

[Code](#) > [PHP](#)

How to Autoload Classes With Composer in PHP

[Sajal Soni](#) Last updated Aug 19, 2020

20 likes | 8 min | English ▾

In this article, we'll discuss the basics of autoloading in PHP and how to autoload PHP classes with Composer. I'll explain why autoloading is so important and show you how to use Composer for autoloading step by step. I'll also explain the difference between the different kinds of autoloading in Composer.

Why Do We Need Autoloading?

When you build PHP applications, you may need to use third-party libraries. And as you know, if you want to use these libraries in your application, you need to include them in your source files by using `require` or `include` statements.

These `require` or `include` statements are fine as long as you're developing small applications. But as your application grows, the list of `require` or `include` statements gets longer and longer, which is a bit annoying and difficult to maintain. The other problem with this approach is that you're loading the entire libraries in your application, including the parts you're not even using. This leads to a heavier memory footprint for your application.

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can find further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

autoloader file which contains the logic of autoloading, and the necessary classes will be included dynamically.

Later in this article, we'll look at autoloading with Composer. But first, I'll explain how you can implement autoloading in PHP *without* Composer.

How Autoloading Works Without Composer

You might not realize it, but it is possible to implement autoloading in PHP without Composer. The `spl_autoload_register()` function is what makes this possible. The `spl_autoload_register()` function allows you to register functions that will be put into a queue to be triggered sequentially when PHP tries to load classes that are not loaded yet.

Let's quickly go through the following example to understand how it works.

```
1  <?php
2  function custom_autoloader($class) {
3      include 'lib/' . $class . '.php';
4  }
5
6  spl_autoload_register('custom_autoloader');
7
8  $objFooBar = new FooBar();
9  ?>
```

In the above example, we've registered the `custom_autoloader()` function as our custom autoloader by using the `spl_autoload_register()` function. Next, when you try to instantiate the `FooBar` class and it's not yet available, PHP will execute all the registered autoloader functions sequentially. And thus the `custom_autoloader` function is called—it includes the necessary class file, and finally the object is instantiated. For this example, we're assuming the `FooBar` class is defined in the `lib/FooBar.php` file.

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can find further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

In practice, you won't often be writing your own autoloader, though. That's what Composer is for! In the next section, we'll discuss how to use Composer for autoloading in PHP.

Advertisement

How Autoloading Works With Composer

Firstly, make sure to [install Composer on your system](#) if you want to follow along with the examples. When it comes to autoloading with Composer, there are different methods you could choose from.



What Is Composer for PHP and How to Install It

Sajal Soni

27 May 2020

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can find further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

As per the official Composer documentation, PSR-4 is the recommended way of autoloading, and we'll go through that in detail in the next section. In this section, we'll briefly discuss the other three options.

Before we go ahead, let's quickly go through the steps that you need to perform when you want to use Composer autoloading.

- Define the **composer.json** file in the root of your project or library. It should contain directives based on the type of autoloading.
- Run the `composer dump-autoload` command to generate the necessary files that Composer will use for autoloading.
- Include the `require 'vendor/autoload.php'` statement at the top of the file where you want to use autoloading.

Autoloading: The `files` Directive

File autoloading works similarly to `include` or `require` statements that allow you to load entire source files. All the source files that are referenced with the `files` directive will be loaded every time your application runs. This is useful for loading source files that do not use classes.

To use file autoloading, provide a list of files in the `files` directive of the **composer.json** file, as shown in the following snippet.

```
1  {  
2      "autoload": {  
3          "files": ["lib/Foo.php", "lib/Bar.php"]  
4      }  
5  }
```

As you can see, we can provide a list of files in the `files` directive that we want to autoload with Composer. After you create the **composer.json** file in your project root

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can find further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

```
1 <?php
2 require 'vendor/autoload.php';
3
4 // code which uses things declared in the "lib/Foo.php" or "lib/Bar.php" file
5 ?>
```

The `require 'vendor/autoload.php'` statement makes sure that the necessary files are loaded dynamically.

Autoloading: The `classmap` Directive

Classmap autoloading is an improved version of file autoloading. You just need to provide a list of directories, and Composer will scan all the files in those directories. For each file, Composer will make a list of classes that are contained in that file, and whenever one of those classes is needed, Composer will autoload the corresponding file.

Let's quickly revise the **composer.json** file to demonstrate the classmap autoloader.

```
1 {
2     "autoload": {
3         "classmap": ["lib"]
4     }
5 }
```

Run the `composer dump-autoload` command, and Composer will read the files in the **lib** directory to create a map of classes that can be autoloaded.

