



- [Downloads](#)
- [Documentation](#)
- [Get Involved](#)
- [Help](#)
- 

[Dutch PHP Conference 2024](#)

[Getting Started](#)

[Introduction](#)

[A simple tutorial](#)

[Language Reference](#)

[Basic syntax](#)

[Types](#)

[Variables](#)

[Constants](#)

[Expressions](#)

[Operators](#)

[Control Structures](#)

[Functions](#)

[Classes and Objects](#)

[Namespaces](#)

[Enumerations](#)

[Errors](#)

[Exceptions](#)

[Fibers](#)

[Generators](#)

[Attributes](#)

[References Explained](#)

[Predefined Variables](#)

[Predefined Exceptions](#)

[Predefined Interfaces and Classes](#)

[Predefined Attributes](#)

[Context options and parameters](#)

[Supported Protocols and Wrappers](#)

[Security](#)

[Introduction](#)

[General considerations](#)

[Installed as CGI binary](#)

[Installed as an Apache module](#)

[Session Security](#)

[Filesystem Security](#)

[Database Security](#)

[Error Reporting](#)

[User Submitted Data](#)

[Hiding PHP](#)

[Keeping Current](#)

[Features](#)

[HTTP authentication with PHP](#)

[Cookies](#)

[Sessions](#)

[Dealing with XForms](#)

[Handling file uploads](#)

[Using remote files](#)

[Connection handling](#)

[Persistent Database Connections](#)

[Command line usage](#)

[Garbage Collection](#)
[DTrace Dynamic Tracing](#)

[Function Reference](#)

[Affecting PHP's Behaviour](#)
[Audio Formats Manipulation](#)
[Authentication Services](#)
[Command Line Specific Extensions](#)
[Compression and Archive Extensions](#)
[Cryptography Extensions](#)
[Database Extensions](#)
[Date and Time Related Extensions](#)
[File System Related Extensions](#)
[Human Language and Character Encoding Support](#)
[Image Processing and Generation](#)
[Mail Related Extensions](#)
[Mathematical Extensions](#)
[Non-Text MIME Output](#)
[Process Control Extensions](#)
[Other Basic Extensions](#)
[Other Services](#)
[Search Engine Extensions](#)
[Server Specific Extensions](#)
[Session Extensions](#)
[Text Processing](#)
[Variable and Type Related Extensions](#)
[Web Services](#)
[Windows Only Extensions](#)
[XML Manipulation](#)
[GUI Extensions](#)

Keyboard Shortcuts

?	This help
j	Next menu item
k	Previous menu item
g p	Previous man page
g n	Next man page
G	Scroll to bottom
g g	Scroll to top
g h	Goto homepage
g s	Goto search (current page)
/	Focus search box

[Пространства имён »](#)
[« Пространства имён](#)

- [Руководство по PHP](#)
- [Справочник языка](#)
- [Пространства имён](#)

Change language: Russian

Обзор пространств имён

(PHP 5 >= 5.3.0, PHP 7, PHP 8)

Пространства имён, в широком смысле, — способ инкапсуляции элементов. Такое абстрактное понятие встречается часто. Например, в каждой операционной системе директории группируют связанные файлы и выступают в роли пространства имён для находящихся в них файлов. Конкретный пример, разрешается размещать файл `foo.txt` сразу в обоих каталогах: `/home/greg` и `/home/other`, но двум копиям файла `foo.txt` нельзя сосуществовать в одной директории. Кроме сказанного, для доступа к файлу `foo.txt` извне каталога `/home/greg` надо через разделитель добавить перед именем файла имя директории, чтобы получить путь `/home/greg/foo.txt`. Этот же принцип распространяется и на пространства имён в программировании.

В PHP пространства имён решают две проблемы, с которыми сталкиваются авторы библиотек и приложений, когда создают переиспользуемые элементы кода, например классы и функции:

- 1. Устраняют конфликт имён между кодом разработчика и внутренними или внешними классами, функциями, константами PHP.
- 2. Создают псевдонимы (или сокращения) для Ну_Очень_Длинных_Имён, чтобы сгладить первую проблему и улучшить читаемость исходного кода.

Пространства имён в PHP помогают группировать логически связанные классы, интерфейсы, функции и константы.

Пример #1 Пример синтаксиса с пространством имён

```
<?php

namespace my\name; // Смотрите раздел «Определение пространств имён»

class MyClass {}
function myfunction() {}
const MYCONST = 1;

$a = new MyClass;
$c = new \my\name\MyClass; // Смотрите раздел «Глобальное пространство»

$a = strlen('hi'); // Смотрите раздел «Пространства имён: возврат
// к глобальному пространству для функций и констант»

$d = namespace\MYCONST; // Смотрите раздел «Ключевое слово namespace и магическая константа __NAMESPACE__»
$d = __NAMESPACE__ . '\MYCONST';
echo constant($d); // Смотрите раздел «Пространства имён и динамические особенности языка»

?>
```


Замечание: Имена пространств имён регистронезависимы.

Замечание:

Название пространства имён «PHP» и составные названия, которые начинаются с этого слова (например, `PHP\Classes`), зарезервированы для внутренних целей языка, их не нужно писать в пользовательском коде.

