php

- Downloads
- Documentation
- Get Involved
- Help
-

Search

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

New features »
« Migrating from PHP 5.6.x to PHP 7.0.x

- PHP Manual
- Appendices
- Migrating from PHP 5.6.x to PHP 7.0.x

Change language: English

Submit a Pull Request Report a Bug

Backward incompatible changes ¶

Changes to error and exception handling ¶

Many fatal and recoverable fatal errors have been converted to exceptions in PHP 7. These error exceptions inherit from the Error class, which itself implements the Throwable interface (the new base interface all exceptions inherit).

This means that custom error handlers may no longer be triggered because exceptions may be thrown instead (causing new fatal errors for uncaught Error exceptions).

A fuller description of how errors operate in PHP 7 can be found on the PHP 7 errors page. This migration guide will merely enumerate the changes that affect backward compatibility.

set_exception_handler() is no longer guaranteed to receive Exception objects ¶

Code that implements an exception handler registered with set_exception_handler() using a type declaration of Exception will cause a fatal error when an Error object is thrown.

If the handler needs to work on both PHP 5 and 7, you should remove the type declaration from the handler, while code that is being migrated to work on PHP 7 exclusively can simply replace the Exception type declaration with Throwable instead.

```
<?php
// PHP 5 era code that will break.
function handler(Exception $e) { ... }
set_exception_handler('handler');

// PHP 5 and 7 compatible.
function handler($e) { ... }

// PHP 7 only.
function handler(Throwable $e) { ... }
?>
```

Internal constructors always throw exceptions on failure ¶

Previously, some internal classes would return `null` or an unusable object when the constructor failed. All internal classes will now throw an Exception in this case in the same way that user classes already had to.

Parse errors throw ParseError ¶

Parser errors now throw a ParseError object. Error handling for eval() should now include a `catch` block that can handle this error.

E_STRICT notices severity changes ¶

All of the `E_STRICT` notices have been reclassified to other levels. `E_STRICT` constant is retained, so calls like `error_reporting(E_ALL|E_STRICT)` will not cause an error.

`E_STRICT` notices severity changes

Situation	New level/behaviour
Indexing by a resource	`E_NOTICE`
Abstract static methods	Notice removed, triggers no error
"Redefining" a constructor	Notice removed, triggers no error
Signature mismatch during inheritance	`E_WARNING`
Same (compatible) property in two used traits	Notice removed, triggers no error
Accessing static property non-statically	`E_NOTICE`
Only variables should be assigned by reference	`E_NOTICE`
Only variables should be passed by reference	`E_NOTICE`
Calling non-static methods statically	`E_DEPRECATED`

Changes to variable handling ¶

PHP 7 now uses an abstract syntax tree when parsing source files. This has permitted many improvements to the language which were previously impossible due to limitations in the parser used in earlier versions of PHP, but has resulted in the removal of a few special cases for consistency reasons, which has resulted in backward compatibility breaks. These cases are detailed in this section.

Changes to the handling of indirect variables, properties, and methods ¶

Indirect access to variables, properties, and methods will now be evaluated strictly in left-to-right order, as opposed to the previous mix of special cases. The table below shows how the order of evaluation has changed.

Old and new evaluation of indirect expressions

Expression	PHP 5 interpretation	PHP 7 interpretation
`$$foo['bar']['baz']`	`${$foo['bar']['baz']}`	`($$foo)['bar']['baz']`
`$foo->$bar['baz']`	`$foo->{$bar['baz']}`	`($foo->$bar)['baz']`
`$foo->$bar['baz']()`	`$foo->{$bar['baz']}()`	`($foo->$bar)['baz']()`
`Foo::$bar['baz']()`	`Foo::{$bar['baz']}()`	`(Foo::$bar)['baz']()`

Code that used the old right-to-left evaluation order must be rewritten to explicitly use that evaluation order with curly braces (see the above middle column). This will make the code both forwards compatible with PHP 7.x and backwards compatible with PHP 5.x.

