

PHP 7.4 New Features

Presented by Georgi Georgiev

Typed properties
Supported types:
bool, int, float, string, array, object
iterable

```
<?php  
class User {  
    public int $id;  
    public string $name;  
}  
?>
```

Arrow functions

```
<?php  
$factor = 10;  
$nums = array_map(fn($n) => $n * $factor, [1, 2, 3, 4]);  
// $nums = array(10, 20, 30, 40);  
?>
```

Limited return type covariance and argument type contravariance

```
<?php
class A {}
class B extends A {}

class Producer {
    public function method(): A {}
}
class ChildProducer extends Producer {
    public function method(): B {}
}
?>
```

Null coalescing assignment operator

```
<?php
$array['key'] ??= computeDefault();
// is roughly equivalent to
if (!isset($array['key'])) {
    $array['key'] = computeDefault();
}
?>
```

Unpacking inside arrays

```
<?php  
$parts = ['apple', 'pear'];  
$fruits = ['banana', 'orange', ...$parts, 'watermelon'];  
// ['banana', 'orange', 'apple', 'pear', 'watermelon'];  
?>
```

Numeric literal separator

```
var_dump(1_000_000); // int(1000000);
```

```
$time = (int) ($adTime / 10000000 -  
11644473600); // without separator
```

```
$time = (int) ($adTime / 10_000_000 -  
11_644_473_600); // with separator
```

Weak references

```
<?php
$obj = new stdClass;
$weakref = WeakReference::create($obj);
var_dump($weakref->get());
unset($obj);
var_dump($weakref->get());
?>
```


Allow exceptions from __toString()

```
class CustomException
extends Exception
{
}

class Foo
{
    public function __toString()
    {
        throw new CustomException('oops!');
    }
}

$foo = new Foo();
try {
    var_dump((string) $foo);
}
catch (CustomException $e) {
    var_dump($e);
}

?>
```

```
/**
 * Outcome:
 * object(CustomException)#2 (7) {
 *   ["message":protected]=>
 *   string(5) "oops!"
 *   ["string":"Exception":private]=>
 *   string(0) ""
 *   ["code":protected]=>
 *   int(0)
 *   ["file":protected]=>
 *   string(27)
 *   "/var/www/ggeorgiev/test.php"
 *   ["line":protected]=>
 *   int(12)
 *   ["trace":"Exception":private]=>
 *   array(1)
 *   {
 *     [0] => array(5)
 *     {
 *       ["file"]=>
 *       string(27)
 *       "/var/www/ggeorgiev/test.php"
 *       ["line"]=>
 *       int(19)
 *       ["function"]=>
 *       string(10)
 *       "__toString"
 *       ["class"]=>
 *       string(3) "Foo"
 *       ["type"]=>
 *       string(2) "->"
 *     }
 *   }
 *   ["previous":"Exception":private]=>
 *   NULL
 * }
```



Filter

```
<?php
if (filter_var(12,2, FILTER_VALIDATE_FLOAT , array("options" => array("min_range"=>1, "max_range"=>200))) === false) {
    echo("Variable value is not within the legal range");
} else {
    echo("Variable value is within the legal range");
}
?>
```

Foreign Function Interface



Foreign Function Interface Installation

To enable the FFI extension, PHP has to be configured with `--with-ffi` .

Windows users have to include `php_ffi.dll` into `php.ini` to enable the FFI extension.

Foreign Function Interface Example: A Complete PHP/FFI/preloading Example

php.ini

```
ffi.enable=preload
```

```
opcache.preload=preload.php
```

Foreign Function Interface Example: A Complete PHP/FFI/preloading Example

preload.php

```
<?php  
FFI::load(__DIR__ . "/dummy.h");  
opcache_compile_file(__DIR__ . "/dummy.php");  
?>
```

Foreign Function Interface Example: A Complete PHP/FFI/preloading Example

dummy.h

```
#define FFI_SCOPE "DUMMY"  
#define FFI_LIB "libc.so.6"  
  
int printf(const char *format, ...);
```

Foreign Function Interface Example: A Complete PHP/FFI/preloading Example

dummy.php

```
<?php
final class Dummy {
    private static $ffi = null;
    function __construct() {
        if (is_null(self::$ffi)) {
            self::$ffi = FFI::scope("DUMMY");
        }
    }
    function printf($format, ...$args) {
        return (int)self::$ffi->printf($format, ...$args);
    }
}
?>
```



