

# PHP 8.1 What's new and changed



DrupalCon  
EUROPE2021  
OCTOBER 4-7

#DrupalConEur



Ayesh Karunaratne | <https://aye.sh/talk/php81-drupalconeur-2021>

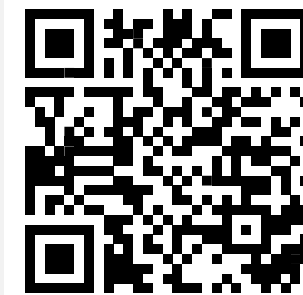
# PHP 8.1

## What's new and changed



**DrupalCon**  
EUROPE2021  
OCTOBER 4-7

#DrupalConEur



Ayesh Karunaratne | <https://aye.sh/talk/php81-drupalconeur-2021>



## Ayesh Karunaratne

---

Freelance Software Developer, Security Researcher, Full-time traveler

 Kandy, Sri Lanka – Everywhere

 <https://aye.sh> | <https://php.watch>

 @Ayeshlive | @phpwch

 [ayesh@aye.sh](mailto:ayesh@aye.sh)

# **PHP 8.1**

## **New and changed**

# PHP 8.1

## New and changed

2020 Nov 26

**PHP 8.0 General Availability**

14 Jun 2021

**PHP 8.1 - First alpha**

20 Jul 2021

**Feature-freeze / First beta**

02 Sep 2021

**First Release Candidate**

2021 Oct 04

**DrupalCon Europe**



2021 Nov 25

**PHP 8.1**

# PHP 8.1: New and Changed



## New Features



Enums



Fibers



Readonly Properties



Type System Improvements



Misc.

Changed Functionality 

Deprecations 

Backwards Compatibility 

Testing out today 

# PHP 8.1: New and Changed

Enums

Fibers

Readonly Properties

Type System Improvements

Misc.

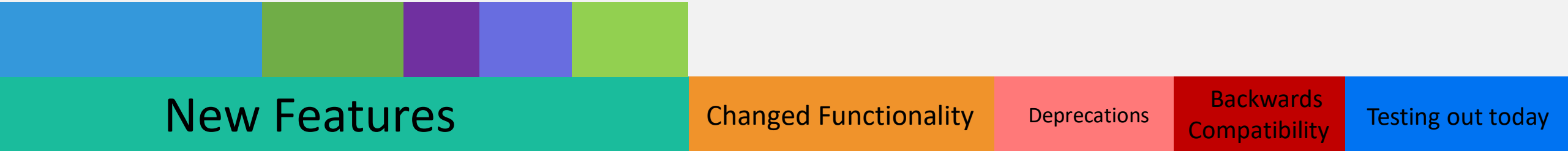
New Features

Changed Functionality

Deprecations

Backwards  
Compatibility

Testing out today





# PHP 8.1 New Features



# Enums



```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}
```

core/modules/comment/src/Plugin/Field/FieldType/CommentItemInterface.php



```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}
```

core/modules/comment/src/Plugin/Field/FieldType/CommentItemInterface.php

```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}
```

```
$node->setCommentStatus(CommentItemInterface::HIDDEN);
$node->save();
```



```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}
```

```
$node →setCommentStatus(CommentItemInterface :: HIDDEN);
$node →save();
```



```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}
```

```
$node->setCommentStatus(CommentItemInterface::HIDDEN);
$node->save();
```



```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}
```

```
$node →setCommentStatus(0);
$node →save();
```





```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}

public function getValueOptions() {
    $this->valueOptions = [
        CommentItemInterface::HIDDEN => $this->t('Hidden'),
        CommentItemInterface::CLOSED => $this->t('Closed'),
        CommentItemInterface::OPEN => $this->t('Open'),
    ];
    return $this->valueOptions;
}
```

core/modules/comment/src/Plugin/views/filter/NodeComment.php

```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}

public function getValueOptions() {
    $this->valueOptions = [
        CommentItemInterface::HIDDEN => $this->t('Hidden'),
        CommentItemInterface::CLOSED => $this->t('Closed'),
        CommentItemInterface::OPEN => $this->t('Open'),
    ];
    return $this->valueOptions;
}
```

core/modules/comment/src/Plugin/views/filter/NodeComment.php

```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}
```

```
public function setCommentStatus(int $state) {
    $node →set('comment', $state);
}
```



```
/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}
```

```
public function setCommentStatus(int $state) {
    $node->set('comment', $state);
}
```



```

/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}

public function setCommentStatus(int $state) {
    if ( $state !== CommentItemInterface::HIDDEN)
        && $state !== CommentItemInterface::CLOSED
        && $state !== CommentItemInterface::OPEN){
        throw new InvalidArgumentException('Invalid comment state');
    }

    $node →set('comment', $state);
}

```

```

/**
 * Interface definition for Comment items.
 */
interface CommentItemInterface {
    const HIDDEN = 0;
    const CLOSED = 1;
    const OPEN = 2;
}

public function setCommentStatus(int $state) {
    if ( $state !== CommentItemInterface::HIDDEN)
        && $state !== CommentItemInterface::CLOSED
        && $state !== CommentItemInterface::OPEN){
        throw new InvalidArgumentException('Invalid comment state');
    }

    $node->set('comment', $state);
}

```

```
enum CommentStatus {  
  
}
```



```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```





