

What's New in PHP 8.[1234]?

phpday 2024

Derick Rethans

derick@php.net — @derickr@phpc.social

<https://derickrethans.nl/talks/php-dpc24>

<https://joind.in/talk/229a9>

About Me

Derick Rethans

- I'm European, living in London
- ~~PHP 7.4 Release Manager~~
- **PHP Foundation**
- **Xdebug** & PHP's Date/Time support
- I ♥ 🌐 maps, I ♥ 🍺 beer, I ♥ 🥃 whisky
- mastodon: @derickr@phpc.social



Readonly Properties

To enforce 'readonly'-bility:

```
<?php
class User
{
    public function __construct(private string $name) {}

    public function getName(): string {
        return $this->name;
    }
}
```

Readonly Properties

In PHP 8.1, with a real keyword, `readonly`:

```
<?php
class User
{
    public function __construct(public readonly string $name) {}
}
```

Readonly Properties

In PHP 8.1, with a real keyword, `readonly`:

```
<?php
class User
{
    public function __construct(public readonly string $name) {}
}
```

Restrictions:

- Can only be initialised once
- Can only be initialised from the scope where it was declared

Readonly Classes

From PHP 8.2, marking all properties as readonly:

```
<?php
readonly class User
{
    public function __construct(public string $name) {}
}
```

Additionally:

- Prevents dynamic properties from being created
- Can't be used if there are untyped or static properties
- Inherited classes **must** also be readonly

Readonly Amendments

In PHP 8.3:

```
<?php
class LogEntry
{
    public function __construct(
        public readonly string $title,
        public readonly DateTimeImmutable $current,
    ) {}

    public function __clone()
    {
        $this->current = new DateTimeImmutable();
    }
}

$entry = new LogEntry("PHP Sussex is great", new DateTimeImmutable());
$newEntry = clone $entry;
?>
```

You can now re-assign readonly properties during `clone`

Deprecating Dynamic Properties

PHP 8.2:

```
<?php
class ezcGraphDataSet
{
    protected $palette;

    function setPalette( $palette )
    {
        $this->palette = $palette;
    }
}

$e = new ezcGraphDataSet;
$e->setPalette( 'rgb' );
```

Deprecated: Creation of dynamic property ezcGraphDataSet::\$palette is deprecated

Deprecating Dynamic Properties

PHP 8.2:

```
<?php
class ezcGraphDataSet
{
    protected $palette;

    function setPalette( $palette )
    {
        $this->palette = $palette;
    }
}

$e = new ezcGraphDataSet;
$e->setPalette( 'rgb' );
```

Deprecated: Creation of dynamic property ezcGraphDataSet::\$palette is deprecated

```
--- src/datasets/base.php
+++ src/datasets/base.php
@@ -83,7 +83,7 @@ abstract class ezcGraphDataSet implements ArrayAccess, Iterator, Countable
     *
     * @var ezcGraphPalette
     */
-    protected $palette;
+    protected $palette;
```

Enumerations — Fetch Properties

```
<?php
enum Currency: string {
    case GBP = '£';
    case EUR = '€';
}

const PRICES = [
    Currency::EUR => 19.99,
    Currency::GBP => 17.99,
];
```

PHP 8.1:

```
Fatal error: Constant expression contains invalid operations
```

PHP 8.2:

```
Fatal error: Constant expression contains invalid operations
```

Enumerations — Fetch Properties

```
<?php
enum Currency: string {
    case GBP = '£';
    case EUR = '€';
}

const PRICES = [
    Currency::EUR->value => 19.99,
    Currency::GBP->value => 17.99,
];
```

First Class Callable Syntax

```
<?php
class Test {
    public function getPrivateMethod() {
        return [$this, 'privateMethod'];
        return Closure::fromCallable([$this, 'privateMethod']);
        return $this->privateMethod(...);
    }

    private function privateMethod() {
        echo     METHOD    , "\n";
    }
}

$test = new Test;
$privateMethod = $test->getPrivateMethod();
$privateMethod();
?>
```