This also affects the `global` keyword. The curly brace syntax can be used to emulate the previous behaviour if required:

```php
<?php
function f() {
    // Valid in PHP 5 only.
    global $$foo->bar;

    // Valid in PHP 5 and 7.
    global ${$foo->bar};
}
?>
```

Changes to list() handling ¶

list() no longer assigns variables in reverse order

list() will now assign values to variables in the order they are defined, rather than reverse order. In general, this only affects the case where list() is being used in conjunction with the array `[]` operator, as shown below:

```php
<?php
list($a[], $a[], $a[]) = [1, 2, 3];
var_dump($a);
?>
```

Output of the above example in PHP 5:

```
array(3) {
  [0]=>
  int(3)
  [1]=>
  int(2)
  [2]=>
  int(1)
}
```

Output of the above example in PHP 7:

```
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  int(3)
}
```

In general, it is recommended not to rely on the order in which list() assignments occur, as this is an implementation detail that may change again in the future.

Empty list() assignments have been removed

list() constructs can no longer be empty. The following are no longer allowed:

```php
<?php
list() = $a;
list(,,) = $a;
list($x, list(), $y) = $a;
?>
```

list() cannot unpack strings

list() can no longer unpack string variables. str_split() should be used instead.

Array ordering when elements are automatically created during by reference assignments has changed ¶

The order of the elements in an array has changed when those elements have been automatically created by referencing them in a by reference assignment. For example:

```php
<?php
$array = [];
$array["a"] =& $array["b"];
$array["b"] = 1;
var_dump($array);
?>
```

Output of the above example in PHP 5:

```
array(2) {
  ["b"]=>
  &int(1)
  ["a"]=>
  &int(1)
}
```

Output of the above example in PHP 7:

```
array(2) {
  ["a"]=>
  &int(1)
  ["b"]=>
  &int(1)
}
```

Parentheses around function arguments no longer affect behaviour ¶

In PHP 5, using redundant parentheses around a function argument could quiet strict standards warnings when the function argument was passed by reference. The warning will now always be issued.

```php
<?php
function getArray() {
    return [1, 2, 3];
}

function squareArray(array &$a) {
    foreach ($a as &$v) {
        $v **= 2;
    }
}

// Generates a warning in PHP 7.
squareArray((getArray()));
?>
```

The above example will output:

```
Notice: Only variables should be passed by reference in /tmp/test.php on line 13
```

Changes to foreach ¶

Minor changes have been made to the behaviour of the foreach control structure, primarily around the handling of the internal array pointer and modification of the array being iterated over.

foreach no longer changes the internal array pointer ¶

Prior to PHP 7, the internal array pointer was modified while an array was being iterated over with foreach. This is no longer the case, as shown in the following example:

```
<?php
$array = [0, 1, 2];
foreach ($array as &$val) {
    var_dump(current($array));
}
?>
```

Output of the above example in PHP 5:

```
int(1)
int(2)
bool(false)
```

Output of the above example in PHP 7:

```
int(0)
int(0)
int(0)
```

foreach by-value operates on a copy of the array ¶

When used in the default by-value mode, foreach will now operate on a copy of the array being iterated rather than the array itself. This means that changes to the array made during iteration will not affect the values that are iterated.

foreach by-reference has improved iteration behaviour ¶

When iterating by-reference, foreach will now do a better job of tracking changes to the array made during iteration. For example, appending to an array while iterating will now result in the appended values being iterated over as well:

```
<?php
$array = [0];
foreach ($array as &$val) {
    var_dump($val);
    $array[1] = 1;
}
?>
```

Output of the above example in PHP 5:

```
int(0)
```

Output of the above example in PHP 7:

```
int(0)
int(1)
```

Iteration of non-Traversable objects ¶

Iterating over a non-Traversable object will now have the same behaviour as iterating over by-reference arrays. This results in the improved behaviour when modifying an array during iteration also being applied when properties are added to or removed from the object.

Changes to int handling ¶

Invalid octal literals ¶

Previously, octal literals that contained invalid numbers were silently truncated (`0128` was taken as `012`). Now, an invalid octal literal will cause a parse error.

