What changes in PHP 8.2?

New features, Deprecations, BC breaks, etc.

George Peter Banyard

November 24, 2022

The PHP Foundation

About me

Twitter: @Girgias

Mastodon: @Girgias@phpc.social

GitHub: Girgias

Site: *https://gpb.moe*

- Studied pure mathematics
- PHP Core dev financed part time by The PHP Foundation
- Lead maintainer for the FR documentation of PHP
- Author of the CSV extension
- RFC Author of:
  - Saner Numeric Strings
  - DNF Types
  - And others...

Table of Contents

Backwards-Compatibility Breaks

ASCII only case conversion for string functions

The following function are now locale *in*sensitive:

- *strtolower()*
- *strtoupper()*
- *stristr()*
- *stripos()*
- *strripos()*
- *lcfirst()*
- *ucfirst()*
- *ucwords()*
- *str_ireplace()*
- *array_change_key_case()* in SORT_FLAG_CASE mode

ASCII only case conversion for string functions: example

```php
<?php
setlocale(LC_ALL, 'de_DE');
$w = "SCHÖN";
var_dump(strtolower($w));
```

Output in PHP 8.2:

```
string(6) "schÖn"
```

Output prior to PHP 8.2:

```
string(6) "schön"
```

Fix

Use the corresponding MBstring function with a specified encoding instead

`str_split()` edge case behavioural change

```php
<?php
$s = '';
var_dump(str_split($s, 3));
```

Output in PHP 8.2:

```
array(0) {
}
```

Output prior to PHP 8.2:

```
array(1) {
  [0]=>
  string(0) ""
}
```

Note
This aligns the behaviour with *`mb_str_split()`*

Internal methods Tentative return type changes

`DateTime::createFromImmutable()`
changed from `DateTime` to `static`

`DateTimeImmutable::createFromMutable()`
changed from `DateTimeImmutable` to `static`

`SplFileObject::hasChildren()`
changed from `bool` to `false`

`SplFileObject::getChildren()`
changed from `?RecursiveIterator` to `null`

Internal method signature enforcement

The following methods, now enforce their signature:

- *`SplFileObject::getCsvControl()`*
- *`SplFileObject::fflush()`*
- *`SplFileObject::ftell()`*
- *`SplFileObject::fgetc()`*
- *`SplFileObject::fpassthru()`*

Deprecations

Dynamic properties

```php
<?php
class O {
    public $prop;
}
$o = new O();
$o->prOp = 5;
```

Output in PHP 8.2:

```
Deprecated: Creation of dynamic property O::$prOp is deprecated in ... on
↪  line 6
```

Output prior to PHP 8.2:

Dynamic properties

```php
<?php
class O {
    public array $props = [];
    public function __set($name, $value)  {
        $this->props[$name] = $value;
    }
    public function __get($name) {
        return $this->props[$name];
    }
}
$o = new O();
$o->value = 5;
var_dump($o->value);
```

Output in PHP 8.2 *and* prior to PHP 8.2:

```
int(5)
```

Dynamic properties: how to fix?

1. Declaring the property
2. Using a **`WeakMap`** if additional data needs to be associated with an object which one does not own
3. Adding the *`#[\AllowDynamicProperties]`* attribute to the class, this also applies to *all* child classes

Relative callables

Callables that cannot be called using the `$callable()` syntax, meaning they can only be called using `call_user_func()` are deprecated. In particular:

- `"self::method"`
- `"parent::method"`
- `"static::method"`
- `["self", "method"]`
- `["parent", "method"]`
- `["static", "method"]`
- `["Foo", "Bar::method"];`
- `[new Foo(), "Bar::method"];`

Relative callables: sorry what?

```php
<?php
class Bar {
    public function method() { echo "Bar\n"; }
    public static function sMethod() {  echo "Bar static\n"; }
}
class Foo extends Bar {
    public function method() { echo "Foo\n"; }
    public static function sMethod() { echo "Foo static\n"; }
    public function proxy(callable $f) { call_user_func($f); }
}
call_user_func(["Foo", "Bar::sMethod"]);
$o = new Foo();
$o->proxy("self::method");
$o->proxy("parent::method");
$o->proxy(["self", "method"]);
$o->proxy(["parent", "method"]);
```