Foreign Function Interface Example: A Complete PHP/FFI/preloading Example

test.php

```
<?php
$d = new Dummy();
$d->printf("Hello %s!\n", "world");
?>
```

IMG_FILTER_SCATTER

```
imagefilter ( resource $image , int $filtertype [, int $arg1 [, int $arg2 [, int $arg3 [, int $arg4 ]]] ) : bool
```

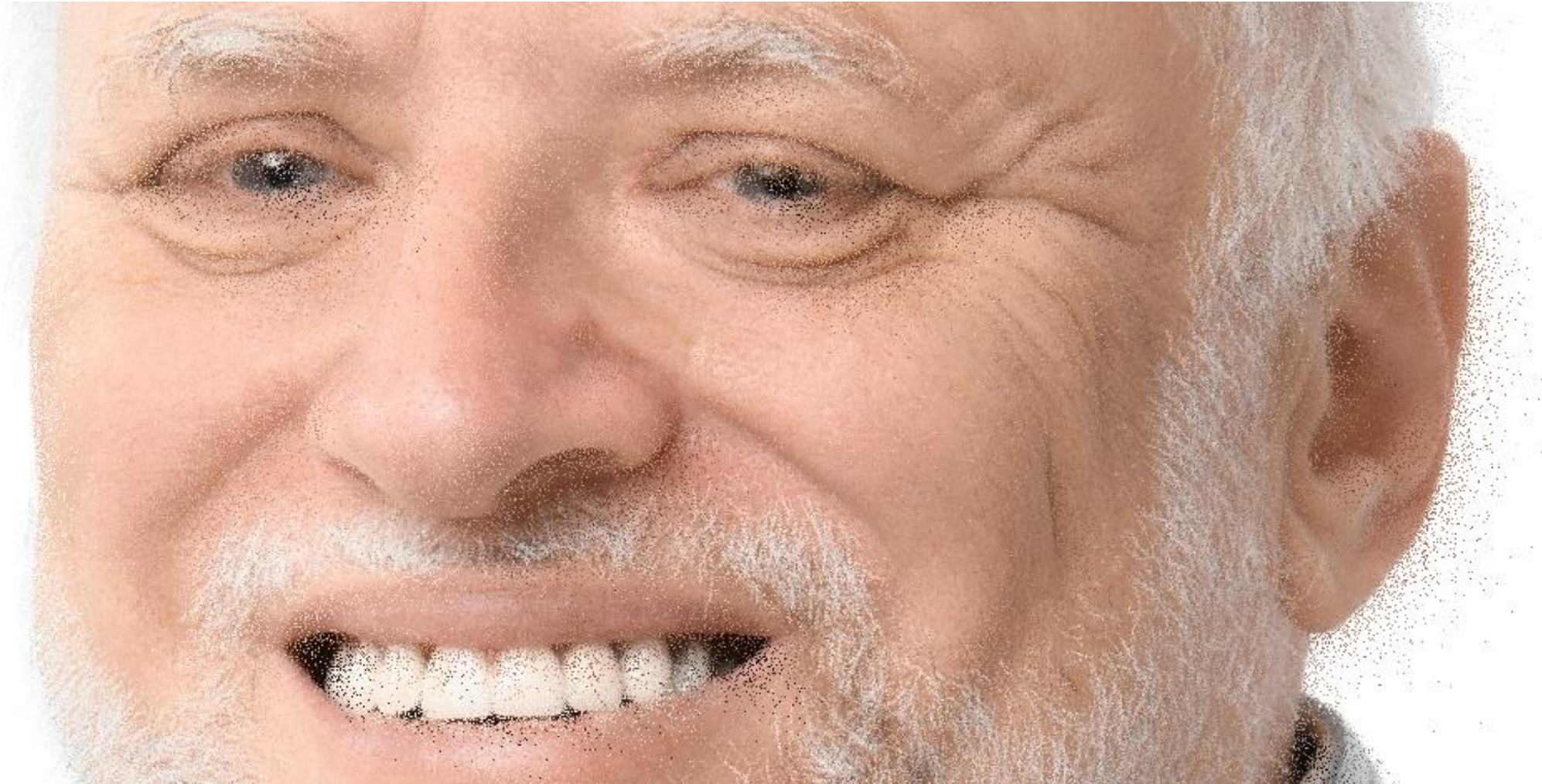
```
<?php
// Load the image
$logo = imagecreatefrompng('./php.png');

// Apply a very soft scatter effect to the image
imagefilter($logo, IMG_FILTER_SCATTER, 3, 5);

// Output the image with the scatter effect
header('Content-Type: image/png');
imagepng($logo);
imagedestroy($logo);
?>
```