Negative bitshifts ¶

Bitwise shifts by negative numbers will now throw an ArithmeticError:

```
<?php
var_dump(1 >> -1);
?>
```

Output of the above example in PHP 5:

```
int(0)
```

Output of the above example in PHP 7:

```
Fatal error: Uncaught ArithmeticError: Bit shift by negative number in /tmp/test.php:2
Stack trace:
#0 {main}
  thrown in /tmp/test.php on line 2
```

Out of range bitshifts ¶

Bitwise shifts (in either direction) beyond the bit width of an int will always result in 0. Previously, the behaviour of such shifts was architecture dependent.

Changes to Division By Zero ¶

Previously, when 0 was used as the divisor for either the divide (/) or modulus (%) operators, an E_WARNING would be emitted and **`false`** would be returned. Now, the divide operator returns a float as either +INF, -INF, or NAN, as specified by IEEE 754. The modulus operator E_WARNING has been removed and will throw a DivisionByZeroError exception.

```
<?php
var_dump(3/0);
var_dump(0/0);
var_dump(0%0);
?>
```

Output of the above example in PHP 5:

```
Warning: Division by zero in %s on line %d
bool(false)

Warning: Division by zero in %s on line %d
bool(false)

Warning: Division by zero in %s on line %d
bool(false)
```

Output of the above example in PHP 7:

```
Warning: Division by zero in %s on line %d
float(INF)

Warning: Division by zero in %s on line %d
float(NAN)

PHP Fatal error:  Uncaught DivisionByZeroError: Modulo by zero in %s line %d
```

Changes to string handling ¶

Hexadecimal strings are no longer considered numeric ¶

Strings containing hexadecimal numbers are no longer considered to be numeric. For example:

```
<?php
var_dump("0x123" == "291");
var_dump(is_numeric("0x123"));
var_dump("0xe" + "0x1");
var_dump(substr("foo", "0x1"));
?>
```

Output of the above example in PHP 5:

```
bool(true)
bool(true)
int(15)
string(2) "oo"
```

Output of the above example in PHP 7:

```
bool(false)
bool(false)
int(0)

Notice: A non well formed numeric value encountered in /tmp/test.php on line 5
string(3) "foo"
```

filter_var() can be used to check if a string contains a hexadecimal number, and also to convert a string of that type to an int:

```
<?php
$str = "0xffff";
$int = filter_var($str, FILTER_VALIDATE_INT, FILTER_FLAG_ALLOW_HEX);
if (false === $int) {
    throw new Exception("Invalid integer!");
}
var_dump($int); // int(65535)
?>
```

`\u{` may cause errors ¶

Due to the addition of the new Unicode codepoint escape syntax, strings containing a literal `\u{` followed by an invalid sequence will cause a fatal error. To avoid this, the leading backslash should be escaped.

Removed functions ¶

call_user_method() and call_user_method_array() ¶

These functions were deprecated in PHP 4.1.0 in favour of call_user_func() and call_user_func_array(). You may also want to consider using variable functions and/or the ... operator.

All ereg* functions ¶

All ereg functions were removed. PCRE is a recommended alternative.

mcrypt aliases ¶

The deprecated mcrypt_generic_end() function has been removed in favour of mcrypt_generic_deinit().

Additionally, the deprecated mcrypt_ecb(), mcrypt_cbc(), mcrypt_cfb() and mcrypt_ofb() functions have been removed in favour of using mcrypt_decrypt() with the appropriate `MCRYPT_MODE_*` constant.

All ext/mysql functions ¶

All ext/mysql functions were removed. For details about choosing a different MySQL API, see Choosing a MySQL API.

All ext/mssql functions ¶

All `ext/mssql` functions were removed.

- PDO_SQLSRV
- PDO_ODBC
- SQLSRV
- Unified ODBC API

intl aliases ¶

The deprecated **datefmt_set_timezone_id()** and **IntlDateFormatter::setTimeZoneID()** aliases have been removed in favour of datefmt_set_timezone() and IntlDateFormatter::setTimeZone(), respectively.

set_magic_quotes_runtime() ¶

set_magic_quotes_runtime(), along with its alias **magic_quotes_runtime()**, have been removed. They were deprecated in PHP 5.3.0, and became effectively non-functional with the removal of magic quotes in PHP 5.4.0.

set_socket_blocking() ¶

The deprecated **set_socket_blocking()** alias has been removed in favour of stream_set_blocking().

dl() in PHP-FPM ¶

dl() can no longer be used in PHP-FPM. It remains functional in the CLI and embed SAPIs.

