Backward Incompatible Changes

- PHP Core
  - $GLOBALS Access Restrictions
  - Usage of static Variables in Inherited Methods
  - Optional parameters specified before required parameters
  - Return Type Compatibility with Internal Classes
  - New Keywords
- Resource to Object Migration
- MySQLi
- MySQLnd
- OpenSSL
- PHP Data Objects
  - MySQL Driver
  - SQLite Driver
- Phar
- Standard
- Standard PHP Library (SPL)

PHP Core

$GLOBALS Access Restrictions

Access to the `$GLOBALS` array is now subject to a number of restrictions. Read and write access to individual array elements like `$GLOBALS['var']` continues to work as-is. Read-only access to the entire `$GLOBALS` array also continues to be supported. However, write access to the entire `$GLOBALS`

array is no longer supported. For example, `array_pop($GLOBALS)` will result in an error.

Usage of static Variables in Inherited Methods

When a method using static variables is inherited (but not overridden), the inherited method will now share static variables with the parent method.

```
<?php
class A {
    public static function counter() {
        static $counter = 0;
        $counter++;
        return $counter;
    }
}
class B extends A {}
var_dump(A::counter()); // int(1)
var_dump(A::counter()); // int(2)
var_dump(B::counter()); // int(3), previously int(1)
var_dump(B::counter()); // int(4), previously int(2)
?>
```

This means that static variables in methods now behave the same way as static properties.

Optional parameters specified before required parameters

An optional parameter specified before required parameters is now always treated as required, even when called using named arguments. As of PHP 8.0.0, but prior to PHP 8.1.0, the below emits a deprecation notice on the definition, but runs successfully when called. As of PHP 8.1.0, an error of class `ArgumentCountError` is thrown, as it would be when called with positional arguments.

```
<?php
function makeyogurt($container = "bowl", $flavour)
{
    return "Making a $container of $flavour yogurt.\n";
}
try
{
    echo makeyogurt(flavour: "raspberry");
}
catch (Error $e)
{
    echo get_class($e), ' - ', $e->getMessage(), "\n";
}
?>
```

ENTITY

```
Deprecated: Required parameter $flavour follows optional parameter $container
 in example.php on line 3
Making a bowl of raspberry yogurt.
```

ENTITY

```
Deprecated: Optional parameter $container declared before required parameter
 $flavour is implicitly treated as a required parameter in example.php on line 3
ArgumentCountError - makeyogurt(): Argument #1 ($container) not passed
```

Note that a default value of ENTITY can be used before required parameters to specify a nullable type, but the parameter will still be required.

Return Type Compatibility with Internal Classes

Most non-final internal methods now require overriding methods to declare a compatible return type, otherwise a deprecated notice is emitted during inheritance validation. In case the return type cannot be declared for an overriding method due to PHP cross-version compatibility concerns,

a `ReturnTypeWillChange` attribute can be added to silence the deprecation notice.

New Keywords

`readonly` is a keyword now. However, it still may be used as function name.

`never` is now a reserved word, so it cannot be used to name a class, interface or trait, and is also prohibited from being used in namespaces.

Resource to Object Migration

Several ENTITYs have been migrated to ENTITYs. Return value checks using `is_resource` should be replaced with checks for ENTITY.

- The FileInfo functions now accept and return `finfo` objects instead of `fileinfo` ENTITYs.
- The FTP functions now accept and return `FTP\Connection` objects instead of `ftp` ENTITYs.
- The IMAP functions now accept and return `IMAP\Connection` objects instead of `imap` ENTITYs.
- The LDAP functions now accept and return `LDAP\Connection` objects instead of `ldap link` ENTITYs.
- The LDAP functions now accept and return `LDAP\Result` objects instead of `ldap result` ENTITYs.
- The LDAP functions now accept and return `LDAP\ResultEntry` objects instead of `ldap result entry` ENTITYs.
- The PgSQL functions now accept and return `PgSql\Connection` objects instead of `pgsql link` ENTITYs.
- The PgSQL functions now accept and return `PgSql\Result` objects instead of `pgsql result` ENTITYs.
- The PgSQL functions now accept and return `PgSql\Lob` objects instead of `pgsql large object` ENTITYs.
- The PSpell functions now accept and return `PSpell\Dictionary` objects instead of `pspell` ENTITYs.
- The PSpell functions now accept and return `PSpell\Config` objects instead of `pspell config` ENTITYs.