```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
/**  
 * Interface definition for Comment items.  
 */  
interface CommentItemInterface {  
    const HIDDEN = 0;  
    const CLOSED = 1;  
    const OPEN = 2;  
}  
  
public function setCommentStatus(int $state) {  
    if ( $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::CLOSED  
        && $state !== CommentItemInterface::OPEN){  
        throw new InvalidArgumentException('Invalid comment state'  
    )  
    }  
  
    $node->set('comment', $state);  
}
```

```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
/**  
 * Interface definition for Comment items.  
 */  
interface CommentItemInterface {  
    const HIDDEN = 0;  
    const CLOSED = 1;  
    const OPEN = 2;  
}  
  
public function setCommentStatus(int $state) {  
    if ( $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::HIDDEN){  
        throw new InvalidArgumentException('Invalid comment state'  
    }  
  
    $node->set('comment', $state);  
}
```

```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
/**  
 * Interface definition for Comment items.  
 */  
interface CommentItemInterface {  
    const HIDDEN = 0;  
    const CLOSED = 1;  
    const OPEN = 2;  
}  
  
public function setCommentStatus(int $state) {  
    if ( $state ≠ CommentItemInterface::HIDDEN  
        && $state ≠ CommentItemInterface::HIDDEN  
        && $state ≠ CommentItemInterface::HIDDEN){  
        throw new InvalidArgumentException('Invalid comment state'  
    }  
  
    $node →set('comment', $state);  
}
```

```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
/**  
 * Interface definition for Comment items.  
 */  
interface CommentItemInterface {  
    const HIDDEN = 0;  
    const CLOSED = 1;  
    const OPEN = 2;  
}  
  
public function setCommentStatus(int $state) {  
    if ( $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::HIDDEN){  
        throw new InvalidArgumentException('Invalid comment state'  
    )  
    }  
  
    $node->set('comment', $state);  
}
```

```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
/**  
 * Interface definition for Comment items.  
 */  
interface CommentItemInterface {  
    const HIDDEN = 0;  
    const CLOSED = 1;  
    const OPEN = 2;  
}  
  
public function setCommentStatus(CommentStatus $state) {  
    if ( $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::HIDDEN){  
        throw new InvalidArgumentException('Invalid comment state'  
    )  
    }  
  
    $node->set('comment', $state);  
}
```

```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
/**  
 * Interface definition for Comment items.  
 */  
interface CommentItemInterface {  
    const HIDDEN = 0;  
    const CLOSED = 1;  
    const OPEN = 2;  
}  
  
public function setCommentStatus(CommentStatus $state) {  
    if ( $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::HIDDEN  
        && $state !== CommentItemInterface::HIDDEN){  
        throw new InvalidArgumentException('Invalid comment state'  
    }  
  
    $node →set('comment', $state);  
}
```

```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
/**  
 * Interface definition for Comment items.  
 */  
interface CommentItemInterface {  
    const HIDDEN = 0;  
    const CLOSED = 1;  
    const OPEN = 2;  
}  
  
public function setCommentStatus(CommentStatus $state) {  
    if ( $state !== CommentItemInterface::HIDDEN  
    && $state !== CommentItemInterface::HIDDEN  
    && $state !== CommentItemInterface::HIDDEN){  
    throw new InvalidArgumentException('Invalid comment state'  
    }  
  
    $node →set('comment', $state);  
}
```

```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
/**  
 * Interface definition for Comment items.  
 */  
interface CommentItemInterface {  
}
```

```
public function setCommentStatus(CommentStatus $state) {  
    $node →set('comment', $state);  
}
```





```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
public function setCommentStatus(CommentStatus $state) {  
    $node->set('comment', $state);  
}
```

```
$node->setCommentStatus(CommentStatus::OPEN);
```



```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```

```
public function setCommentStatus(CommentStatus $state) {  
    $node->set('comment', $state);  
}
```

```
$node->setCommentStatus(CommentStatus::OPEN);
```



```
enum CommentStatus {  
  case HIDDEN;  
  case CLOSED;  
  case OPEN;  
}
```

```
public function setCommentStatus(CommentStatus $state) {  
  $node →set('comment', $state);  
}
```

```
$node →setCommentStatus(Node::Published);  
$node →setCommentStatus('potato');  
$node →setCommentStatus(null);  
$node →setCommentStatus(3.14);
```



```
enum CommentStatus {  
  case HIDDEN;  
  case CLOSED;  
  case OPEN;  
}
```

```
public function setCommentStatus(CommentStatus $state) {  
    $node->set('comment', $state);  
}
```

```
$node->setCommentStatus(Node::Published);  
$node->setCommentStatus('potato');  
$node->setCommentStatus(null);  
$node->setCommentStatus(3.14);
```

**TypeError:** setCommentStatus(): Argument #1 (\$state) must be of type CommentStatus, string given ...



```
function setIsSponsored(bool $sponsored): void {  
  
}  
  
function isSponsored(): bool {  
  
}  
  
setIsSponsored(true);  
setIsSponsored(false);
```



- Enums can have zero or more members

```
enum Suit {  
}
```



- Enums can have zero or more members

```
enum Suit {  
    case Clubs;  
    case Diamonds;  
    case Spades;  
    case Hearts;  
}
```



- Enums can have zero or more members
- **Enum members are objects**

```
enum Suit {  
    case Clubs;  
    case Diamonds;  
    case Spades;  
    case Hearts;  
}
```

```
is_object(Suit::Hearts);  
// true
```