First Class Callable Syntax

```
<?php
class Test {
    public function getPrivateMethod() {
        return [$this, 'privateMethod'];
        return Closure::fromCallable([$this, 'privateMethod']);
        return $this->privateMethod(...);
    }

    private function privateMethod() {
        echo     METHOD    , "\n";
    }
}

$test = new Test;
$privateMethod = $test->getPrivateMethod();
$privateMethod();
?>
```

First Class Callable Syntax

```
<?php
class Test {
    public function getPrivateMethod() {
        return [$this, 'privateMethod'];*
        return Closure::fromCallable([$this, 'privateMethod']);
        return $this->privateMethod(...); // new in PHP 8.1
    }

    private function privateMethod() {
        echo     METHOD    , "\n";
    }
}

$test = new Test;
$privateMethod = $test->getPrivateMethod();
$privateMethod();
?>
```

First Class Callable Syntax

Syntactic Sugar

```
$fn = Closure::fromCallable('strlen');  
$fn = strlen(...);  
  
$fn = Closure::fromCallable([$this, 'method']);  
$fn = $this->method(...)  
  
$fn = Closure::fromCallable([Foo::class, 'method']);  
$fn = Foo::method(...);
```

Other variants

```
strlen(...);  
$closure(...);  
$obj->method(...);  
$obj->$methodStr(...);  
($obj->property)(...);  
Foo::method(...);  
$classStr::$methodStr(...);  
self::{ $complex . $expression }(...);  
[$obj, 'method'](...);  
[Foo::class, 'method'](...);
```

PHP 8.2: null and false types

- `false` could only be used as part of a union (i.e. `string|false`)
- But not as `null|false` and had to use `null|bool`
- You could not use `null` standalone, even though it's a narrower type:

```
<?php
class User {}

class UserFinder {
    function findUserByEmail(string $email) : User|null {}
}

class AlwaysNullUserFinder extends UserFinder {
    function findUserByEmail(string $email) : null {}
}
?>
```

`void` means: Does not return **anything**

`null` means: Does return the **null** value

PHP 8.2: true type

We had `false` and `null`, but not `true`



PHP 8.2: DNF Types

Union Types: $X|Y$

```
class Number {  
    private int|float $number;  
  
    public function setNumber(int|float $number): void {  
        $this->number = $number;  
    }  
  
    public function getNumber(): int|float {  
        return $this->number;  
    }  
}
```

(Pure) Intersection Types: $X\&Y$

Disjunctive Normal Form Types: $(X\&A) | (Y\&B)$

PHP 8.2: DNF Types

Union Types: **X|Y**

(Pure) Intersection Types: **X&Y**

```
class Iteraty {  
    private Traversable&Countable $countableIterator;  
  
    public function setIterator(Traversable&Countable $countableIterator): void {  
        $this->countableIterator = $countableIterator;  
    }  
  
    public function getIterator(): Traversable&Countable {  
        return $this->countableIterator;  
    }  
}
```

Disjunctive Normal Form Types: **(X&A) | (Y&B)**

PHP 8.2: DNF Types

Union Types: $X|Y$

(Pure) Intersection Types: $X\&Y$

Disjunctive Normal Form Types: $(X\&A) | (Y\&B)$

```
class Foreachy {  
    private array|(Traversable&Countable) $foreachableData;  
  
    public function setIterator(array|(Traversable&Countable) $foreachable): void  
        $this->foreachableData = $foreachable;  
    }  
  
    public function getIterator(): array|(Traversable&Countable) {  
        return $this->foreachableData;  
    }  
}
```