MySQLi

`mysqli_fetch_fields`, and `mysqli_fetch_field_direct` will now always return `0` for the max_length. This information can be computed by iterating over the result set, and taking the maximum length. This is what PHP was doing internally previously.

The `MYSQLI_STMT_ATTR_UPDATE_MAX_LENGTH` option no longer has any effect.

The `MYSQLI_STORE_RESULT_COPY_DATA` option no longer has any effect. Passing any value to the `mode` parameter of mysqli::store_result no longer has any effect.

mysqli::connect now returns ENTITY instead of ENTITY on success.

The default error handling mode has been changed from "silent" to "exceptions" See the MySQLi reporting mode page for more details on what this entails, and how to explicitly set this attribute. To restore the previous behaviour use: `mysqli_report(MYSQLI_REPORT_OFF);`

Classes extending mysqli_stmt::execute are now required to specify the additional optional parameter.

MySQLnd

The mysqlnd.fetch_data_copy INI directive has been removed. This should not result in user-visible behavior changes.

OpenSSL

EC private keys will now be exported in PKCS#8 format rather than traditional format, just like all other keys.

`openssl_pkcs7_encrypt` and `openssl_cms_encrypt` will now default to using AES-128-CBC rather than RC2-40. The RC2-40 cipher is considered insecure and not enabled by default by OpenSSL 3.

PHP Data Objects

`PDO::ATTR_STRINGIFY_FETCHES` now stringifies values of type ENTITY to `"0"` or `"1"`. Previously ENTITYs were not stringified.

Calling PDOStatement::bindColumn with `PDO::PARAM_LOB` will now constantly bind a stream result when `PDO::ATTR_STRINGIFY_FETCHES` is not enabled. Previously, the result would either be a stream or

a string depending on the used database driver and the time the binding is performed.

MySQL Driver

Integers and floats in result sets will now be returned using native PHP types instead of ENTITYs when using emulated prepared statements. This matches the behavior of native prepared statements. The previous behaviour can be restored by enabling the `PDO::ATTR_STRINGIFY_FETCHES` option.

SQLite Driver

Integers and floats in results sets will now be returned using native PHP types. The previous behaviour can be restored by enabling the `PDO::ATTR_STRINGIFY_FETCHES` option.

Phar

To comply with the ArrayAccess interface, Phar::offsetUnset and PharData::offsetUnset no longer return a ENTITY.

Standard

`version_compare` no longer accepts undocumented operator abbreviations.

`htmlspecialchars`, `htmlentities`, `htmlspecialchars_decode`, `html_entity_decode`, and `get_html_translation_table` now use `ENT_QUOTES | ENT_SUBSTITUTE` rather than `ENT_COMPAT` by default. This means that `'` is escaped to `ENTITY#039;` while previously nothing was done. Additionally, malformed UTF-8 will be replaced by a Unicode substitution character, instead of resulting in an empty string.

`debug_zval_dump` now prints the refcount of a reference wrappers with their refcount, instead of only prepending `ENTITY` to the value. This more accurately models reference representation since PHP 7.0.

`debug_zval_dump` now prints `interned` instead of a dummy refcount for interned strings and immutable arrays.

Standard PHP Library (SPL)

`SplFixedArray`, will now be JSON encoded like an ENTITY

□□□ #2
□ □□□ □□□ 2026-10-31 17:35:14 UTC
□ □□□ □□□ 2026-10-31 17:35:16 UTC