- Enums can have zero or more members
- **Enum members are objects**

```
enum Suit {  
    case Clubs;  
    case Diamonds;  
    case Spades;  
    case Hearts;  
}
```

```
var_dump(Suit::Hearts);  
// enum(Suit::Hearts)
```



- Enums can have zero or more members
- Enum members are objects
- **Enums can be namespaced and autoloaded**

```
namespace App\PlayingCards;
```

```
enum Suit {  
    case Clubs;  
    case Diamonds;  
    case Spades;  
    case Hearts;  
}
```



```
namespace App\PlayingCards;
```

```
enum Suit: int {  
    case Clubs = 1;  
    case Diamonds = 2;  
    case Spades = 3;  
    case Hearts = 4;  
}
```

- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- **May contain string|int backed values**



- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- **May contain `string|int` backed values**

```
namespace App\PlayingCards;
```

```
enum Suit: string {  
    case Clubs = '♣';  
    case Diamonds = '♦';  
    case Spades = '♠';  
    case Hearts = '♥';  
}
```



```
namespace App\PlayingCards;
```

```
enum Suit: string {
```

```
    const AWESOME = 'Yes';
```

```
    case Clubs = '♣';
```

```
    case Diamonds = '♦';
```

```
    case Spades = '♠';
```

```
    case Hearts = '♥';
```

```
}
```

- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- May contain string|int backed values
- **May contain non-duplicated constants**



```
namespace App\PlayingCards;
```

```
enum Suit: string {
```

```
    const AWESOME = 'Yes';
```

```
    case Clubs = '♣';
```

```
    case Diamonds = '♦';
```

```
    case Spades = '♠';
```

```
    case Hearts = '♥';
```

```
    public static function cheer(): void {
```

```
        echo 'Yay!';
```

```
    }
```

```
}
```

- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- May contain string|int backed values
- May contain non-duplicated constants
- **May contain static methods**

```
Suit::cheer();
```

```
// Yay!
```



```
namespace App\PlayingCards;
```

```
enum Suit: string {  
    const AWESOME = 'Yes';
```

```
    case Clubs = '♣';  
    case Diamonds = '♦';  
    case Spades = '♠';  
    case Hearts = '♥';
```

```
    public static function cheer(): void {  
        echo 'Yay!';  
    }
```

```
    public function show(): void {  
        var_dump($this);  
        var_dump($this → name);  
        var_dump(self::Clubs → name);  
        var_dump($this → value);  
        var_dump(self::Clubs → value);  
    }
```

```
}
```

- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- May contain string|int backed values
- May contain non-duplicated constants
- May contain static methods
- **May contain non-static methods**

```
Suit::Clubs → show();
```



```
namespace App\PlayingCards;
```

```
enum Suit: string {  
    const AWESOME = 'Yes';
```

```
    case Clubs = '♣';  
    case Diamonds = '♦';  
    case Spades = '♠';  
    case Hearts = '♥';
```

```
public static function cheer(): void {  
    echo 'Yay!';  
}
```

```
public function show(): void {  
    var_dump($this);  
    var_dump($this → name);  
    var_dump(self::Clubs → name);  
    var_dump($this → value);  
    var_dump(self::Clubs → value);  
}
```

```
}
```

- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- May contain string|int backed values
- May contain non-duplicated constants
- May contain static methods
- May contain non-static methods
- `$this` refers to the Enumerated element

```
Suit::Clubs → show();
```

```
enum(App\PlayingCards\Suit::Clubs)
```





```
namespace App\PlayingCards;
```

```
enum Suit: string {  
    const AWESOME = 'Yes';
```

```
    case Clubs = '♣';  
    case Diamonds = '♦';  
    case Spades = '♠';  
    case Hearts = '♥';
```

```
    public static function cheer(): void {  
        echo 'Yay!';  
    }
```

```
    public function show(): void {  
        var_dump($this);  
        var_dump($this->name);  
        var_dump(self::Clubs->name);  
        var_dump($this->value);  
        var_dump(self::Clubs->value);  
    }
```

```
}
```

- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- May contain string|int backed values
- May contain non-duplicated constants
- May contain static methods
- May contain non-static methods
- `$this` refers to the Enumerated element
- `->name` property is the name of the member

```
Suit::Clubs->show();
```

```
enum(App\PlayingCards\Suit::Clubs)  
string(5) "Clubs"  
string(5) "Clubs"
```



```
namespace App\PlayingCards;
```

```
enum Suit: string {  
    const AWESOME = 'Yes';
```

```
    case Clubs = '♣';  
    case Diamonds = '♦';  
    case Spades = '♠';  
    case Hearts = '♥';
```

```
    public static function cheer(): void {  
        echo 'Yay!';  
    }
```

```
    public function show(): void {  
        var_dump($this);  
        var_dump($this->name);  
        var_dump(self::Clubs->name);  
        var_dump($this->value);  
        var_dump(self::Clubs->value);  
    }  
}
```

- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- May contain string|int backed values
- May contain non-duplicated constants
- May contain static methods
- May contain non-static methods
- `$this` refers to the Enumerated element
- `->name` property is the name of the member
- `->value` property is the backed value

```
Suit::Clubs->show();
```

```
enum(App\PlayingCards\Suit::Clubs)  
string(5) "Clubs"  
string(5) "Clubs"  
string(6) "♣"  
string(6) "♣"
```

```
namespace App\PlayingCards;
```

```
enum Suit: string {  
    const AWESOME = 'Yes';
```

```
    case Clubs = '♣';  
    case Diamonds = '♦';  
    case Spades = '♠';  
    case Hearts = '♥';
```

```
    public static function cheer(): void {  
        echo 'Yay!';  
    }
```

```
    public function show(): void {  
        var_dump($this);  
        var_dump($this → name);  
        var_dump(self::Clubs → name);  
        var_dump($this → value);  
        var_dump(self::Clubs → value);  
    }
```

```
}
```

