

# Cool new things in PHP



“While I wasn't paying attention,  
PHP got quite good” [\[1\]](#)

01

# What's already available

added in PHP  $\leq 7.4$ , or polyfilled by MediaWiki

# yield

```
public function provideValues(): iterable {
    yield 'null' => [ null ];
    yield 'empty string' => [ '' ];
    foreach ( [ 0, 1, -1 ] as $number ) {
        yield "int $number" => [ $number ];
        yield "string $number" => [ (string)$number ];
    }
}

/** @dataProvider provideValues */
public function testSomething( $value ): void { /* ... */ }
```

# yield

- [generator syntax](#)
- returns an iterable object
- extremely useful for PHPUnit data providers

• • •

```
public function msg( string $msg, ...$args ): Message {  
    return wfMessage( $msg, ...$args )  
        ->inLanguage( $this->getTargetLanguage() )  
        ->page( $this->getPage() );  
}
```

```
$array1 = [ 2, 3, 4 ];
```

```
$array2 = [ 0, 1, ...$array1, 5 ];
```

```
// [ 0, 1, 2, 3, 4, 5 ]
```

includes/parser/Parser.php,  
as of commit a4da635e8a,  
GPL-2.0-or-later

- • •

- you can mostly forget about `func_get_args()` and `call_user_func()` / `call_user_func_array()`
- writing functions that wrap other functions, passing through all their arguments, is much easier now 😊
- spreading in arrays is neat too I guess

# array destructuring

```
[ $causeAction, $causeAgent ] = $this->getCause();
```

```
[ , $lag, $index ] = $this->loadBalancer->getMaxLag();
```

```
[
```

```
    'tables' => $tables,
```

```
    'fields' => $fields,
```

```
    'joins' => $joins,
```

```
] = $revStore->getQueryInfo();
```

includes/Storage/DerivedPageDataUpdater.php,

includes/api/ApiQuerySiteinfo.php,  
as of commit a4da635e8a,  
GPL-2.0-or-later

# array destructuring

- forget about `list()`, just like you forgot about `array()` – both are `[]` now
- we can also destructure associative arrays now



# static types / type declarations

```
class CreditsAction extends FormlessAction {
```

includes/actions/CreditsAction.php (edited),

```
    /** @var LinkRenderer */
```

```
    private $linkRenderer;
```

```
    /** @var UserFactory */
```

```
    private $userFactory;
```

```
    /**
```

includes/Message/Converter.php (edited),

as of commit c40084e898,

GPL-2.0-or-later

```
    * Convert a Message to a MessageValue
```

```
    * @param Message $m
```

```
    * @return MessageValue
```

```
    */
```

```
    public function convertMessage( $m ) {
```



# static types / type declarations

```
class CreditsAction extends FormlessAction {
```

includes/actions/CreditsAction.php (edited),

```
    private LinkRenderer $linkRenderer;
```

```
    private UserFactory $userFactory;
```

```
/** Convert a Message to a MessageValue */
```

includes/Message/Converter.php (edited),

```
public function convertMessage( Message $m ): MessageValue {
```

as of commit c40084e898,

GPL-2.0-or-later



# static types / type declarations



Caty Shadows 🇺🇸 IPsec

@maris

🔒 12h

rust has ruined me, I want algebraic type systems everywhere



1+



nota 🐟 ✨

@nota@chaos.social

@maris This Unit Test Could Have Been A Type Error

May 15, 2023, 21:41 · 🔒 · 🔄 11 · ★ 33



# static types / type declarations

- type declarations, unlike PHPdoc comments, get checked at runtime and are guaranteed to be correct
  - [// sometimes I believe compiler ignores all my comments](#)
- mostly obsoletes `Assert::parameterType( 'MyType' , $x, '$x' );`
- if the doc comment has no information beyond the types, it's redundant and can be left out altogether

# strict types

```
<?php // a.php
```

```
function f( string $a ) {  
    var_dump( $a );  
}
```

```
<?php // b.php
```

```
require_once __DIR__ . '/a.php';  
f( 1 );  
  
# logs string(1) "1"
```

# strict types

```
<?php // a.php
```

```
function f( string $a ) {  
    var_dump( $a );  
}
```

```
<?php // b.php
```

```
declare( strict_types = 1 );
```

```
require_once __DIR__ . '/a.php';
```

```
f( 1 );
```

```
# TypeError: f(): Argument #1 ($a) must be of type string, int given
```

# strict types

- without strict types, PHP attempts to cast arguments to the expected parameter type
- if you would rather have an `TypeError`, declare strict types
- note that the strict types declaration affects the *caller*, not the *callee*
- in other words, at the callee, a type declaration guarantees that the parameter now has the expected type, but it might have been cast to that type depending on the caller's argument and its strict types declaration