IMG_FILTER_SCATTER



Hash

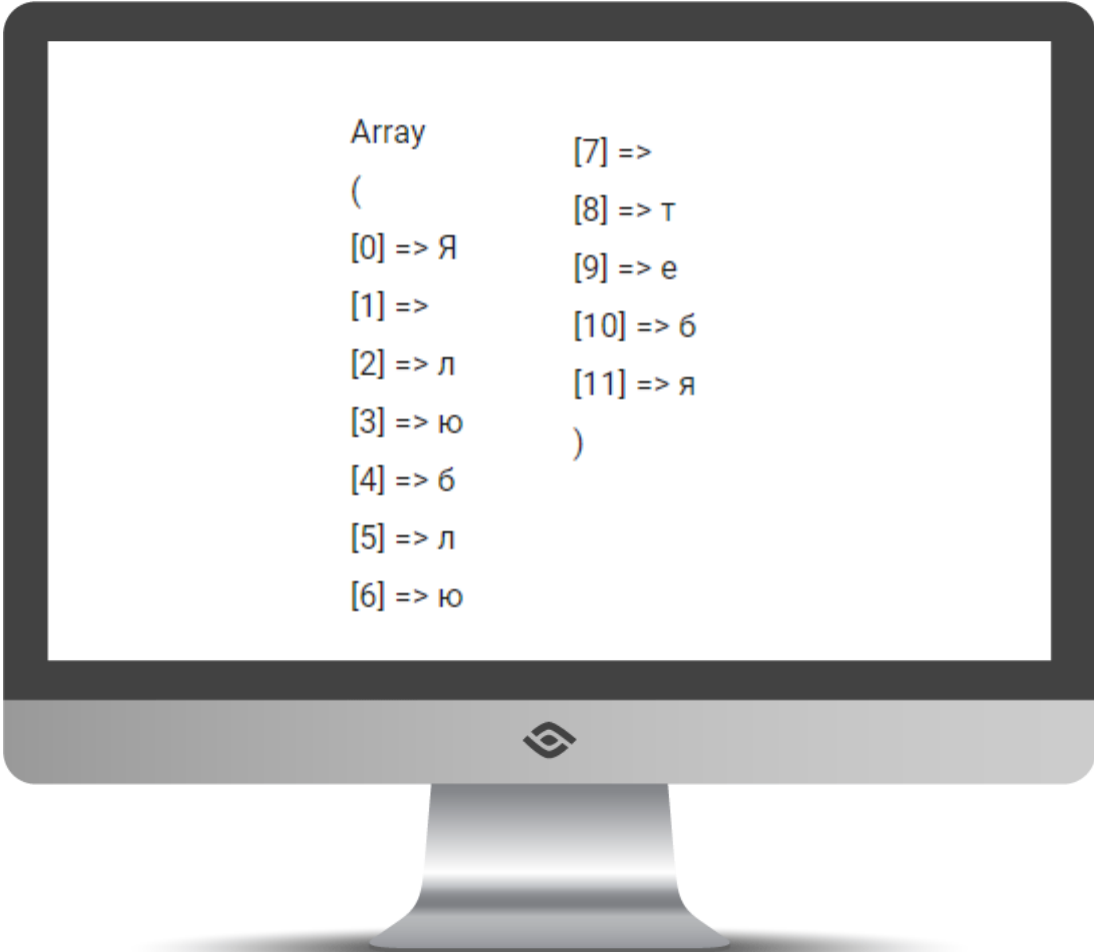
Added *crc32* hash using Castagnoli's polynomial

```
<?php
$checksum = crc32("The quick brown fox jumped over the lazy dog.");
printf("%u\n", $checksum);
?>
//2191738434
```