PHP 8.2: DNF Types

Union Types: $X | Y$

(Pure) Intersection Types: $X \& Y$

Disjunctive Normal Form Types: $(X \& A) | (Y \& B)$

PHP 8.2: New Random extension

```
<?php
$engine = new \Random\Engine\Xoshiro256StarStar( /* 42 */ );
$r = new \Random\Randomizer($engine);

echo bin2hex( $r->getBytes( 8 ) ), "\n";

echo $r->getInt(1, 100), "\n";
echo $r->nextInt(), "\n";

$fruits = ['red' => '🍎', 'green' => '🥝', 'yellow' => '🍌', 'purple' => '🍇'];
echo "Keys: ", join( ' ', $r->pickArrayKeys( $fruits, 2 ) ), "\n";

echo "Salad: ", join( ' ', $r->shuffleArray( $fruits ) ), "\n";

echo $r->shuffleBytes( "PHP is great!" ), "\n";
?>
```

Result:

```
c66b29a1dc7ec682
61
8117993188557276625
Keys: red, green
Salad: 🍌, 🍎, 🍇, 🥝
rs!agPt eH Pi
```

PHP 8.3: Randomiser Additions

```
<?php
$r = new \Random\Randomizer();

echo "Random domain name: ",
    $r->getBytesFromString('abcdefghijklmnopqrstuvwxyz0123456789', 16),
    ".example.com\n";
echo "Back-up Code: ",
    implode('-', str_split($r->getBytesFromString('0123456789', 20), 5)),
    "\n";

printf(
    "Random Coordinates: Lat: %+.6f Long: %+.6f",
    $r->getFloat(-90, 90, \Random\IntervalBoundary::ClosedClosed),
    $r->getFloat(-180, 180, \Random\IntervalBoundary::OpenClosed),
);

?>
```

Result:

```
Random domain name: 5qgmtqyywanmgim2.example.com
Back-up Code:      81426-91840-12496-18904
Random Coordinates: Lat: -67.205278 Long: +101.458724
```

Sensitive Parameters

PHP 8.2:

```
<?php
function logIn($userName, #[\SensitiveParameter] $password) {
    throw new \Exception('Error');
}

logIn( 'derick', 'secret-elephant' );
```

Without Xdebug:

```
Fatal error: Uncaught Exception: Error in Standard input code:4
Stack trace:
#0 Standard input code(7): logIn('derick', Object(SensitiveParameterValue))
#1 {main}
  thrown in Standard input code on line 4
```

Sensitive Parameters

PHP 8.2:

```
<?php
function logIn($userName, #[\SensitiveParameter] $password) {
    throw new \Exception('Error');
}

logIn( 'derick', 'secret-elephant' );
```

With Xdebug:

```
Fatal error: Uncaught Exception: Error in Standard input code on line 4

Exception: Error in Standard input code on line 4

Call Stack:
 4.8626  399576  1. {main}() Standard input code:0
 4.8626  399576  2. logIn($userName = 'derick', $password = '[Sensitive Parameter
```

PHP 8.3: New #[Override] Attribute

```
<?php
final class MyApp\Tests\MyTest extends PHPUnit\Framework\TestCase
{
    protected $logFile;

    protected function setUp(): void
    {
        $this->logFile = fopen('/tmp/logfile', 'w');
    }

    protected function tearDown(): void
    {
        fclose($this->logFile);
        unlink('/tmp/logfile');
    }
}
?>
```

PHP 8.3: New #[Override] Attribute

```
<?php
final class MyApp\Tests\MyTest extends PHPUnit\Framework\TestCase
{
    protected $logFile;

    protected function setUp(): void
    {
        $this->logFile = fopen('/tmp/logfile', 'w');
    }

    #[\Override]
    protected function tearDown(): void
    {
        fclose($this->logFile);
        unlink('/tmp/logfile');
    }
}
?>
```

If this attribute is added to a method, PHP validates that a method with the same name exists in a parent class or any of the implemented interfaces.