- Enums can have zero or more members
- Enum members are objects
- Enums can be namespaced and autoloaded
- May contain string|int backed values
- May contain non-duplicated constants
- May contain static methods
- May contain non-static methods
- \$this refers to the Enumerated element
- →name property is the name of the member
- →value property is the backed value

```
Suit::Clubs → show();
```

```
enum(App\PlayingCards\Suit::Clubs)  
string(5) "Clubs"  
string(5) "Clubs"  
string(6) "♣"  
string(6) "♣"
```



# Class Semantics

- Supports namespaces
- Supports traits
- Supports autoloading
- Supports magic constants
- Supports instanceof
- **Supports methods**

```
namespace Foo\Bar;

enum PostStatus: string implements EntityStatees {

    use TestTrait;

    case DRAFT = 'draft';
    case PENDING = 'pending';
    case RETURNED = 'returned';
    case PUBLISHED = 'published';

    public static function showOff(): void {
        echo __CLASS__ . static::class;
    }
}
```

# Enum Semantics

- Must not contain state

```
enum Suit {  
    case Spades;  
    case Hearts;  
    case Clubs;  
    case Diamonds;  
  
    private string $foo;  
}
```

Fatal error: Enums may not include properties

# Enum Semantics

- Must not contain state
- **Enum members are identical to same member**

CommentStatus :: OPEN  $\equiv$  CommentStatus :: OPEN



```
enum CommentStatus {  
    case HIDDEN;  
    case CLOSED;  
    case OPEN;  
}
```



```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```





```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$connection->insert('mytable')  
->fields([  
    'entity_id' => $entity->getId(),  
    'comment' => $commentState->value,  
])  
->execute();
```

```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$connection->insert('mytable')  
->fields([  
    'entity_id' => $entity->getId(),  
    'comment' => $commentState->value,  
])  
->execute();
```



```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$database->query("  
    SELECT entity_id  
    FROM {my_table}  
    WHERE status = :status",  
    [':status' => CommentStatus::OPEN->value]  
);
```



```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$database->query("  
    SELECT entity_id  
    FROM {my_table}  
    WHERE status = :status",  
    [':status' => CommentStatus::OPEN->value]  
);
```



```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$database->query("  
    SELECT entity_id  
    FROM {my_table}  
    WHERE status = :status",  
    [':status' => CommentStatus::OPEN->value]  
);
```



```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$database->query("  
    SELECT entity_id  
    FROM {my_table}  
    WHERE status = :status",  
    [':status' => CommentStatus::OPEN->value]  
);
```

```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$state = $formState->getValue('comment_state');  
$state = CommentStatus::from($state);  
$node->setCommentState($state);
```



```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$state = $formState->getValue('comment_state');  
$state = CommentStatus::from($state);  
$node->setCommentState($state);
```





```
enum CommentStatus: int {  
    case HIDDEN = 0;  
    case CLOSED = 1;  
    case OPEN = 2;  
}
```

```
$state = $formState->getValue('comment_state');  
$state = CommentStatus::from($state);  
$node->setCommentState($state);
```