Multibyte String

```
<?php
$string = 'Я люблю тебя';
$chars = mb_str_split($string);

print_r($chars);
?>
```



```
Array
(
    [0] => Я
    [1] => 
    [2] => л
    [3] => ю
    [4] => б
    [5] => л
    [6] => ю
    [7] => 
    [8] => т
    [9] => е
    [10] => б
    [11] => я
)
```


OPcache

Here's how you'd link to this script in php.ini:

```
opcache.preload=/path/to/project/preload.php
```

And here's a dummy implementation:

```
$files = /* An array of files you want to pre  
  
foreach ($files as $file) {  
    opcache_compile_file($file);  
}
```



PDO

```
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass); //before 7.4  
$dbh = new PDO('mysql:host=localhost;dbname=test;username=root;password=""'); //after 7.4
```

PDOStatement::getColumnMeta

```
<?php
$select = $DB-
>query('SELECT COUNT(*) FROM fruit');
$meta = $select->getColumnMeta(0);
var_dump($meta);
?>
```

```
array(6) {
  ["native_type"]=>
  string(7) "integer"
  ["flags"]=>
  array(0) {
  }
  ["name"]=>
  string(8) "COUNT(*)"
  ["len"]=>
  int(-1)
  ["precision"]=>
  int(0)
  ["pdo_type"]=>
  int(2)
}
```


PDOStatement::getColumnMeta

Example for ENUM column

```
if($conn->query("CREATE TABLE shirts (name VARCHAR(40),size ENUM('x-small', 'small', 'medium', 'large', 'x-large'))")){
    echo 'Query Successfull';
}
echo '<br/>';
$sql='SELECT * FROM `shirts` WHERE 1';
$select = $conn->query($sql);
$meta = $select->getColumnMeta(1);
var_dump($meta);
//Output: array(7) { ["native_type"]=> string(6) "STRING" ["pdo_type"]=> int(2) ["flags"]=> array(0) { } ["table"]=> string(6) "shirts" ["name"]=> string(4) "size" ["len"]=> int(28) ["precision"]=> int(0) }
```

SQLite3_

```
SQLite3::lastExtendedErrorCode() // 7.4
```

strip_tags() with array of tag names

```
<?php  
$text = '<p>Test paragraph.</p><!-- Comment --> <a href="#fragment">Other text</a>';  
// Allow <p> and <a>  
echo strip_tags($text, ['a','p']); //before 7.4: strip_tags($text, '<p><a>')
```

Test paragraph.

[Other text](#)

Custom object serialization

7.4

```
<?php
class Test {
    public $prop;
    public $prop2;
    public function __serialize() {
        return ["value" => $this->prop, 42 => $this->prop2];
    }
    public function __unserialize(array $data) {
        $this->prop = $data["value"];
        $this->prop2 = $data[42];
    }
}

$test = new Test;
$test->prop = "foobar";
$test->prop2 = "barfoo";
var_dump($s = serialize($test));
var_dump(unserialize($s));
?>
```

```
//string(58) "O:4:"Test":2:{s:5:"value";s:6:
"foobar";i:42;s:6:"barfoo";}"
```

```
//object(Test)#2 (2) { ["prop"]=> string(6)
"foobar" ["prop2"]=> string(6) "barfoo" }
```