# arrow functions

```
$messageKeys = array_map( fn( MessageSpecifier $m ) => $m->getKey(), $messages );
```

```
$database->method( 'newSelectQueryBuilder' )
```

```
->willReturnCallback( fn() => new SelectQueryBuilder( $database ) );
```

tests/phpunit/includes/api/ApiBaseTest.php,  
tests/phpunit/unit/includes/PingbackTest.php,  
as of commit a4da635e8a,  
GPL-2.0-or-later



# arrow functions

- short and sweet 😊
- automatically capture variables from the outer scope – notice that the second example didn't need use ( `$database` )!
  - but if you want to capture by reference, i.e. use ( `&$database` ), you'll still need an old-style function expression

# new operators

```
'cat_pages' => $countByType['page'] ?? 0,
```

includes/Category.php,

```
'cat_subcats' => $countByType['subcat'] ?? 0,
```

```
'cat_files' => $countByType['file'] ?? 0
```

```
// If length is null, calculate and remember it (potentially SLOW!).
```

```
// This is for compatibility with old database rows that don't have the field set.
```

```
$this->mSize ??= $this->mSlots->computeSize();
```

includes/Revision/RevisionStoreRecord.php,

```
return count( $a ) <=> count( $b );
```

includes/GlobalFunctions.php,  
as of commit 5c9674df53,  
GPL-2.0-or-later

# new operators

- `??` and `??=` effectively replace `isset()` (not `empty()`!) – like `isset()`, there is no error if anything is unset (e.g. missing array key, undefined variable)
- `$a ? : $b` evaluates to `$b` if `$a` is falsy in any way (zero, empty string, etc.), whereas `$a ?? $b` only evaluates to `$b` if `$a` is `null`
- `<=>` (“spaceship” :3) is useful for making comparator functions – it compares the two operands and evaluates to an integer less than, equal to or greater than `0`, respectively

# new functions

```
public function isKeyGlobal( $key ) {  
    return str_starts_with( $key, self::GLOBAL_PREFIX );  
}
```

includes/libs/objectcache/BagOStuff.php,

```
if ( str_ends_with( MW_CONFIG_FILE, '.php' ) ) {
```

includes/Setup.php,

```
public static function isExternal( $username ) {  
    return str_contains( $username, '>' );  
}
```

includes/user/ExternalUserNames.php,  
as of commit a4da635e8a,  
GPL-2.0-or-later

# new functions

- harder-to-read workarounds (`strpos()`, `substr()`, whatever) are no longer needed (see the [two RFCs](#) for other problems with such workarounds)
- technically, these functions are from PHP 8.0, but MediaWiki pulls in [symfony/polyfill-php80](#) – thanks to Symfony for this useful library ❤️

# 02

# What's coming up

PHP ≥8.0, coming soon™ to a production near you!

# Union types

```
public function getWikiId(): string|false;
```

includes/page/PageReference.php (edited)

```
public function getId( string|false $wikiId = self::LOCAL ): int;
```

includes/page/PageIdentity.php (edited),

```
public function createComment(
```

includes/CommentStore/CommentStoreBase.php  
(edited)

```
    IDatabase $dbw,
```

as of commit c40084e898,

```
    string|Message|CommentStoreComment $comment,
```

GPL-2.0-or-later

```
    array $data = null
```

```
) {
```

# Union types

- we've been using them in PHPDoc for a long time
- now we can finally make them proper types too, and get a runtime error if the value isn't one of the expected types
- mostly obsoletes `Assert::parameterType( 'A|B|C', $x, '$x' );`



# never return type

```
public function dieReadOnly(): never {  
  
    $this->fail( 'Unexpected call to selectField' );  
  
    throw new LogicException( 'Oops' ); // Can't happen, make analyzer happy
```

includes/api/ApiBase.php (edited),

tests/phpunit/includes/Revision/  
RevisionRendererTest.php,  
as of commit c40084e898,  
GPL-2.0-or-later

# never return type

- have a guarantee, at the language level, that the method will never ever return normally
  - if it doesn't exit or throw an exception, then PHP will throw an error when the end of the method body is reached
- lets us get rid of all sorts of `throw new LogicException( 'dieError did not throw an exception' );`

# Named arguments

```
self::$extensionJsonCache[$this->extensionJsonPath] = json_decode(  
    file_get_contents( $this->extensionJsonPath ),  
    true,  
    512,  
    JSON_THROW_ON_ERROR  
);
```

tests/phpunit/integration/includes/  
ExtensionJsonTestBase.php,  
as of commit c40084e898,  
GPL-2.0-or-later



# Named arguments

```
self::$extensionJsonCache[$this->extensionJsonPath] = json_decode(  
    file_get_contents( $this->extensionJsonPath ),  
    associative: true,  
    flags: JSON_THROW_ON_ERROR  
);
```

tests/phpunit/integration/includes/  
ExtensionJsonTestBase.php (edited),  
as of commit c40084e898,  
GPL-2.0-or-later



# Named arguments

- can make code much more readable
- can skip parameters instead of having to specify the default value
- caution: are parameter names part of the stable interface? is renaming a parameter a breaking change? to be decided...