```
enum(CommentStatus::Open)
```



# Fibers



# Fibers

Lightweight and controlled concurrency to PHP

A Fiber is a code block that **maintains its own stack** (variables and state), that can be **started, suspended, or terminated cooperatively** by the main code and the Fiber.



# Normal Execution Flow



**New Features: Fibers**



# Concurrent Execution Flow



**New Features: Fibers**

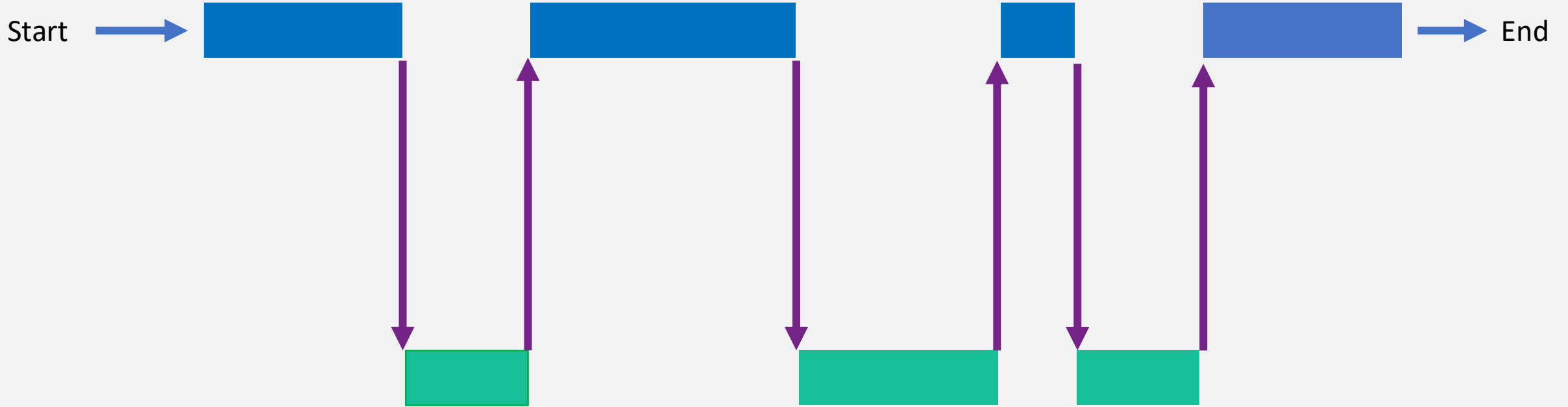


# Concurrent Execution Flow



**New Features: Fibers**

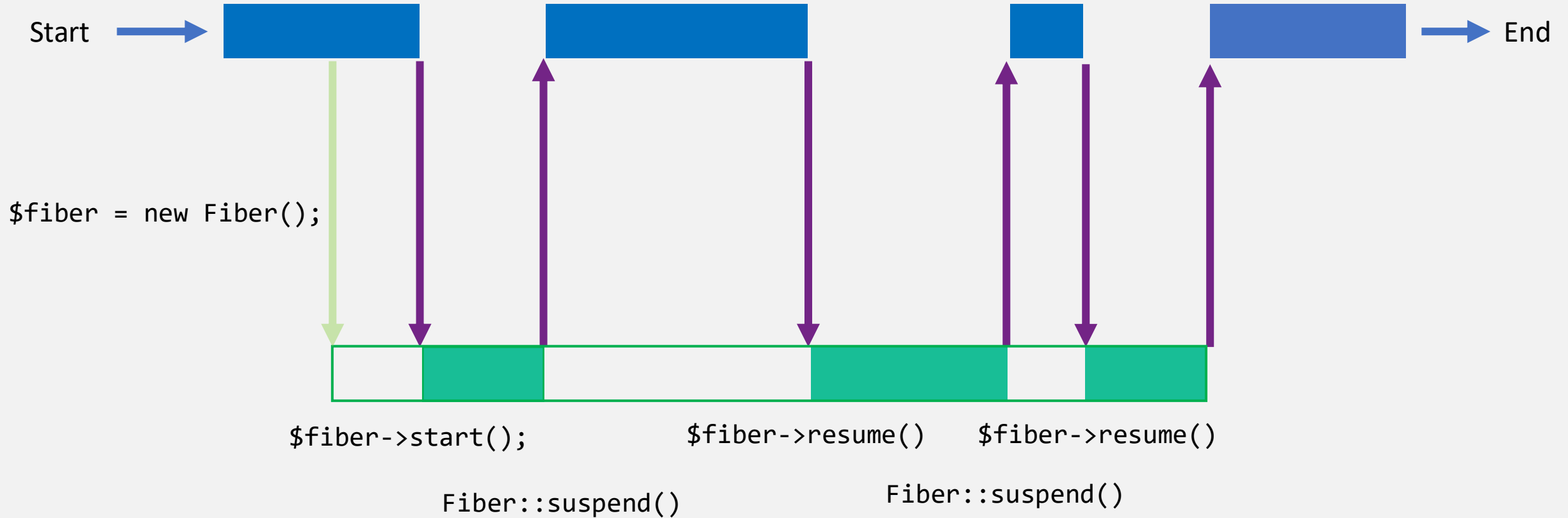
# Concurrent Execution Flow



New Features: Fibers



# Concurrent Execution Flow





```
echo "Main started";

$fiber = new Fiber(function(): void {
    echo "Fiber started";
    Fiber::suspend();
    echo "Fiber resumed";
});

$fiber->start();
echo "Fiber returned";
$fiber->resume();
echo "Fiber Ended";
```

```
echo "Main started";

$fiber = new Fiber(function(): void {
    echo "Fiber started";
    Fiber::suspend();
    echo "Fiber resumed";
});

$fiber->start();
echo "Fiber returned";
$fiber->resume();
echo "Fiber Ended";
```

```
echo "Main started";

$fiber = new Fiber(function(): void {
    echo "Fiber started";
    Fiber::suspend();
    echo "Fiber resumed";
});

$fiber->start();
echo "Fiber returned";
$fiber->resume();
echo "Fiber Ended";
```

```
echo "Main started";

$fiber = new Fiber(function(): void {
    echo "Fiber started";
    Fiber::suspend();
    echo "Fiber resumed";
});

$fiber->start();
echo "Fiber returned";
$fiber->resume();
echo "Fiber Ended";
```

```
echo "Main started";

$fiber = new Fiber(function(): void {
    echo "Fiber started";
    Fiber::suspend();
    echo "Fiber resumed";
});

$fiber->start();
echo "Fiber returned";
$fiber->resume();
echo "Fiber Ended";
```

```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {  
    echo "Fiber started";  
    Fiber::suspend();  
    echo "Fiber resumed";  
});
```

```
$fiber->start();  
echo "Fiber returned";  
$fiber->resume();  
echo "Fiber Ended";
```

Main started



```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {  
    echo "Fiber started";  
    Fiber::suspend();  
    echo "Fiber resumed";  
});
```

```
$fiber->start();  
echo "Fiber returned";  
$fiber->resume();  
echo "Fiber Ended";
```

Main started



```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {  
    echo "Fiber started";  
    Fiber::suspend();  
    echo "Fiber resumed";  
});
```