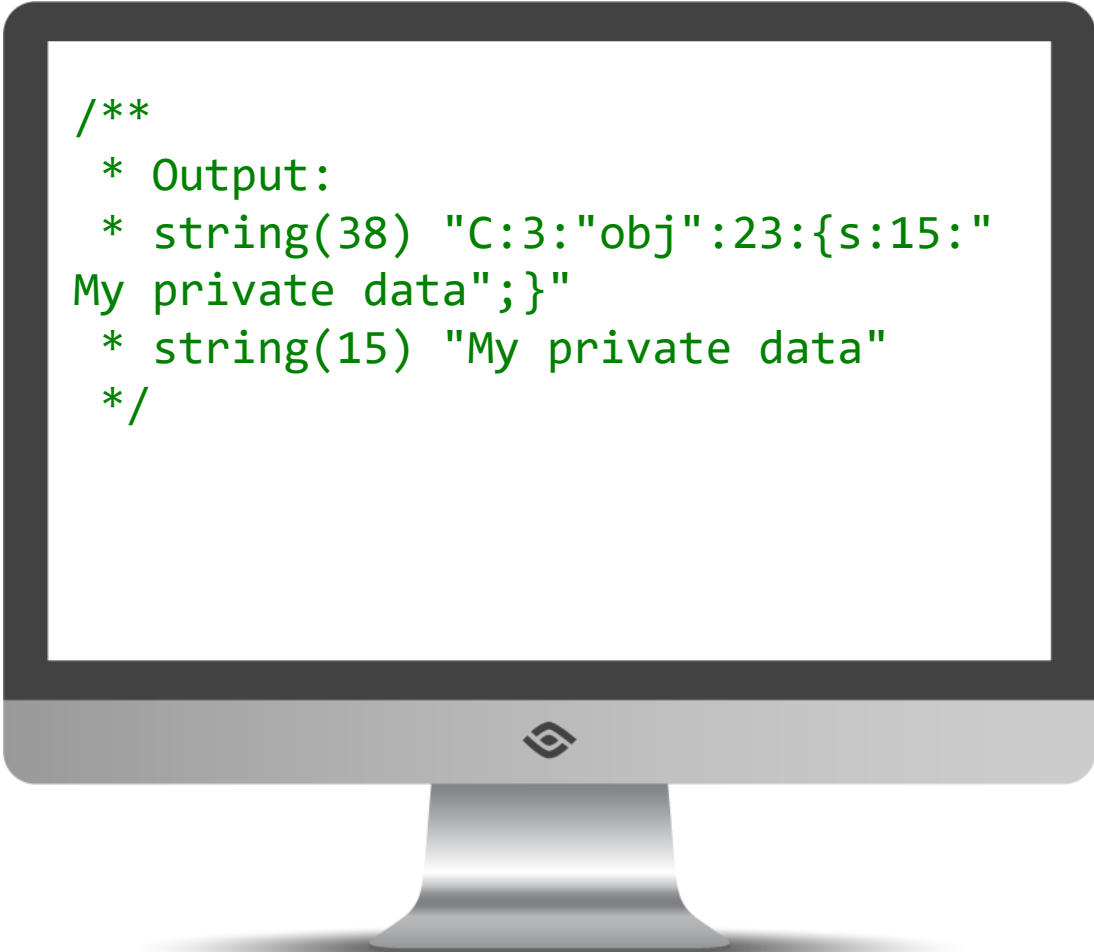
Custom object serialization before 7.4

```
<?php
class obj implements Serializable
{
    private $data;
    public function __construct()
    {
        $this->data = "My private data";
    }
    public function serialize()
    {
        return serialize($this->data);
    }
    public function unserialize($data)
    {
        $this->data = unserialize($data);
    }
    public function getData()
    {
        return $this->data;
    }
}

$obj = new obj;
$ser = serialize($obj);

var_dump($ser);

$newobj = unserialize($ser);
var_dump($newobj->getData());
?>
```



```
/**
 * Output:
 * string(38) "C:3:"obj":23:{s:15:"
My private data";}
 * string(15) "My private data"
 */
```

Array merge functions without arguments

```
<?php  
$result = array_merge();  
var_dump($result);  
//array(0) { }
```

proc_open() function

```
proc_open ( string $cmd , array $descriptorspec , array &$pipes [, string $cwd  
= NULL [, array $env = NULL [, array $other_options = NULL ]]] ) : resource
```

proc_open() function

```
<?php
$descriptorspec = array(
    0 => array("pipe", "r"), // stdin is a pipe that the child will read from
    1 => array("pipe", "w"), // stdout is a pipe that the child will write to
    2 => array("file", "/tmp/testfile.txt", "a") // stderr is a file to write to
);

$cwd = '/tmp';
$env = array('some_option' => 'aeiou');

$process = proc_open(['php', '-r', 'echo "Hello World\n";'], $descriptorspec, $pipes, $cwd, $env);

if (is_resource($process)) {

    fwrite($pipes[0], '<?php print_r($_ENV); ?>');
    fclose($pipes[0]);

    echo stream_get_contents($pipes[1]);
    fclose($pipes[1]);

    $return_value = proc_close($process);

    echo "command returned $return_value\n";
}

/**
 * root@ggeorgiev_server:/var/www/ggeorgiev# php test.php
 * Hello World
 * command returned 0
 */
```



Thank You