Output prior to PHP 8.2:

```
Bar static
Foo
Bar
Foo
Bar
```

Relative callables: fixing them

If compatibility with versions of PHP prior to 8.1 is not needed: use the first-class callable syntax *`self::method(...)`*

Otherwise, concatenate the class name to remove the dependence on context:

```php
<?php
"self::method"       -> self::class . "::method"
"parent::method"     -> parent::class . "::method"
"static::method"     -> static::class . "::method"
["self", "method"]   -> [self::class, "method"]
["parent", "method"] -> [parent::class, "method"]
["static", "method"] -> [static::class, "method"]
```

Relative callables: sorry what?[2]

Calling the parent implementation of an overriden method:

```php
<?php
class Bar {
    public function method() { echo "Bar\n"; }
}
class Foo extends Bar {
    public function method() { echo "Foo\n"; }
}

$o = new Foo();
call_user_func([$o, "Bar::method"]);
```

Output in PHP 8.2:

```
Deprecated: Callables of the form ["Foo", "Bar::method"] are deprecated
Bar
```

Output prior to PHP 8.2:

```
Bar
```

Relative callables: calling parent implementation

```php
<?php
class Bar {
    public function method() { echo "Bar\n"; }
}
class Foo extends Bar {
    public function method() { echo "Foo\n"; }
}

// Using reflection:
(new ReflectionMethod("Bar", "method"))->invoke(new Foo);
// Using closure rebinding:
Closure::fromCallable([new Bar, "method"])->bindTo(new Foo)();
```

Output in PHP 8.2 *and* prior to PHP 8.2:

```
Bar
Bar
```

`utf8_encode()` and `utf8_decode()`

Use *`mb_convert_encoding()`*:

```php
<?php
// replace utf8_encode($string) with
$encoded = mb_convert_encoding($string, 'UTF-8', 'ISO-8859-1');

// replace utf8_decode($string) with
$decoded = mb_convert_encoding($string, 'ISO-8859-1', 'UTF-8');
```

New features

Type System Improvements

- **`false`** and **`null`** types can now be used standalone
- **`true`** type has been added
- Disjunctive Normal Form (DNF) types have been added
  e.g. **`(Traversable&Countable)|array`** or **`(A&B)|null`**

OOP Improvements

Traits can now declare constants:

```php
<?php
trait ConstantsTrait {
    public const FLAG_MUTABLE = 1;
    final public const FLAG_IMMUTABLE = 5;
}
class ConstantsExample {
    use ConstantTrait;
}
$example = new ConstantExample;
$example::FLAG;
```

Warning
Trait constants cannot be accessed directly.

```php
<?php
var_dump(ConstantsTrait::FLAG_MUTABLE);
// Fatal error: Trait "ConstantTrait" not found
```

OOP Improvements

The *readonly* modifier can be added to classes. This adds the *readonly* modifier to every property *and* disallows setting dynamic properties

```php
<?php
readonly class Point {
    public function __construct(public int $x, public int $y) {}
}
$p = new Point(5, 25);
try {
    $p->x = 10;
} catch (\Error $e) { echo $e->getMessage(), \PHP_EOL; }
try {
    $p->z = 10;
} catch (\Error $e) { echo $e->getMessage(), \PHP_EOL; }
```

Output in PHP 8.2:

```
Cannot modify readonly property Point::$x
Cannot create dynamic property Point::$z
```

SensitiveParameter Attribute

```php
<?php
function login(
    string $user,
    #[\SensitiveParameter]
    string $password
) { throw new Exception('Error!'); }
try {
    login('Girgias', 'I <3 PHP');
} catch (\Exception $e) { echo $e, PHP_EOL; }
```

Output in PHP 8.2:

```
Exception: Error! in /in/TKaRU:8
Stack trace:
#0 /in/TKaRU(12): login('Girgias', Object(SensitiveParameterValue))
#1 {main}
```

Output prior to PHP 8.2:

```
Exception: Error! in /in/TKaRU:8
Stack trace:
#0 /in/TKaRU(12): login('Girgias', 'I <3 PHP')
#1 {main}
```

Enumerations properties in constant expressions

```php
<?php
enum Suit {
    case Heart;
    case Diamond;
    case Clubs;
    case Spades;
}

class Foo {
    const SUIT = Suit::Heart->name;
}

var_dump(Foo::SUIT);
```