```
$fiber->start();
```

```
echo "Fiber returned";
```

```
$fiber->resume();
```

```
echo "Fiber Ended";
```

Main started





```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {
```

```
    echo "Fiber started";
```

```
    Fiber::suspend();
```

```
    echo "Fiber resumed";
```

```
});
```

```
$fiber->start();
```

```
echo "Fiber returned";
```

```
$fiber->resume();
```

```
echo "Fiber Ended";
```

```
Main started  
Fiber started
```



```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {  
    echo "Fiber started";  
    Fiber::suspend();  
    echo "Fiber resumed";  
});
```

```
$fiber->start();  
echo "Fiber returned";  
$fiber->resume();  
echo "Fiber Ended";
```

```
Main started  
Fiber started
```



```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {  
    echo "Fiber started";  
    Fiber::suspend();  
    echo "Fiber resumed";  
});
```

```
$fiber->start();
```

```
echo "Fiber returned";
```

```
$fiber->resume();
```

```
echo "Fiber Ended";
```

```
Main started  
Fiber started  
Fiber returned
```



```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {  
    echo "Fiber started";  
    Fiber::suspend();  
    echo "Fiber resumed";  
});
```

```
$fiber->start();  
echo "Fiber returned";  
$fiber->resume();  
echo "Fiber Ended";
```

```
Main started  
Fiber started  
Fiber returned
```



```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {  
    echo "Fiber started";  
    Fiber::suspend();  
    echo "Fiber resumed";  
});
```

```
$fiber->start();  
echo "Fiber returned";  
$fiber->resume();  
echo "Fiber Ended";
```

```
Main started  
Fiber started  
Fiber returned  
Fiber resumed
```



```
echo "Main started";
```

```
$fiber = new Fiber(function(): void {  
    echo "Fiber started";  
    Fiber::suspend();  
    echo "Fiber resumed";  
});
```

```
$fiber->start();  
echo "Fiber returned";  
$fiber->resume();
```

```
echo "Fiber Ended";
```

```
Main started  
Fiber started  
Fiber returned  
Fiber resumed  
Fiber Ended
```



```
final class Fiber {
    public function __construct(callable $callback) {}
    public function start(mixed ...$args): mixed {}
    public static function suspend(mixed $value = null): mixed {}
    public function resume(mixed $value = null): mixed {}
    public static function getCurrent(): ?self {}
    public function getReturn(): mixed {}
    public function throw(Throwable $exception): mixed {}
    public function isStarted(): bool {}
    public function isSuspended(): bool {}
    public function isRunning(): bool {}
    public function isTerminated(): bool {}
}
```

<https://php.watch/versions/8.1/fibers>

# Readonly Properties





# Readonly Properties

Properties with the readonly flag can only be initialized from object scope, and cannot be overwritten.



```
class User {  
    public int $uid;  
  
    public function __construct(int $uid) {  
        $this->uid = $uid;  
    }  
}
```



```
class User {  
    public readonly int $uid;  
  
    public function __construct(int $uid) {  
        $this->uid = $uid;  
    }  
}
```



```
class User {  
    public readonly int $uid;  
  
    public function __construct(int $uid) {  
        $this->uid = $uid;  
        $this->uid = $uid;  
    }  
}
```

Fatal error: Uncaught Error: Cannot modify readonly property User::\$uid in ...:...

```
class User {  
    public readonly int $uid;  
}  
$user = new User();  
$user →uid = 9;
```

Error: Cannot initialize readonly property  
User::\$uid from global scope in ... : ...

# Type System Improvements



# Intersection Types

```
function count_and_iterate(Iterator&Countable $value) {  
    foreach($value as $val) {}  
    count($value);  
}
```

# Intersection Types

```
function count_and_iterate(Iterator & Countable $value) {  
    foreach($value as $val) {}  
    count($value);  
}
```



# Intersection Types

```
function count_and_iterate(Iterator&Countable $value) {  
    foreach($value as $val) {}  
    count($value);  
}
```

# Intersection Types

```
function count_and_iterate(Iterator&Countable $value) {  
    foreach($value as $val) {}  
    count($value);  
}
```

```
class CountableIterator implements Iterator, Countable {  
    public function current(): mixed {}  
    public function key(): mixed {}  
    public function next(): void {}  
    public function rewind(): void {}  
    public function valid(): bool {}  
  
    public function count(): int {}  
}
```

# Intersection Types

```
function count_and_iterate(Iterator&Countable $value) {  
    foreach($value as $val) {}  
    count($value);  
}
```

```
class CountableIterator implements Iterator, Countable {  
    public function current(): mixed {}  
    public function key(): mixed {}  
    public function next(): void {}  
    public function rewind(): void {}  
    public function valid(): bool {}  
  
    public function count(): int {}  
}
```

# Intersection Types

```
function count_and_iterate(Iterator&Countable $value) {  
    foreach($value as $val) {}  
    count($value);  
}
```

```
class CountableIterator implements Iterator, Countable {  
    public function current(): mixed {}  
    public function key(): mixed {}  
    public function next(): void {}  
    public function rewind(): void {}  
    public function valid(): bool {}  
  
    public function count(): int {}  
}
```

# never type

```
function redirectUser(): void {  
    header('Location: https://example.com');  
    die();  
}
```



# never type

```
function redirectUser(): never {  
    header('Location: https://example.com');  
    die();  
}
```

# never type