PHP 8.3: Typed Class Constants

```
<?php
interface I {
    const TEST = "Test";
}

class Foo implements I {
    const TEST = [];
}

class Bar extends Foo {
    const TEST = null;
}
?>
```

PHP 8.3: Typed Class Constants

```
<?php
interface I {
    const string TEST = "Test";
}

class Foo implements I {
    const TEST = [];
}

class Bar extends Foo {
    const float TEST = M_PI;
}
?>
```

PHP 8.3: Typed Class Constants

```
<?php
interface I {
    const string TEST = "Test";
}

class Foo implements I {
    const TEST = [];
}

class Bar extends Foo {
    const float TEST = M_PI;
}
?>
```

Fatal error: Type of Foo:TEST must be compatible with I::TEST of type string

Dynamic Class Constant Fetch

Looking up members' names:

- Variables: `$$foo`
- Properties: `$foo->$bar`
- Static properties: `Foo::${$bar}`
- Methods: `$foo->{$bar}()`
- Static methods: `Foo::{$bar}()`
- Classes for static properties: `$foo::$bar`
- Classes for static methods: `$foo::bar()`
- ~~Class constants: `$foo::{$bar}`~~

Dynamic Class Constant Fetch

Looking up members' names:

- ...
- Class constants: `$foo::{$bar}`

```
<?php
class Foo {
    const BAR = 'bar';
}
$bar = 'BAR';

echo Foo::{$bar}; // bar
?>
```

Also works for magic `class` constant:

```
<?php
namespace Foo;
class Bar {}

$class = 'class';
echo Bar::{$class}; // Foo\Bar
?>
```

PHP 8.3: Arbitrary Static Variable Initializers

```
<?php
function bar() {
    echo "bar() called\n";
    return 1;
}

function foo() {
    static $i = bar();
    echo $i++, "\n";
}

echo "BEGIN\n";
foo();
foo();
foo();
```

Result:

```
BEGIN
bar() called
1
2
3
```

PHP 8.3: json_validate()

For when you need to know JSON is valid, but don't need to parse it:

```
<?php
$valid = json_validate('{ "test": { "foo": "bar" } }');

$valid = json_validate('{ ": ": " } }');
var_dump($valid, json_last_error(), json_last_error_msg());
```

Result:

```
/home/httpd/pres2/show2.php(27) : eval()'d code:5:boolean false
```

```
/home/httpd/pres2/show2.php(27) : eval()'d code:5:int 4
```

```
/home/httpd/pres2/show2.php(27) : eval()'d code:5:string 'Syntax error' (length=12)
```

PHP 8.4: Parsing HTML5

HTML 5 parsing, through the following new class hierarchy:

```
<?php
namespace DOM {
    abstract class Document extends DOM\Node implements DOM\ParentNode {
        /* all properties and methods that are common and sensible for both XML
        * & HTML documents */
    }

    final class XMLDocument extends Document {
        /* XML specific properties and methods */
    }

    final class HTMLDocument extends Document {
        /* HTML specific properties and methods */
    }
}

class DOMDocument extends DOM\Document {
    /* Keep methods, properties, and constructor the same as they are now */
}
?>
```

PHP 8.0: Just-In-Time (JIT) Compiler

- Compiles PHP code to x86 machine code
- Performance improvements mostly apply to math heavy code
- Part of opcache:

```
opcache.jit=on  
opcache.jit_buffer_size=128M
```

PHP 8.4: Just-In-Time (JIT) Compiler

- Now based on a separately developed Intermediate Representation
- Should be easier to maintain, and support multiple targets
- Still part of opcache, and replaces the old JIT

Also changes to default settings:

```
opcache.jit=disable  
opcache.jit_buffer_size=64M
```

PHP 8.4: PDO Subclasses

Each PDO driver now has it's own class, containing driver specific elements

PDODBLib, PDOFirebird, PDOOCI, PDOODBC are just children, without any changes.