Autoloading: PSR-0

PSR-0 is a standard recommended by the PHP-FIG group for autoloading. In the PSR-0 standard, you must use namespaces to define your libraries. The fully qualified class

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can find further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

```

4 |         "Tutspplus\\Library": "src"
5 |     }
6 | }
7 | }

```

In PSR-0 autoloading, you need to map namespaces to directories. In the above example, we're telling Composer that anything which starts with the `Tutspplus\Library` namespace should be available in the **src\Tutspplus\Library** directory.

For example, if you want to define the `Foo` class in the **src\Tutspplus\Library** directory, you need to create the **src\Tutspplus\Library\Foo.php** file as shown in the following snippet:

```

1 | <?php
2 | namespace Tutspplus\Library;
3 |
4 | class Foo
5 | {
6 |     //...
7 | }
8 | ?>

```

As you can see, this class is defined in the `Tutspplus\Library` namespace. Also, the file name corresponds to the class name. Let's quickly see how you could autoload the `Foo` class.

```

1 | <?php
2 | require 'vendor/autoload.php';
3 |
4 | $objFoo = new Tutspplus\Library\Foo();
5 | ?>

```

Composer will autoload the `Foo` class from the **src\Tutspplus\Library** directory.

So that was a brief explanation of file, classmap, and PSR-0 autoloading in Composer. In the next section, we'll see how PSR-4 autoloading works.

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can find further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

mimic the directory structure with namespaces.

In PSR-0 autoloading, you must map namespaces to the directory structure. As we discussed in the previous section, if you want to autoload the `Tutspplus\Library\Foo` class, it must be located at `src\Tutspplus\Library\Foo.php`. In PSR-4 autoloading, you can shorten the directory structure, which results in a much simpler directory structure compared to PSR-0 autoloading.

We'll revise the example above—see if you can spot the differences.

Here's what the **composer.json** file looks with PSR-4 autoloading.

```
1 {  
2     "autoload": {  
3         "psr-4": {  
4             "Tutspplus\\Library\\": "src"  
5         }  
6     }  
7 }
```

It's important to note that we've added trailing backslashes at the end of namespaces. The above mapping tells Composer that anything which starts with the `Tutspplus\Library` namespace should be available in the **src** directory. So you don't need to create the **Tutspplus** and **Library** directories. For example, if you request the `Tutspplus\Library\Foo` class, Composer will try to load the **src\Foo.php** file.

It's important to understand that the `Foo` class is still defined under the `Tutspplus\Library` namespace; it's just that you don't need to create directories that mimic the namespaces. The **src\Foo.php** file contents will be identical to those of the **src\Tutspplus\Library\Foo.php** file in the previous section.

As you can see, PSR-4 results in a much simpler directory structure, since you can omit creating nested directories while still using full namespaces.

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can find further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

Advertisement

Conclusion

Today, we discussed autoloading in PHP. Starting with the introduction of different kinds of Composer autoloading techniques, we discussed PSR-0 and PSR-4 autoloading standards in detail in the latter half of the article.

Learn PHP With a Free Online Course

If you want to learn PHP, check out our [free online course on PHP fundamentals!](#)

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can learn more about our privacy policy and how to manage cookies in our [privacy policy](#). For further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

In this course, you'll learn the fundamentals of PHP programming. You'll start with the basics, learning how PHP works and writing simple PHP loops and functions. Then you'll build up to coding classes for simple object-oriented programming (OOP).

Along the way, you'll learn all the most important skills for writing apps for the web: you'll get a chance to practice responding to GET and POST requests, parsing JSON, authenticating users, and using a MySQL database.

**PHP Fundamentals**

Jeremy McPeak

29 Oct 2021

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can find further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



[Show details](#)[OK](#)

PHP

Did you find this post useful?



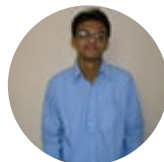
Yes



No

Want a weekly email summary?

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

[Sign up](#)

Sajal Soni

Software Engineer, FSPL, India

I'm a software engineer by profession, and I've done my engineering in computer science. It's been around 14 years I've been working in the field of website development and open-source technologies.

Primarily, I work on PHP and MySQL-based projects and frameworks. Among them, I've worked on web frameworks like CodeIgniter, Symfony, and Laravel.

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can learn more about our privacy policy and how to manage cookies in our privacy policy. For further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.



Show details

OK


travel, explore new places, and listen to music:





Advertisement

QUICK LINKS - Explore popular categories

ENVATO TUTORIALS	+
JOIN OUR COMMUNITY	+
HELP	+


31,093
Tutorials


1,316
Courses


49,125
Translations

Your privacy matters

Cookies and similar technologies are used on our sites to personalise content and ads, provide and improve product features and to analyse traffic on our sites by Envato, our business partners and authors. You can manage your preferences further details below. By continuing to use our sites and services, you agree to the use of these cookies and similar technologies.