```
function redirectUser(): never {  
    header('Location: https://example.com');  
    return;  
}
```

Fatal error: A never-returning function must not return in ... on line ...

# Other New Features





- Final class constants
- New `fsync` and `fdatasync` functions
- New `array_is_list` function
- Sodium functions
- First-class callable syntax
- Explicit Octal numeral notation
- xxHash and MurmurHash3 hashing algorithms
- Directory-upload support with `$_FILES['full_path']`

# Changed Functionality



# Tentative Return Types

```
interface ArrayAccess {  
    /** @return bool */  
    public function offsetExists($offset);  
  
    /** @return mixed */  
    public function offsetGet($offset);  
  
    /** @return void */  
    public function offsetSet($offset, $value);  
  
    /** @return void */  
    public function offsetUnset($offset);  
}
```

# Tentative Return Types

```
interface ArrayAccess {  
    public function offsetExists($offset): bool;  
  
    public function offsetGet($offset): mixed;  
  
    public function offsetSet($offset, $value): void;  
  
    public function offsetUnset($offset): void;  
}
```

# Tentative Return Types

```
interface ArrayAccess {  
    public function offsetExists($offset): bool;  
    public function offsetGet($offset): mixed;  
    public function offsetSet($offset, $value): void;  
    public function offsetUnset($offset): void;  
}
```

```
class Foo implements ArrayAccess {  
    public function offsetExists(mixed $offset) {}  
    // ...  
}
```

# Tentative Return Types

```
interface ArrayAccess {  
    public function offsetExists($offset): bool;  
    public function offsetGet($offset): mixed;  
    public function offsetSet($offset, $value): void;  
    public function offsetUnset($offset): void;  
}
```

```
class Foo implements ArrayAccess {  
    public function offsetExists(mixed $offset) {}  
    // ...  
}
```

Deprecated: Return type of Test::offsetExists(mixed \$offset) should either be compatible with ArrayAccess::offsetExists(mixed \$offset): bool, or the #[\ReturnTypeWillChange] attribute should be used to temporarily suppress the notice in... on line ...

# Tentative Return Types

```
interface ArrayAccess {  
    public function offsetExists($offset): bool;  
    public function offsetGet($offset): mixed;  
    public function offsetSet($offset, $value): void;  
    public function offsetUnset($offset): void;  
}
```

```
class Foo implements ArrayAccess {  
    public function offsetExists(mixed $offset): bool {}  
    // ...  
}
```

# Tentative Return Types

```
interface ArrayAccess {  
    public function offsetExists($offset): bool;  
    public function offsetGet($offset): mixed;  
    public function offsetSet($offset, $value): void;  
    public function offsetUnset($offset): void;  
}
```

```
class Foo implements ArrayAccess {  
    #[\ReturnWillChange]  
    public function offsetExists(mixed $offset){}  
    // ...  
}
```



# \$GLOBALS Variable Restrictions

```
$GLOBALS = get_new_vars();
```

```
Fatal error: $GLOBALS can only  
be modified using the  
$GLOBALS[$name] = $value syntax  
in ... on line ...
```



# \$GLOBALS Variable Restrictions

```
$GLOBALS = get_new_vars();  
$GLOBALS = [];  
$GLOBALS = ['foo' => 1, 'bar' => 2];  
$GLOBALS =& $new_vars;  
list($GLOBALS) = [1];  
foreach ($new_var_c as $GLOBALS) {}  
unset($GLOBALS);
```

Fatal error: \$GLOBALS can only be modified using the \$GLOBALS[\$name] = \$value syntax in ... on line ...



# Resource to Object Migrations

## Extension

[GD](#)

[FTP](#)

[IMAP](#)

[fileinfo](#)

[Pspell](#)

[Pspell](#)

[LDAP](#)

[LDAP](#)

[LDAP](#)

[PgSQL](#)

[PgSQL](#)

[PgSQL](#)

## resource (PHP < 8.1)

gd font (*integer*)

ftp

imap

file\_info

pspell (*int*)

pspell config (*int*)

ldap link

ldap result

ldap result entry

pgsql link

pgsql result

pgsql large object

## object (PHP >= 8.1)

[GdFont](#)

[FTP\Connection](#)

[IMAP\Connection](#)

[fileinfo](#)

[Pspell\Dictionary](#)

[Pspell\Config](#)

[LDAP\Connection](#)

[LDAP\Result](#)

[LDAP\ResultEntry](#)

[\PgSql\Connection](#)

[\PgSql\Result](#)

[\PgSql\Lob](#)

# Deprecations



- Passing `null` to non-nullable internal function parameters is deprecated
- Return types in PHP built-in class methods and deprecation notices
- Automatic conversion of "false" into an empty array on write operands is deprecated.
- Serializable interface deprecated
- Implicit incompatible float to int conversion is deprecated
- Calling a static method or accessing a static property directly on a trait is deprecated
- `mysqli::get_client_info` method and `mysqli_get_client_info($param)` is deprecated
- `date_sunrise`, `date_sunset` functions and related INI settings are deprecated
- `strftime()` and `gmstrftime()` functions are deprecated
- `mhash*()` functions (hash extension) are deprecated
- `filter.default` and `filter.default_options` INI settings are deprecated
- `PDO::FETCH_SERIALIZE` is deprecated
- `auto_detect_line_endings` INI directive is deprecated
- `MySQLi: mysqli_driver->driver_version` property is deprecated

# Backwards Compatibility



Drupal 10

Drupal 9.3

Drupal 8.9 – Until Nov 2021

Drupal 7 – Until Nov 2022



# Issues for Drupal core

[Create a new issue](#) [E-mail notifications](#)

**Search for**

**Assigned**

**Submitted by**

**Followers**

**Status**

- Open issues -
- Closed issues -
- Active
- Needs work

**Priority**

- Critical
- Major
- Normal
- Minor

**Category**

- Bug report
- Task
- Feature request
- Support request

**Version**

- Any 10.\* -
- Any 9.\* -
- Any 8.\* -
- Any 7.\* -

**Component**

- ajax system
- asset library system
- base system
- batch system

**Issue tags**

Is one of

Search

Displaying 1 - 14 of 14

Last





# Trying Out PHP 8.1



# Try it online with [3v4l.org](https://3v4l.org)

**3v4l** run code in 300+ PHP versions simultaneously [sponsor](#) | [bughunt](#) | [about](#)

Untitled