GD Type1 functions ¶

Support for PostScript Type1 fonts has been removed from the GD extension, resulting in the removal of the following functions:

- **imagepsbbox()**
- **imagepsencodefont()**
- **imagepsextendfont()**
- **imagepsfreefont()**
- **imagepsloadfont()**
- **imagepsslantfont()**
- **imagepstext()**

Using TrueType fonts and their associated functions is recommended instead.

Removed INI directives ¶

Removed features ¶

The following INI directives have been removed as their associated features have also been removed:

- `always_populate_raw_post_data`
- `asp_tags`

`xsl.security_prefs` ¶

The `xsl.security_prefs` directive has been removed. Instead, the XsltProcessor::setSecurityPrefs() method should be called to control the security preferences on a per-processor basis.

Other backward incompatible changes ¶

New objects cannot be assigned by reference ¶

The result of the new statement can no longer be assigned to a variable by reference:

```
<?php
class C {}
$c =& new C;
?>
```

Output of the above example in PHP 5:

```
Deprecated: Assigning the return value of new by reference is deprecated in /tmp/test.php on line 3
```

Output of the above example in PHP 7:

```
Parse error: syntax error, unexpected 'new' (T_NEW) in /tmp/test.php on line 3
```

Invalid class, interface and trait names ¶

The following names cannot be used to name classes, interfaces or traits:

- bool
- int
- float
- string
- **null**
- **true**
- **false**

Furthermore, the following names should not be used. Although they will not generate an error in PHP 7.0, they are reserved for future use and should be considered deprecated.

- resource
- object
- mixed
- numeric

ASP and script PHP tags removed ¶

Support for using ASP and script tags to delimit PHP code has been removed. The affected tags are:

Removed ASP and script tags

Opening tag	Closing tag
`<%`	`%>`
`<%=`	`%>`
`<script language="php">`	`</script>`

Calls from incompatible context removed ¶

Previously deprecated in PHP 5.6, static calls made to a non-static method with an incompatible context will now result in the called method having an undefined `$this` variable and a deprecation warning being issued.

```
<?php
class A {
    public function test() { var_dump($this); }
}

// Note: Does NOT extend A
class B {
    public function callNonStaticMethodOfA() { A::test(); }
}

(new B)->callNonStaticMethodOfA();
?>
```

Output of the above example in PHP 5.6:

```
Deprecated: Non-static method A::test() should not be called statically, assuming $this from incompatible context in /tmp/test.php on line 8
object(B)#1 (0) {
}
```

Output of the above example in PHP 7:

```
Deprecated: Non-static method A::test() should not be called statically in /tmp/test.php on line 8

Notice: Undefined variable: this in /tmp/test.php on line 3
NULL
```

yield is now a right associative operator ¶

The yield construct no longer requires parentheses, and has been changed to a right associative operator with precedence between `print` and `=>`. This can result in changed behaviour:

```
<?php
echo yield -1;
// Was previously interpreted as
echo (yield) - 1;
// And is now interpreted as
echo yield (-1);

yield $foo or die;
// Was previously interpreted as
yield ($foo or die);
// And is now interpreted as
(yield $foo) or die;
?>
```

Parentheses can be used to disambiguate those cases.

Functions cannot have multiple parameters with the same name ¶

It is no longer possible to define two or more function parameters with the same name. For example, the following function will trigger an **`E_COMPILE_ERROR`**:

```
<?php
function foo($a, $b, $unused, $unused) {
    //
}
?>
```

Functions inspecting arguments report the *current* parameter value ¶

func_get_arg(), func_get_args(), debug_backtrace() and exception backtraces will no longer report the original value that was passed to a parameter, but will instead provide the current value (which might have been modified).

```
<?php
function foo($x) {
    $x++;
    var_dump(func_get_arg(0));
}
foo(1);?>
```

Output of the above example in PHP 5:

```
1
```

Output of the above example in PHP 7:

```
2
```

Switch statements cannot have multiple default blocks ¶

It is no longer possible to define two or more default blocks in a switch statement. For example, the following switch statement will trigger an **`E_COMPILE_ERROR`**:

```
<?php
switch (1) {
    default:
    break;
    default:
    break;
}
?>
```

$HTTP_RAW_POST_DATA removed ¶

$HTTP_RAW_POST_DATA is no longer available. The `php://input` stream should be used instead.

