
PHP 8.6: The Inside Scoop

— Daniel Scherzer —

Daniel Scherzer

- Full-stack developer and open source contributor
- <https://scherzer.dev>
- <https://github.com/DanielEScherzer>
- PHP core contributor
 - ext/reflection maintainer
 - 8.5 rookie release manager
 - 8.6 veteran release manager

Overview

- Partial Function Application (PFA)
- ReflectionProperty: Beyond Visibility
- ReflectionParameter::getDocComment()
- ext/uri improvements (x4)
- Polling API
- Quick hits
- Deprecation of mbregex
- Questions?

Partial Function Application

- Placeholders with `?`
 - Or with `...` for multiple/variadic
 - But can provide some arguments (“partial”)
- Compile-time conversion to closures
- Nice integration with pipes
- Any callable*

PFA example (8.5 upgrade)

```
<?php
```

```
// ...
```

```
$url = new Url($requestURI, new Url('https://scherzer.dev'));  
$path = $url->getPath()  
    |> (static fn (string $path): string => trim($path, '/'))  
    |> strtolower(...)  
    |> ucfirst(...);
```

```
// ...
```

```
<?php
```

```
// ...
```

```
$url = new Url($requestURI, new Url('https://scherzer.dev'));  
$path = $url->getPath()  
    |> trim(' /')  
    |> strtolower(...)  
    |> ucfirst(...);
```

```
// ...
```

PFA example (older upgrade)

<?php

```
class BlogPostStore {
// ...
public function listBlogSlugs(): array {
    $entries = scandir( __DIR__ . '/posts/' );
    $files = array_filter(
        $entries,
        static fn ( string $path ) => str_ends_with( $path, '.md' )
    );
    $slugs = array_map(
        static fn ( string $title ) => substr( $title, 0, -3 ),
        $files
    );
    rsort( $slugs );
    return $slugs;
}
}
```

<?php

```
class BlogPostStore {
// ...
public function listBlogSlugs(): array {
    $slugs = scandir( __DIR__ . '/posts/' )
        |> array_filter( ?, str_ends_with( ?, '.md' ) )
        |> array_map( substr( ?, 0, -3 ), ? );

    rsort( $slugs );
    return $slugs;
}
}
```

ReflectionProperty: Beyond Visibility

- PHP 7.4: properties can be typed (must be initialized before reading)
- PHP 8.1: properties can be readonly (cannot write multiple times)
- PHP 8.4: asymmetric visibility, property hooks
- PHP 8.5: clone with (readonly properties can be re-initialized)

“Is it safe to (read/write) a specific class property from a given scope” is no longer just a question of visibility

ReflectionProperty::isReadable()

From the RFC

- If the property is defined and readable from the passed scope
 - and has no hooks or is backed, return true
 - and is virtual return true only if the get hook is defined
- If a `__get` hook is defined
 - and a `__isset` hook is defined, return the result of `__isset`
 - and there is no `__isset` hook, return true
- If an object is provided, also confirm:
 - The property is initialized
 - The property has not been unset(). If it has, follow the same `__isset`/`__get` check as above

ReflectionProperty::isReadable() examples

```
<?php
```

```
#[AllowDynamicProperties]
class Demo {
    public $unHooked;
    public $backed { set => $value; }
    public $virtualNoGet { set {} }
    public $virtualWithGet { set {} get => true; }
    public $magic;
    public $noMagic;
    public bool $unInit;

    public function __construct() {
        unset($this->magic, $this->noMagic);
    }
    public function __isset(string $name): bool {
        return $name === 'magic';
    }
    public function __get(string $name) {}
}
```

| Property | On object | | Generally |
|------------------|--------------|--|-----------|
| \$unHooked | Readable | | |
| \$backed | Readable | | |
| \$virtualNoGet | Not readable | | |
| \$virtualWithGet | Readable | | |
| \$magic | Readable | | |
| \$noMagic | Not readable | | Readable |
| \$unInit | Not readable | | Readable |
| \$dynamic | Readable | | |

ReflectionProperty::isWritable()

From the RFC

- If the property is defined and writable from the passed scope
 - and has no hooks or is backed, return true
 - and is virtual, return true only if the set hook is defined
- If a __set hook is defined, return true
- If an object is provided, also confirm at least one of:
 - the property is not readonly
 - the property is not yet initialized,
 - or is reinitializable (__clone)

