbitrary instances with a logger.

The `Psr\Log\LoggerAwareTrait` trait can be used to implement the equivalent interface easily in any class. It gives you access to `$this->logger`.

The `Psr\Log\LogLevel` class holds constants for the eight log levels.



By **Dave Child** (DaveChild)
cheatography.com/davechild/
www.addedbytes.com

Published 21st February, 2014.
 Last updated 1st June, 2014.
 Page 3 of 4.

Sponsored by **Readability-Score.com**
 Measure your website readability!
<https://readability-score.com>

PSR-4 Specification

A fully qualified class name has the following form: \
<NamespaceName>(\<SubNamespaceNames>)*\
<ClassName>

The fully qualified class name **MUST** have a top-level namespace name, also known as a "vendor namespace".

The fully qualified class name **MAY** have one or more sub-namespace names.

The fully qualified class name **MUST** have a terminating class name.

Underscores have no special meaning in any portion of the fully qualified class name.

Alphabetic characters in the fully qualified class name **MAY** be any combination of lower case and upper case.

All class names **MUST** be referenced in a case-sensitive fashion.

A contiguous series of one or more leading namespace and sub-namespace names, not including the leading namespace separator, in the fully qualified class name (a "namespace prefix") corresponds to at least one "base directory".

The contiguous sub-namespace names after the "namespace prefix" correspond to a subdirectory within a "base directory", in which the namespace separators represent directory separators. The subdirectory name **MUST** match the case of the sub-namespace names.

The terminating class name corresponds to a file name ending in .php. The file name **MUST** match the case of the terminating class name.

Autoloader implementations **MUST NOT** throw exceptions, **MUST NOT** raise errors of any level, and **SHOULD NOT** return a value.



By **Dave Child** (DaveChild)
cheatography.com/davechild/
www.addedbytes.com

Published 21st February, 2014.
Last updated 1st June, 2014.
Page 4 of 4.

Sponsored by **Readability-Score.com**
Measure your website readability!
<https://readability-score.com>