```
1 <?php
2
3 enum PostStatuses {
4     case DRAFT;
5     case PENDING;
6     case RETURNED;
7     case PUBLISHED;
8 }
9
10 echo PostStatuses::DRAFT->name;
```

eol versions

[eval\(\);](#) or quick preview in [git.master](#)

Preview

Output for git.master | released 2021-04-22 | took 20 ms, 16.84 MiB

```
DRAFT
```

# Nightly Docker Images

```
docker pull phpdaily/php:8.1-dev
```



# Compiled Windows Binaries

<https://windows.php.net/qa/>

<https://www.apachehaus.com/cgi-bin/download.plx>

## PHP 8.1 (8.1.0RC3)

[Download source code](#) [25.13MB]

[Download tests package \(phpt\)](#) [14.93MB]

### VS16 x64 Non Thread Safe (2021-Sep-29 09:11:42)

- [Zip](#) [28.74MB]

sha256: 3221a445cdc85d112788105cc03632e8a32087f76ea7d00b9a9d0461bc0b743a

- [Debug Pack](#) [23.59MB]

sha256: ce9ed6a3f60af95cd793c45c7dfa2e5ac932085bdee3a452712b9b849145b9bf

- [Development package \(SDK to develop PHP extensions\)](#) [1.2MB]

sha256: dd88c9be7202a330b3c8cf6004b188ee62f5b14675723370cb82ebfd66a7c255

### VS16 x64 Thread Safe (2021-Sep-29 09:16:36)

- [Zip](#) [28.84MB]

sha256: 4bcae7824d62d86a32200bbd49a5e4fcac703af18274ee740bfc8f8b9c58bcba

- [Debug Pack](#) [23.59MB]

sha256: 71949221bba869bfa908979b366174b414b45c0931a17a9415fa2fd7ed063d70

- [Development package \(SDK to develop PHP extensions\)](#) [1.21MB]

sha256: 8f1e0e73389e2c9e950a19da94b9b16d01422fec394a97c5ee9cf631deff9b21

## Apache 2.4.x OpenSSL 1.1.1 VS16

Built using C sources from the ASF and OpenSSL on Visual Studio 2019 (VS16).

**Note: VS16 binaries do not run on Windows XP or Server 2003**

See readme\_first.html file for details.

### Apache 2.4.49

with OpenSSL 1.1.1, brotli 1.0.9, nghttp 1.44.0, Zlib 1.2.11, PCRE 8.45

httpd-2.4.49-o111i-x86-vs16.zip

11.4 MB

Download Locations



[SHA1 Checksum](#)

### Apache 2.4.49 x64

with OpenSSL 1.1.1, brotli 1.0.9, nghttp 1.44.0, Zlib 1.2.11, PCRE 8.45

httpd-2.4.49-o111i-x64-vs16.zip

12.5 MB

Download Locations



[SHA1 Checksum](#)



# Self-compile PHP from source

```
$ git clone git@github.com:php/php-src.git
$ ./buildconf
$ ./configure
$ make -j$(nproc)
$ ./sapi/cli/php -a
```

```
ayesh@Ayesh-Laptop:/work/php-src$ ./sapi/cli/php -a
Interactive shell

php > var_dump(function_exists('enum_exists'));
bool(true)
php >
```

<https://php.watch/articles/compile-php-ubuntu>

<https://php.watch/articles/compile-php-fedora-rhel-centos>

# Further Resources

- <https://aye.sh/talk/php81-drupalconeur-2021>
- <https://php.watch/versions/8.1/enums>
- <https://php.watch/versions/8.1/fibers>
- <https://php.watch/versions/8.1/readonly>
- <https://php.watch/versions/8.1>
- <https://github.com/phpdaily/php>
- <https://3v4l.org/>
- <https://php.watch/articles/compile-php-ubuntu>
- <https://php.watch/articles/compile-php-fedora-rhel-centos>

# Questions?

No question is too small.



**#DrupalConEur** @Ayeshlive [ayesh@php.watch](mailto:ayesh@php.watch)

<https://aye.sh/talk/php81-drupalconeur-2021>

arigatô paldies dziękuję Ďakujem tak  
diolch dankie děkuji mahalo kop khun  
cảm ơn bạn хвала شڪرا لك köszönöm  
a dank gràcies tänan ngiyabonga Баярлалаа dhanyavād  
Дякую ευχαριστώ **THANK YOU** Благодарам  
спасибо takk благодаря  
grazie Mh'gōi Dank u Благодаря ти gracias  
mulțumesc ačiū நன்றி הודת.  
danke takk choukrane faleminderit 谢谢  
teşekkür ederim obrigado kiitos  
Հնրհակալությունս terima kasih hvala grazzi