Output in PHP 8.2:

```
string(5) "Heart"
```

Random extension

A new and better API to deal with Randomness in PHP

- A new **`Random\Randomizer`** *`final`* class
- Which takes a **`Random\Engine`** that takes care of generating the randomness
- Various default engines provided by PHP:
  - **`Random\Engine\Secure`** (the default one)
  - **`Random\Engine\Mt19937`**
  - **`Random\Engine\PcgOneseq128XslRr64`**
  - **`Random\Engine\Xoshiro256StarStar`**
- Throws specialised exception in case of errors. *`random_bytes()`* and *`random_int()`* now throw a **`\Random\RandomException`** instead of a simple **`\Exception`**

Random extension: Random\Randomizer

```php
<?php
final class Random\Randomizer {
    /* Properties */
    public readonly Random\Engine $engine;

    /* Methods */
    public __construct(?Random\Engine $engine = null)
    public getBytes(int $length): string
    public getInt(int $min, int $max): int
    public nextInt(): int
    public pickArrayKeys(array $array, int $num): array
    public __serialize(): array
    public shuffleArray(array $array): array
    public shuffleBytes(string $bytes): string
    public __unserialize(array $data): void
}
```

PHP 8.3 new methods

Improvements to this API have already been accepted by introducing *getBytesFromString(...)*, *nextFloat()*, *getFloat(...)* methods.

Rapid Fire Additions

- Added the *curl_upkeep()* function to perform any connection upkeep checks.
- Added support for new cURL options released between versions 7.62 and 7.80 of libcurl
- Added various new Socket options
- Added the *FILTER_FLAG_GLOBAL_RANGE* filter
- Added the *error_log_mode* INI directive
- Reflection improvements by adding *ReflectionFunction::isAnonymous()* and *ReflectionMethod::hasPrototype()* methods
- Added the *memory_reset_peak_usage()* function.

Other changes

iterable type

As of PHP 8.2, **`iterable`** changed from a pseudo-type to a compile time resolved alias of **`Traversable|array`**.

BC is preserved *only* when querying a type via Reflection produced a **`ReflectionNamedType`** (i.e. **`iterable`** and **`?iterable`**). Otherwise, it is decomposed into **`Traversable|array`**.

iterable type

```php
<?php
$function = function(): iterable {};
$returnType = (new ReflectionFunction($function))->getReturnType();
var_dump($returnType::class, (string) $returnType);

$function = function(): iterable|bool {};
$returnType = (new ReflectionFunction($function))->getReturnType();
var_dump($returnType::class, (string) $returnType);
```

Output in PHP 8.2:

```
string(19) "ReflectionNamedType"
string(8) "iterable"
string(19) "ReflectionUnionType"
string(22) "Traversable|array|bool"
```

Output prior to PHP 8.2:

```
string(19) "ReflectionNamedType"
string(8) "iterable"
string(19) "ReflectionUnionType"
string(13) "iterable|bool"
```

INI parsing

Parsing of ill-formatted values will now trigger a warning, but the *interpretation* of these values hasn't changed.

Support for `0O`/`0o` and `0B`/`0b` prefixes for octal and binary integer literals has been added.

Octal and Hexadecimal literals

Those were already support using the `0X`/`0x` prefix for hexadecimals or simply having a `0` in before the integer to be interpreted as octal.

Added the `ini_parse_quantity()` function to gain access to the INI parsing algorithm.

Rapid Fire Changes

- The *$iterator* parameter of *iterator_to_array()* and *iterator_count()* is widened to **iterable** from **Iterator**, allowing **array**s to be passed.
- Properties of the **DatePeriod** and tidy classes are now properly declared.
- Various instances of Intl classes are no longer serializable, because unserialising them yielded unusable objects.
- NUL characters (*\0*) in PCRE pattern strings are now supported.
- Support for libmysql in ext-mysqli has been removed, all related things have been removed or deprecated.
- In the CLI SAPI: The STDOUT, STDERR and STDIN streams are no longer closed on resource destruction, it is still possible to close them using *fclose()*

Thank you!

Twitter: @Girgias

Mastodon: @Girgias@phpc.social

GitHub: Girgias

Site: `https://gpb.moe`