? ->

```
return $this->user === null ? null : $this->user->getName();
```

includes/session/UserInfo.php,  
(edited),

```
return $this->user?->getName();
```

includes/actions/HistoryAction.php,

```
$revComment = $rev->getComment() === null ? null : $rev->getComment()->text;
```

(edited),

```
$revComment = $rev->getComment()?->text;
```

as of commit c40084e898,  
GPL-2.0-or-later

**?->**

- like -> but only if left-hand side is not null
- if used in a chain, the rest of the chain is skipped in case of null, i.e. you can chain normal ->s after the first ?-> and they won't crash

# Attributes

```
/**  
 * @return stdClass|array|false  
 */  
#[\ReturnTypeWillChange]  
public function current();
```

includes/libs/rdbms/database/  
resultwrapper/IResultWrapper.php,  
as of commit c40084e898,  
GPL-2.0-or-later



# Attributes

- syntax for adding additional information to classes, methods, etc.
- can be retrieved via reflection
- backwards-compatible: single-line attributes will simply be parsed as comments by PHP <8
- `ReturnTypeWillChange` lets us acknowledge the *upcoming* change to the return type of e.g. `Iterator::current()` in PHP 9, to avoid a deprecation warning in PHP 8, while staying compatible with PHP 7

# Constructor property promotion

```
class CreditsAction extends FormlessAction {  
    private LinkRenderer $linkRenderer;  
    private UserFactory $userFactory;  
  
    public function __construct(  
        Article $article,  
        IContextSource $context,  
        LinkRenderer $linkRenderer,  
        UserFactory $userFactory  
    ) {  
        parent::__construct( $article, $context );  
        $this->linkRenderer = $linkRenderer;  
        $this->userFactory = $userFactory;  
    }  
}
```

includes/actions/CreditsAction.php (edited),  
as of commit c40084e898,  
GPL-2.0-or-later



# Constructor property promotion

```
class CreditsAction extends FormlessAction {  
    public function __construct(  
        Article $article,  
        IContextSource $context,  
        private LinkRenderer $linkRenderer,  
        private UserFactory $userFactory  
    ) {  
        parent::__construct( $article, $context );  
    }  
}
```

includes/actions/CreditsAction.php (edited),  
as of commit c40084e898,  
GPL-2.0-or-later



# Constructor property promotion

- declare class properties as constructor parameters
- skip the `$this->whatever = $whatever` assignment in the constructor
- might be contentious, we'll see 😊

# match expressions

```
switch ( $ext ) {  
    case 'gif':  
        return 'image/gif';  
    case 'png':  
        return 'image/png';  
    case 'jpg':  
    case 'jpeg':  
        return 'image/jpeg';  
}  
  
return 'unknown/unknown';
```

includes/StreamFile.php,  
as of commit c40084e898,  
GPL-2.0-or-later



# match expressions

```
return match ( $ext ) {  
    'gif' => 'image/gif',  
    'png' => 'image/png',  
    'jpg', 'jpeg' => 'image/jpeg',  
    default => 'unknown/unknown',  
};
```

includes/StreamFile.php (edited),  
as of commit c40084e898,  
GPL-2.0-or-later



# match expressions

- concise syntax
- strict comparison (`===`) unlike `switch`'s loose comparison (`==`)

# enums

```
// Audience options for accessors
```

```
public const FOR_PUBLIC = 1;
```

```
public const FOR_THIS_USER = 2;
```

```
public const RAW = 3;
```

```
public function getUser( $audience = self::FOR_PUBLIC, Authority $performer = null ) {
```

```
// Audience options for accessors
```

```
public const AUDIENCE_PUBLIC = 1;
```

```
public const AUDIENCE_RAW = 2;
```

```
includes/Revision/RevisionRecord.php,
```

```
includes/user/CentralId/CentralIdLookup.php,  
as of commit 973253a7ee,  
GPL-2.0-or-later
```





# enums

```
enum RevisionAudience {  
    case ForPublic;  
    case ForThisUser;  
    case Raw;  
}
```

```
enum CentralIdAudience {  
    case Public;  
    case Raw;  
}
```



# enums

- “fancy objects” flavor of enums, with potential for future extension to “full Algebraic Data Types (ADTs)” flavor later
  - as opposed to “fancy constants” flavor
  - see their [survey of enums in various languages](#)
- “make invalid states unrepresentable” modeling technique
- see the [Enumerations RFC](#), it’s well written

# That's all!

Enjoy writing nicer PHP code and look forward to an even brighter future :)