PDOMySQL has getWarningCount() defined on it

PDOPgSQL and PDOSQLite have a whole bunch of constants and methods

A new factory method, PDO::connect returns an object representing one of the new subclasses:

```
<?php
$db = PDO::connect('sqlite::memory:');
if (!$db instanceof PDOSQLite) {
    /* ERROR ZONE */
}
```

PHP 8.4: Property Hooks

```
<?php
class User implements Named
{
    private bool $isModified = false;

    public function __construct(private string $first, private string $last) {}

    public string $fullName {
        // Override the "read" action with arbitrary logic.
        get => $this->first . " " . $this->last;

        // Override the "write" action with arbitrary logic.
        set {
            [$this->first, $this->last] = explode(' ', $value);
            $this->isModified = true;
        }
    }
}
?>
```

Policies and Procedures

- Information hidden away in many RFCs
- No consolidated documentation

<https://github.com/php/policies>

Policies and Procedures

- Information hidden away in many RFCs
- No consolidated documentation

<https://github.com/php/policies>

Still to do:

- Rewrite documents to be statements of facts instead of intent
- Figure out which information isn't written down authoritatively yet

Release Cycle Update

Current:

- 2 + 1 years of support
- 3 alphas, 3, betas, 6 RCs
- No new minor features in beta releases
- New minor features could be added during RC and bug fix releases
- Support ends exactly 3 years after initial release

Release Cycle Update

New:

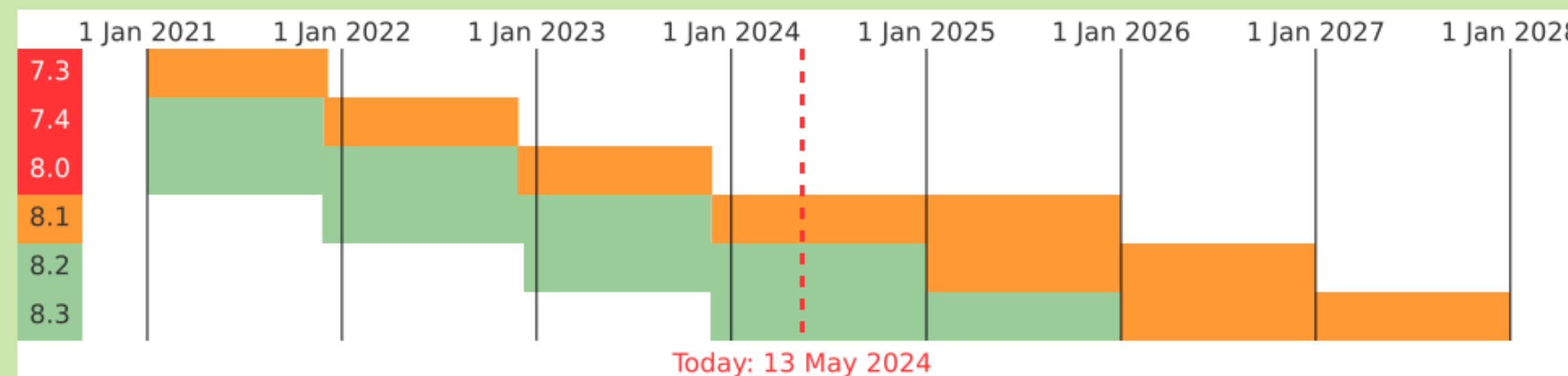
- 2 + 2 years of support
- 3 alphas, 3, betas, 4 RCs
- New minor features in beta releases
- New minor features may not be added during RC and bug fix releases
- Support ends at the end of year 4 after initial release

Release Cycle Update

New:

- 2 + 2 years of support
- 3 alphas, 3, betas, 4 RCs
- New minor features in beta releases
- New minor features **may not** be added during RC and bug fix releases
- Support ends **at the end of year 4** after initial release

Effective Immediately





Any Queries?

Resources



Slides

<https://derickrethans.nl/talks/php-dpc24>

<https://joind.in/talk/229a9>

<https://xdebug.cloud>

Derick Rethans — @derickr@phpc.social — derick@php.net