`#` comments in INI files removed ¶

Support for prefixing comments with `#` in INI files has been removed. `;` (semi-colon) should be used instead. This change applies to *php.ini*, as well as files handled by parse_ini_file() and parse_ini_string().

JSON extension replaced with JSOND ¶

The JSON extension has been replaced with JSOND, causing three minor BC breaks. Firstly, a number must not end in a decimal point (i.e. `34.` must be changed to either `34.0` or `34`). Secondly, when using scientific notation, the `e` exponent must not immediately follow a decimal point (i.e. `3.e3` must be changed to either `3.0e3` or `3e3`). Finally, an empty string is no longer considered valid JSON.

Internal function failure on overflow ¶

Previously, internal functions would silently truncate numbers produced from float-to-integer coercions when the float was too large to represent as an integer. Now, an E_WARNING will be emitted and **`null`** will be returned.

Fixes to custom session handler return values ¶

Any predicate functions implemented by custom session handlers that return either **`false`** or `-1` will be fatal errors. If any value from these functions other than a boolean, `-1`, or `0` is returned, then it will fail and an E_WARNING will be emitted.

Sort order of equal elements ¶

The internal sorting algorithm has been improved, what may result in different sort order of elements, which compare as equal, than before.

> **Note**:
>
> Don't rely on the order of elements which compare as equal; it might change anytime.

Misplaced break and switch statements ¶

`break` and `continue` statements outside of a loop or `switch` control structure are now detected at compile-time instead of run-time as before, and trigger an **`E_COMPILE_ERROR`**.

Mhash is not an extension anymore ¶

The Mhash extension has been fully integrated into the Hash extension. Therefore, it is no longer possible to detect Mhash support with extension_loaded(); use function_exists() instead. Furthermore, Mhash is no longer reported by get_loaded_extensions() and related features.

declare(ticks) ¶

The declare(ticks) directive does no longer leak into different compilation units.

add a note

User Contributed Notes 10 notes

up
down
181
me at fquff dot io ¶
5 years ago
```
[Editor's note: fixed limit on user request]

As a mathematician, 3/0 == +INF IS JUST WRONG. You can't just assume 3/0 == lim_{x->0+} 3/x, which is +INF indeed, because division IS NOT
A CONTINUOUS FUNCTION in x == 0.

Also, 3/0 == +INF ("positive" infinity) while -3/0 == -INF ("negative" infinity) requires the assumption that 0 is a positive number,
which is just as illogical as it looks like.

The fact that a warning is emitted is good, but it should definitely equals to NaN. ±INF is just illogical (and arithmetically wrong).

Except for this "detail", looks an amazing update, can't wait to test it even further!

Cheers,
P.
```
up
down
103
mossy2100 ¶
5 years ago
```
Although $x/0 is technically not infinity in a purely mathematical sense, when you understand why the IEEE float includes a value for
infinity, and returns infinity in this case, it makes sense that PHP would agree with this.

The reason is that programmers don't usually divide by 0 on purpose. A value of 0 as a divisor usually occurs due to underflow, i.e. a
value which is too small to be represented as a float. So, for example, if you have values like:
$x = 1;
$y = 1e-15 * 1e-15;
$z = $x/$y;
Because $y will have underflowed to 0, the division operation will throw the division by zero warning, and $z will be set to INF. In a
better computer, however, $y would not have the value 0 (it would be 1e-30) and $z would not have the value INF (it would be 1e30).

In other words, 0 is not only representative of an actual 0, but also a very small number which float cannot represent correctly
(underflow), and INF does not only represent infinity, but also a very big number which float cannot represent correctly (overflow). We do
the best we can within the limitations of floating point values, which are really just good approximations of the actual numbers being
represented.

What does bother me is that division by zero is handled in two different ways depending on the operator. I would have preferred the new
DivisionByZeroError exception to be thrown in all cases, for consistency and to enforce good programming practices.
```
up
down
126
tuxedobob ¶
5 years ago
```
As a programmer, I don't care whether 3/0 is INF or NaN. Both answers are (probably) equally useless, and tell me that something somewhere
else is screwed up.
```