[+add a note](#)

User Contributed Notes 4 notes

[up](#)
[down](#)
877
[SteveWa](#) 
12 years ago
Thought this might help other newbies like me...

Name collisions means:

you create a function named `db_connect`, and somebody else's code that you use in your file (i.e. an include) has the same function with the same name.

To get around that problem, you rename your function `SteveWa_db_connect` which makes your code longer and harder to read.

Now you can use namespaces to keep your function name separate from anyone else's function name, and you won't have to make extra_long_named functions to get around the name collision problem.

So a namespace is like a pointer to a file path where you can find the source of the function you are working with

[up](#)

[down](#)

290

[Dmitry Snytkine ¶](#)

12 years ago

Just a note: namespace (even nested or sub-namespace) cannot be just a number, it must start with a letter.

For example, let's say you want to use namespace for versioning of your packages or versioning of your API:

```
namespace Mynamespace\1; // Illegal
```

Instead use this:

```
namespace Mynamespace\v1; // OK
```

[up](#)

[down](#)

176

[pierstoval at gmail dot com ¶](#)

9 years ago

To people coming here by searching about namespaces, know that a consortium has studied about best practices in PHP, in order to allow developers to have common coding standards.

These best practices are called "PHP Standard Recommendations" , also known as PSR.

They are visible on this link : <http://www.php-fig.org/psr>

Actually there are 5 coding standards categories :

PSR-0 : Autoloading Standard , which goal is to make the use of Namespaces easier, in order to convert a namespace into a file path.

PSR-1 : Basic Coding Standard , basically, standards :)

PSR-2 : Coding Style Guide, where to put braces, how to write a class, etc.

PSR-3 : Logger Interface , how to write a standard logger

PSR-4 : Improved Autoloading , to resolve more Namespaces into paths.

The ones I want to point are PSR-0 and PSR-4 : they use namespaces to resolve a FQCN (Fully qualified class name = full namespace + class name) into a file path.

Basic example, you have this directory structure :

```
./src/Pierstoval/Tools/MyTool.php
```

The namespacing PSR-0 or PSR-4 standard tells that you can transform this path into a FQCN.

Read the principles of autoload if you need to know what it means, because it's almost mandatory ;) .

Structure :

```
{path}/autoloader.php
```

```
{path}/index.php
```

```
{path}/src/Pierstoval/Tools/MyTool.php
```

Files :

```
<?php
```

```
// {path}/index.php
```

```
include 'autoloader.php';
```

```
$tool = new Pierstoval/Tools/MyTool();
```

```
?>
```

```
<?php
```

```
// {path}/src/Pierstoval/Tools/MyTool.php
namespace Pierstoval\Tools;
class MyTool {}
?>

<?php
// {path}/autoloader.php
function loadClass($className) {
$fileName = '';
$namespace = '';

// Sets the include path as the "src" directory
$includePath = dirname(__FILE__).DIRECTORY_SEPARATOR.'src';

if (false !== ($lastNsPos = stripos($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';
$fullFileName = $includePath . DIRECTORY_SEPARATOR . $fileName;

if (file_exists($fullFileName)) {
require $fullFileName;
} else {
echo 'Class "'. $className. '" does not exist.';
}
}

spl_autoload_register('loadClass'); // Registers the autoloader
?>
```

A standardized autoloader will get the class you want to instantiate (MyTool) and get the FQCN, transform it into a file path, and check if the file exists. If it does, it will `<?php include(); ?>` it, and if you wrote your class correctly, the class will be available within its correct namespace.

Then, if you have the following code :

```
<?php $tool = new Pierstoval/Tools/MyTool(); ?>
```

The autoloader will transform the FQCN into this path :

```
{path}/src/Pierstoval/Tools/MyTool.php
```

This might be the best practices ever in PHP framework developments, such as Symfony or others.

[up](#)

[down](#)

24

[shewa12kpi at gmail dot com ¶](#)

2 years ago

```
<?php
//Here is the simple use case of namespace. See how we can use same named class with the help of namespace. This is how
namespace resolve naming collision.
```

```
namespace Mobile;
```

```
class User
{
```

```
public $name = 'mobile user';
}
```

```
$user = new \Mobile\User;
echo $user->name;
```

```
namespace TV ;
```

```
class User
{
public static $name = 'tv user';
}
```

```
echo \TV\User::$name;
```

[+add a note](#)

- [Пространства имён](#)
 - [Обзор](#)
 - [Пространства имён](#)
 - [Подпространства имён](#)
 - [Несколько пространств имён в одном файле](#)
 - [Основы](#)
 - [Пространства имён и динамические особенности языка](#)
 - [Ключевое слово namespace и константа `NAMESPACE`](#)
 - [Псевдонимирование и импорт](#)
 - [Глобальное пространство](#)
 - [Возврат к глобальному пространству](#)
 - [Правила разрешения имён](#)
 - [FAQ](#)
- [Copyright © 2001-2024 The PHP Group](#)
- [My PHP.net](#)
- [Contact](#)
- [Other PHP.net sites](#)
- [Privacy policy](#)

