Why the code has no type declarations

An explanation of one rule that surprises PHP developers - Exponential code is written without scalar, return and property types and without strict_types - what is allowed instead, how types are documented and checked...

Source file: doc/eng-us/help/php-developers/why-no-type-declarations.md Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/php-developers/why-no-type-declarations.md

This page is for PHP developers who come from code bases where `declare( strict_types=1 )` and typed signatures are the norm. In Exponential they are not allowed in new code. The rule is deliberate; this page explains it, so you can follow it without fighting it.

Contents

1. The rule
2. Why
3. How types are documented instead
4. How types are still checked
5. The exceptions
6. Common questions

The rule

The sniff `bin/phpcs/Exponential/Sniffs/TypeDeclarations/NoTypeDeclarationsSniff.php` reports, each with its own code:

Code	Example it reports
`StrictTypes`	`declare( strict_types=1 );`
`ParameterType`	`function foo( int $a )`, `function foo( string|array $a )`, also promoted constructor parameters
`ReturnType`	`function foo(): array`
`PropertyType`	`public string $name;`
`ConstantType`	`const string NAME = '...';`

Allowed, because PHP 5 had them and the kernel uses them throughout: `array`, `callable`, `self` and class or interface names, also nullable:

PHP

```php
public function render( array $items, ?eZINI $ini = null, ?callable $filter = null )
```

Why

1. **Consistency with the existing code.** The 6.0.15 kernel and libraries are written without these declarations. A file that mixes two styles is harder to read and review than either style alone.
2. **Behavior does not change by accident.** Adding `int` to a parameter in code that has always received numeric strings, `null` or `false` turns a working call into a `TypeError`. In a code base with thousands of callers, including extensions the project has never seen, that risk is real; the kernel's own PHP 8 work shows how many values arrive as `null` or strings (PHP 8 migration notes for your code).
3. **Signatures stay compatible.** Extensions subclass kernel classes (`eZDataType`, `eZPersistentObject`, `eZWorkflowEventType`). A return type added to a kernel method forces every subclass to add it too, or the subclass no longer loads.
4. **PHP 8.0.** Several declaration forms (`never`, intersection types, typed constants, `readonly`) do not exist on 8.0, the oldest supported version (7.4 is usable without support from 6.0.15).

How types are documented instead

Every class and public method carries a docblock with `@param` and `@return`, and properties with `@var`:

PHP

```php
/**
 * The items of one user, newest first.
 *
 * @param int $userID
 * @param int $limit at most 100
 * @return acmeReportItem[]
 */
public static function fetchByUser( $userID, $limit = 20 )
{
    $userID = (int)$userID;
    $limit = max( 1, min( 100, (int)$limit ) );
    ...
}
```

Note the casts at the top of the method: input is converted where it enters, which is the kernel's way of making a value the type it should be (Validating input and casting).

How types are still checked

- **PHPStan** reads the docblocks. `phpstan.neon.dist` sets `treatPhpDocTypesAsCertain: false`, so a docblock type guides the analysis without turning every defensive check into an error (Static analysis with PHPStan).
- **Tests** assert the values, not only the types (PHPUnit without a database).
- **Casts and checks** at the boundary: `(int)` for ids, allow lists for choices, `is_array()` before `count()`.

The exceptions

Where	Allowed
files under `tests/`	`: void` on `setUp()`, `tearDown()` and the other PHPUnit template methods, which PHPUnit requires
methods implementing SPL interfaces	no return type, but the `#[\ReturnTypeWillChange]` attribute
third-party code in `vendor/`	whatever its authors wrote; it is not checked
an extension with its own standard	the rule applies to code that follows the kernel's standard; an extension may declare its own `phpcs.xml`

Common questions

Can I disable one code of the sniff for my extension? Yes: each report has its own code, so a ruleset can exclude one. The kernel does not.

Will the rule change? The standard describes the code as it is. Raising it would be a project decision taken in small steps, like raising the PHPStan level; until then, new kernel code follows it.

Related pages

- The coding standard (phpcs.xml.dist)
- Coding standards
- Quality checks
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

> **Become an Exponential Partner**
>
> Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.
>
> Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.