up
down
37
opitz dot alexander at googlemail dot com ¶
5 years ago

```
About the division by zero, please see discussion to IEEEE 754
For example: http://stackoverflow.com/questions/14682005/why-does-division-by-zero-in-ieee754-standard-results-in-infinite-value
```

up
down
17
tkondrashov at gmail dot com ¶
2 years ago

```
split() was also removed in 7.0, so be sure to check your old code for it as well as the functions listed in this doc
```

up
down
7
maba at mb-systemhaus dot net ¶
4 years ago

```
NOTE:
the new variable handling in PHP 7 also has side effects on the COM .NET extension. Numeric variants (in the Windows application space)
now must be quoted when passed over from PHP. This is the case when the receiving application expects a variable of type variant.

Here is an example:

<?php
  $word = new COM('Word.Application');

  // now load a document, ...

  // the following works in PHP 5 but will throw an exception in PHP 7
  $word->ActiveDocument->PrintOut(false, false, 0, $outfile);

  // the following works in PHP 7 as well, please note the quotes around the '0'
  $word->ActiveDocument->PrintOut(false, false, '0', $outfile);
?>
```

up
down
12
Frank ¶
4 years ago

```
[Editor's Note: that change is listed in the "Changed functions" section.]

The substr function has also been changed in a backward incompatible way.

<?php
substr("test",4);  # false in PHP 5,  "" in PHP 7
?>

In fact, this is the only thing we had to change in a number of places for our code base to run on PHP 7. It's definitely an improvement
though, as the old behavior tended to cause bugs in border cases.
```

up
down
1
ilya dot chase at yandex dot ru ¶
1 year ago

```
Take note that in preg_replace() function, flag '\e' was deleted in PHP 7.0.0. This function will return null always with this flag. Doc:
https://www.php.net/manual/ru/function.preg-replace.php
```

up
down
0
viktor dot csiky at nospam dot nospam dot eu ¶
4 years ago

```
It is stated:

"foreach by-value operates on a copy of the array

When used in the default by-value mode, foreach will now operate on a copy of the array being iterated rather than the array itself. This
means that changes to the array made during iteration will not affect the values that are iterated."

Please note that this is not exactly true. New foreach operates on a copy of the array, by-value or by-reference. It seems that in the
latter case, the array copy is simply moved over (to) the original array before it is presumably destroyed.

As a consequence of this, you may not "dereference" an array containing values - e.g. for use with ReflectionMethod::invokeArgs() or the
good ole' call_user_func().
Consider the snippet below:

<?php
function deref(Array $inputArray)
{
        $retVal = [];
        foreach ($inputArray as &$inputValue)
        {
            $retVal[] = $inputValue;
        }

        return $retVal;
}
?>

As of PHP 7.0, this *will no longer work*. You will get the usual suspect:

PHP Warning:  Parameter n to whatever() expected to be a reference, value given in baz.php on line x

You need to convert it to explicitly reference the original array:
<?php
function deref(Array $inputArray)
{
        $retVal = [];

        foreach ($inputArray as $inputKey => $inputValue)
        {
            $retVal[$inputKey] = &$inputArray[$inputKey];
        }

        return $retVal;
}
?>

PLEASE NOTE that this might have the unforeseen consequence of your code not working anymore in php versions less than 5.3 (that is, 5.2
and below).
```

up
down
-12
Ray.Paseur sometimes uses Gmail ¶
4 years ago

```
In the section captioned "Changes to the handling of indirect variables, properties, and methods" there are parentheses used in the table
directly beneath "PHP 7 interpretation."

The parentheses are intended to show the evaluation order, but they are not part of the syntax, and should not be used in the variable
definition or reference.  This juxtaposition confused one of my colleagues; hopefully this note will save someone else some time.

Examples of the correct curly-brace syntax is further down the page, in the section captioned "global only accepts simple variables."
```

add a note