**PHP Coding Standard**

The following describes the mandatory requirements that must be adhered to for autoloader interoperability.

1. **Mandatory**

* 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.

1. **Examples**

* \Doctrine\Common\IsolatedClassLoader => /path/to/project/lib/vendor/Doctrine/Common/IsolatedClassLoader.php
* \Symfony\Core\Request => /path/to/project/lib/vendor/Symfony/Core/Request.php
* \Zend\Acl => /path/to/project/lib/vendor/Zend/Acl.php
* \Zend\Mail\Message => /path/to/project/lib/vendor/Zend/Mail/Message.php

1. **Underscores in Namespaces and Class Names**

* \namespace\package\Class\_Name => /path/to/project/lib/vendor/namespace/package/Class/Name.php
* \namespace\package\_name\Class\_Name => /path/to/project/lib/vendor/namespace/package\_name/Class/Name.php

The standards we set here should be the lowest common denominator for painless autoloader interoperability. You can test that you are following these standards by utilizing this sample SplClassLoader implementation which is able to load PHP 5.3 classes.

1. **Example Implementation**

Below is an example function to simply demonstrate how the above proposed standards are autoloaded.

<?php

function autoload($className)

{

$className = ltrim($className, '\\');

$fileName = '';

$namespace = '';

if ($lastNsPos = strrpos($className, '\\')) {

$namespace = substr($className, 0, $lastNsPos);

$className = substr($className, $lastNsPos + 1);

$fileName = str\_replace('\\', DIRECTORY\_SEPARATOR, $namespace) . DIRECTORY\_SEPARATOR;

}

$fileName .= str\_replace('\_', DIRECTORY\_SEPARATOR, $className) . '.php';

require $fileName;

}

<https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md>

**PSR-1-basic-coding-standard**

# Basic Coding Standard

This section of the standard comprises what should be considered the standard coding elements that are required to ensure a high level of technical interoperability between shared PHP code.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](http://www.ietf.org/rfc/rfc2119.txt).

## 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](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md).
* 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.

## 2. Files

### 2.1. PHP Tags

PHP code MUST use the long <?php ?> tags or the short-echo <?= ?> tags; it MUST NOT use the other tag variations.

### 2.2. Character Encoding

PHP code MUST use only UTF-8 without BOM.

### 2.3. Side Effects

A file SHOULD declare new symbols (classes, functions, constants, etc.) and cause no other side effects, or it SHOULD execute logic with side effects, but SHOULD NOT do both.

The phrase "side effects" means execution of logic not directly related to declaring classes, functions, constants, etc., merely from including the file.

"Side effects" include but are not limited to: generating output, explicit use of require or include, connecting to external services, modifying ini settings, emitting errors or exceptions, modifying global or static variables, reading from or writing to a file, and so on.

The following is an example of a file with both declarations and side effects; i.e, an example of what to avoid:

<?php

// side effect: change ini settings

ini\_set('error\_reporting', E\_ALL);

// side effect: loads a file

include "file.php";

// side effect: generates output

echo "<html>\n";

// declaration

function foo()

{

// function body

}

The following example is of a file that contains declarations without side effects; i.e., an example of what to emulate:

<?php

// declaration

function foo()

{

// function body

}

// conditional declaration is \*not\* a side effect

if (! function\_exists('bar')) {

function bar()

{

// function body

}

}

## 3. Namespace and Class Names

Namespaces and classes MUST follow [PSR-0](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md).

This means each class is in a file by itself, and is in a namespace of at least one level: a top-level vendor name.

Class names MUST be declared in StudlyCaps.

Code written for PHP 5.3 and after MUST use formal namespaces.

For example:

<?php

// PHP 5.3 and later:

namespace Vendor\Model;

class Foo

{

}

Code written for 5.2.x and before SHOULD use the pseudo-namespacing convention of Vendor\_ prefixes on class names.

<?php

// PHP 5.2.x and earlier:

class Vendor\_Model\_Foo

{

}

## 4. Class Constants, Properties, and Methods

The term "class" refers to all classes, interfaces, and traits.

### 4.1. Constants

Class constants MUST be declared in all upper case with underscore separators. For example:

<?php

namespace Vendor\Model;

class Foo

{

const VERSION = '1.0';

const DATE\_APPROVED = '2012-06-01';

}

### 4.2. Properties

This guide intentionally avoids any recommendation regarding the use of $StudlyCaps, $camelCase, or $under\_score property names.

Whatever naming convention is used SHOULD be applied consistently within a reasonable scope. That scope may be vendor-level, package-level, class-level, or method-level.

### 4.3. Methods

Method names MUST be declared in camelCase().

use BarClass as Bar;

use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements \ArrayAccess, \Countable

{

// constants, properties, methods

}

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.

<?php

namespace Vendor\Package;

use FooClass;

use BarClass as Bar;

use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements

\ArrayAccess,

\Countable,

\Serializable

{

// constants, properties, methods

}

### 4.2. Properties

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.

A property declaration looks like the following.

<?php

namespace Vendor\Package;

class ClassName

{

public $foo = null;

}

### 4.3. Methods

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 MUST go on its own line, and the closing brace MUST go on the next line following the body. There MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis.

A method declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces:

<?php

namespace Vendor\Package;

class ClassName

{

public function fooBarBaz($arg1, &$arg2, $arg3 = [])

{

// method body

}

}

### 4.4. Method Arguments

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.

<?php

namespace Vendor\Package;

class ClassName

{

public function foo($arg1, &$arg2, $arg3 = [])

{

// method body

}

}

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.