ReflectionProperty::isWritable() examples

```
<?php
```

```
#[AllowDynamicProperties]
```

```
class Demo {
```

```
    public $unHooked;
```

```
    public $backed { set => $value; }
```

```
    public $virtualNoSet { get => true; }
```

```
    public $virtualWithSet { set {} get => true; }
```

```
    public public(set) readonly bool $unInit;
```

```
    public public(set) readonly bool $init;
```

```
    public function __construct() {
```

```
        $this->init = true;
```

```
    }
```

```
    public function __set(string $name, mixed $value) {
```

```
        $this->$name = $value;
```

```
    }
```

```
}
```

| Property | On object | Generally |
|------------------|--------------|-----------|
| \$unHooked | Writable | |
| \$backed | Writable | |
| \$virtualNoSet | Not writable | |
| \$virtualWithSet | Writable | |
| \$unInit | Writable | |
| \$init | Not writable | Writable |
| \$dynamic | Writable | |

ReflectionParameter::getDocComment()

Located anywhere up to the next comma; last comment controls

```
<?php
function demo(
    /** Comment1 */ $param1,
    /** Comment2-A */ $param2 /** Comment2-B */,
    $param3 /** Comment3 */,
    /** Comment4-A */
    /** Comment4-B */
    $param4
    /** Comment4-C */,
    $param5,
    /** Unused */
) {}

$ref = new ReflectionFunction('demo');
foreach ($ref->getParameters() as $param) {
    echo $param->name . ': ' . ($param->getDocComment() ? '(false)' . "\n";
}
```

ext/uri follow ups - builders

```
<?php
namespace Uri\Rfc3986;
final class UriBuilder {
    public function __construct() {}
    public function reset(): static {}
    public function setScheme(?string $scheme): static {}
    public function setUserInfo(
        #[SensitiveParameter] ?string $userInfo
    ): static {}
    public function setHost(?string $host): static {}
    public function setPath(string $path): static {}
    public function setQuery(?string $query): static {}
    public function setFragment(?string $fragment): static {}
    public function build(?Uri $baseUrl = null): Uri {}
}
```

```
<?php
namespace Uri\WhatWg;
final class UriBuilder {
    public function __construct() {}
    public function reset(): static {}
    public function setScheme(?string $scheme): static {}
    public function setUsername(
        ?string $username): static {}
    public function setPassword(
        #[SensitiveParameter] ?string $password
    ): static {}
    public function setHost(?string $host): static {}
    public function setPath(string $path): static {}
    public function setQuery(?string $query): static {}
    public function setFragment(?string $fragment): static {}
    public function build(
        ?Uri $baseUrl = null, &$errors = null): Uri {}
}
```

ext/uri follow ups - URI types

```
<?php
namespace Uri\Rfc3986;
enum UriType {
    case AbsolutePathReference;
    case RelativePathReference;
    case NetworkPathReference;
    case Uri;
}

final readonly class Uri {

    // ...

    public function getUriType(): UriType {}
}
```

```
<?php
namespace Uri\WhatWg;
final readonly class Uri {

    // ...

    public function isSpecialScheme(): bool {}
}
```

ext/uri follow ups - host types

```
<?php
namespace Uri\Rfc3986;
enum UriHostType {
    case IPv4;
    case IPv6;
    case IpVFuture;
    case RegisteredName;
}

final readonly class Uri {

    // ...

    public function getHostType(): ?UriHostType {}
}
```

```
<?php
namespace Uri\WhatWg;
enum UriHostType {
    case IPv4;
    case IPv6;
    case Domain;
    case Opaque;
    case Empty;
}

final readonly class Uri {

    // ...

    public function getHostType(): ?UriHostType {}
}
```

ext/uri follow ups - percent encoding

```
<?php
namespace Uri\Rfc3986;
enum UriPercentEncodingMode {
    case UserInfo;
    case RegisteredNameHost;
    case Path;
    case PathSegment;
    case Query;
    case FormQuery;
    case Fragment;
    case AllReservedCharacters;
    case AllButUnreservedCharacters;
}
function uri_percent_encode(
    string $input,
    UriPercentEncodingMode $mode
): string {}
```

```
<?php
namespace Uri\WhatWg;
enum UriPercentEncodingMode {
    case Username;
    case Password;
    case OpaqueHost;
    case Path;
    case OpaquePath;
    case PathSegment;
    case Query;
    case SpecialQuery;
    case FormQuery;
    case Fragment;
}
function url_percent_encode(
    string $input,
    UriPercentEncodingMode $mode
): string {}
```

Polling API

- Everything in `\Io\Poll`
 - Context: monitor handles, wait for events
 - Watcher: result of `\Io\Poll\Context::add()`, used to check events
 - StreamPollHandle: handle for monitoring streams
 - Event: enum with events to listen to
- Laying the groundwork for big future plans
 - [RFC: Polling API Additions](#) - weak handles, ext/socket integration, timers, and more
 - [RFC: IO Hooks and Operations](#) - integration point for concurrency