<?php

namespace Vendor\Package;

class ClassName

{

public function aVeryLongMethodName(

ClassTypeHint $arg1,

&$arg2,

array $arg3 = []

) {

// method body

}

}

### 4.5. abstract, final, and static

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

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

<?php

namespace Vendor\Package;

abstract class ClassName

{

protected static $foo;

abstract protected function zim();

final public static function bar()

{

// method body

}

}

### 4.6. Method and Function Calls

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.

<?php

bar();

$foo->bar($arg1);

Foo::bar($arg2, $arg3);

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.

<?php

$foo->bar(

$longArgument,

$longerArgument,

$muchLongerArgument

);

## 5. Control Structures

The general style rules for control structures are as follows:

* 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. This standardizes how the structures look, and reduces the likelihood of introducing errors as new lines get added to the body.

### 5.1. if, elseif, else

An if structure looks like the following. Note the placement of parentheses, spaces, and braces; and that else and elseif are on the same line as the closing brace from the earlier body.

<?php

if ($expr1) {

// if body

} elseif ($expr2) {

// elseif body

} else {

// else body;

}

The keyword elseif SHOULD be used instead of else if so that all control keywords look like single words.

### 5.2. switch, case

A switch structure looks like the following. Note the placement of parentheses, spaces, and braces. 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.

<?php

switch ($expr) {

case 0:

echo 'First case, with a break';

break;

case 1:

echo 'Second case, which falls through';

// no break

case 2:

case 3:

case 4:

echo 'Third case, return instead of break';

return;

default:

echo 'Default case';

break;

}

### 5.3. while, do while

A while statement looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

while ($expr) {

// structure body

}

Similarly, a do while statement looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

do {

// structure body;

} while ($expr);

### 5.4. for

A for statement looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

for ($i = 0; $i < 10; $i++) {

// for body

}

### 5.5. foreach

A foreach statement looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

foreach ($iterable as $key => $value) {

// foreach body

}

### 5.6. try, catch

A try catch block looks like the following. Note the placement of parentheses, spaces, and braces.

<?php

try {

// try body

} catch (FirstExceptionType $e) {

// catch body

} catch (OtherExceptionType $e) {

// catch body

}

## 6. 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.

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.

A closure declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces:

<?php

$closureWithArgs = function ($arg1, $arg2) {

// body

};

$closureWithArgsAndVars = function ($arg1, $arg2) use ($var1, $var2) {

// body

};

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.

The following are examples of closures with and without argument lists and variable lists split across multiple lines.

<?php

$longArgs\_noVars = function (

$longArgument,

$longerArgument,

$muchLongerArgument

) {

// body

};

$noArgs\_longVars = function () use (

$longVar1,

$longerVar2,

$muchLongerVar3

) {

// body

};

$longArgs\_longVars = function (

$longArgument,

$longerArgument,

$muchLongerArgument

) use (

$longVar1,

$longerVar2,

$muchLongerVar3

) {

// body

};

$longArgs\_shortVars = function (

$longArgument,

$longerArgument,

$muchLongerArgument

) use ($var1) {

// body

};

$shortArgs\_longVars = function ($arg) use (

$longVar1,

$longerVar2,

$muchLongerVar3

) {

// body

};

Note that the formatting rules also apply when the closure is used directly in a function or method call as an argument.

<?php

$foo->bar(

$arg1,

function ($arg2) use ($var1) {

// body

},

$arg3

);

**For Database**

* Use PHP PDO Database driver.
* All tables will be prefixed will relevant [*max 3 chars followed by underscore*] e.g. for HRMIS module all tables prefix will be ‘hrm\_'
* All Table name & Column name will lower case.
* Table name will be “Plural Form”. e.g: users, photos, assigned\_roles etc.
* All table AUTO\_INCREMENT , PRIMART KEY column name will be ‘id’;
* PRIMART KEY column’s (id) default datatype will be “BIGINT(20) UNSIGNED”.
* All column name will be full descriptive and meaning full words, use only underscore (\_) for word separator. e.g. user\_id, ip\_address, last\_attempt\_at, reset\_password\_code, first\_name etc.
* All table “ ENGINE=INNODB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci “
* All string fields will be “COLLATE utf8\_unicode\_ci”
* In QUERY all database and query keyword, function will be in UPPERCASE. e.g.
  + Not “select” -> will be “SELECT”
  + Not “where” -> will be “WHERE”
  + SELECT id, COUNT(\*) AS cnt FROM tbl\_name WHERE cnt > 0 GROUP BY id;
  + SELECT DATE\_ADD('1998-01-30', INTERVAL 1 MONTH);
  + INSERT INTO tbl\_name (col1,col2) VALUES(col2\*2,15);
* Default columns for all tables
  + `id` BIGINT(20) UNSIGNED NOT NULL AUTO\_INCREMENT
  + `organization\_id` INT(11) NOT NULL
  + All Custom fields define Here
  + created\_at TIMESTAMP NOT NULL DEFAULT '0000-00-00 00:00:00',
  + created\_by BIGINT(20) UNSIGNED
  + updated\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP
  + updated\_by BIGINT(20) UNSIGNED

ALL TABLE DEFAULT KEY

* + PRIMARY KEY (`id`,`organization\_id`),
  + UNIQUE KEY `id` (`id`),

### File Name

File names MUST be declared in PascalCaseForm. e.g:

* IsolatedClassLoader.php
* ExampleModel.php
* PhotosController.php

**URI Method / Function (those method call by HTTP )**

URI method/function names MUST be declared in lowercase. e.g:

* create()
* show()
* view\_all\_list()

**Attachment storage policy**

Not define yet, now save in directory / file system in structured way