Polling API - example

```
<?php

$descriptors = [
    ['pipe', 'r'],
    ['pipe', 'w'],
    ['file', '/dev/null', 'a'],
];
$pipes = [];
$process = proc_open($cmd, $descriptors, $pipes);
$poll = new \Io\Poll\Context;
$handle = new StreamPollHandle($pipes[1]);
$poll->add($handle, [\Io\Poll\Event::Read]);
```

// Continued on the right

```
<?php

// Given the prior setup on the left

while (true) {
    $watchers = $poll->wait();
    foreach ($watchers as $watcher) {
        foreach ($watcher->getTriggeredEvents() as $event) {
            switch ($event) {
                case \Io\Poll\Event::Read:
                    $content = trim(fread($pipes[1], 8192));
                    // Do some stuff
                    break;
                case \Io\Poll\Event::HangUp:
                    break 4;
            }
        }
    }
}
```

Taken from my code at <https://github.com/dseguy/phpenomenal/pull/6>

SortDirection

```
<?php
```

```
enum SortDirection {  
    case Ascending;  
    case Descending;  
}
```

Clamp

```
<?php
```

```
function clamp(mixed $value, mixed $min, mixed $max) {  
    if (is_float($min) && is_nan($min)) {  
        throw new ValueError('Argument 2 ($min) must not be NaN');  
    } else if (is_float($max) && is_nan($max)) {  
        throw new ValueError('Argument 3 ($max) must not be NaN');  
    } else if ($min > $max) {  
        throw new ValueError(  
            'Argument 2 ($max) must be smaller than '  
            'or equal to argument #3 ($max)'  
        );  
    }  
    if ($value < $min) {  
        return $min;  
    } else if ($value > $max) {  
        return $max;  
    } else {  
        return $value;  
    }  
}
```

Readonly defaults

- Satisfies `get`-only property hooks
- Basically a constant
- Default is initialization

?php

```
interface Conference {  
    public string $name { get; }  
}
```

```
class LonghornPHP2026 implements Conference {  
    public readonly string $name = 'Longhorn PHP 2026';  
}
```

```
var_dump(new LonghornPHP2026()->name);
```

Constant writes

- Constant holds an object
- Change the object's properties

<?php

```
class ConferenceTracker {  
    public string $mostRecent = 'FOSSY 2026';  
}
```

```
const MY_CONFERENCES = new ConferenceTracker();
```

```
MY_CONFERENCES->mostRecent = 'Longhorn PHP 2026';
```

```
var_dump(MY_CONFERENCES);
```

Debuggable enums

- Because just a case might not be clear

```
<?php
enum Command {
    /** Print usage instructions */
    case Help;

    public function __debugInfo(): array {
        $r = new ReflectionEnumUnitCase(
            __CLASS__,
            $this->name
        );
        return [ 'docs' => $r->getDocComment() ];
    }
}
var_dump(Command::Help);
```

Overridden constants

- Overriding constants: PHP 5.0 (at least)
- `#[\Override]` on methods: 8.3
- `#[\Override]` on properties: 8.5
- `#[\Override]` on constants: 8.6

```
<?php

interface Command {
    public const string NAME = 'ERROR: NO OVERRIDE';
}

class DemoCommand implements Command {
    #[\Override]
    public const string NAME = 'DEMO';
}
```

Other big changes

- Updated session defaults ([RFC](#))
 - `session.use_strict_mode` (0 -> 1)
 - `session.cookie_httponly` (0 -> 1)
 - `session.cookie_samesite` (not set -> "Lax")
- `grapheme_strrev()` to reverse graphemes ([RFC](#))
- `mysqli_quote_string()` and `mysqli::quote_string()` for escaping ([RFC](#))
- `trim()` (and `ltrim()` / `rtrim()`) now remove `\f` (form feed) by default ([RFC](#))
- New `\Time\Duration` class ([RFC](#))

Deprecated: mbregex support (--enable-mbstring)

- Oniguruma library is no longer maintained
 - `mb_ereg()`, `mb_ereg_*`()
 - `mb_eregi()`, `mb_eregi_replace()`
 - `mb_regex_encoding()`, `mb_regex_set_options()`
 - `mb_split()`
 - `MB_ONIGURUMA_VERSION`
 - `mbstring.regex_retry_limit`
 - `mbstring.regex_stack_limit`
- ``mb_onig/mb_onig`` extension available via PIE

Any requests for PHP 8.7?

- FILTER_THROW_ON_FAILURE requested on GitHub
 - I added this in 8.5 after [an RFC](#)
 - Previously hadn't worked on ext/filter ([my commits](#))
- Connecting attributes with their targets
 - Long-requested on GitHub and internals
 - Requested by Larry at Longhorn PHP 2025
 - Implementation created, RFC discussion will resume shortly
- Adding friendship
 - Discussed with Dave at Longhorn PHP and ConFoo
 - Implementation created, RFC discussion will resume shortly

Thank You

Questions?

Feedback on Joind.in

