

# PHP 8.1 Enums

#DPC21



Ayesh Karunaratne | <https://aye.sh/talk/dpc2021-php-enums>



# PHP 8.1 Enums



#DPC21



Ayesh Karunaratne | <https://aye.sh/talk/dpc2021-php-enums>


# Ayesh Karunaratne

---


Freelance Software Developer, Security Researcher, Full-time traveler



 Kandy, Sri Lanka - Everywhere

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

 @Ayeshlive | @phpwch

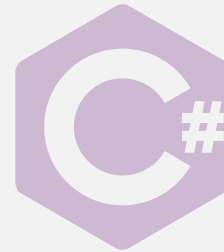
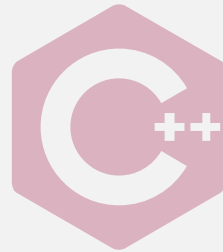
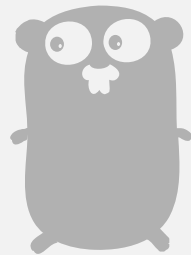
 ayesh@aye.sh

# PHP 8.1 Enums

# PHP 8.1 Enumerations

# PHP 8.1 Enumerations

# PHP 8.1 Enumerations



04 Dec 2020

**Enumerations: RFC Created**

03 Feb 2021

**Voting started**

17 Feb 2021

**Voting ended: 44:7**

18 Jun 2021

**DPC 2021** 

25 Nov 2021

**PHP 8.1**



# PHP 8.1: Enums



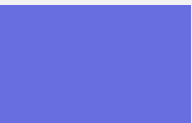
Why we need Enums



How Enums can help



Enums in PHP 8.1



Enum Semantics

Usage Examples 

Trying out Enums today 

Backwards Compatibility 

# Why we need Enums



```
$handle = curl_init();  
$options = [  
    CURLOPT_URL => 'https://example.com',  
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_2_0,  
    CURLOPT_RETURNTRANSFER => true,  
];  
curl_setopt_array($handle, $options);  
curl_exec($handle);
```

```
$handle = curl_init();
$options = [
    CURLOPT_URL => 'https://example.com',
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_2_0,
    CURLOPT_RETURNTRANSFER => true,
];
curl_setopt_array($handle, $options);
curl_exec($handle);
```

```
$handle = curl_init();
$options = [
    CURLOPT_URL => 'https://example.com',
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_2_0,
    CURLOPT_RETURNTRANSFER => true,
];
var_dump($options);
curl_setopt_array($handle, $options);
curl_exec($handle);
```

```
$handle = curl_init();
$options = [
    CURLOPT_URL => 'https://example.com',
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_2_0,
    CURLOPT_RETURNTRANSFER => true,
];
var_dump($options);
curl_setopt_array($handle, $options);
curl_exec($handle);
```

```
array(3) {
    [10002]=> string(19) "https://example.com"
    [84]=> int(3)
    [19913]=> bool(true)
}
```

```
define ('CURLOPT_URL', 10002);  
define ('CURLOPT_HTTP_VERSION', 84);  
define ('CURL_HTTP_VERSION_1_1', 2);  
define ('CURL_HTTP_VERSION_2_0', 3);  
define ('CURLOPT_RETURNTRANSFER', 19913);
```

```
$handle = curl_init();  
$options = [  
    CURLOPT_URL => 'https://example.com',  
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_2_0,  
    CURLOPT_RETURNTRANSFER => true,  
];  
var_dump($options);  
curl_setopt_array($handle, $options);  
curl_exec($handle);
```

```
array(3) {  
    [10002]=> string(19) "https://example.com"  
    [84]=> int(3)  
    [19913]=> bool(true)  
}
```

```
define ('CURLOPT_URL', 10002);
define ('CURLOPT_HTTP_VERSION', 84);
define ('CURL_HTTP_VERSION_1_1', 2);
define ('CURL_HTTP_VERSION_2_0', 3);
define ('CURLOPT_RETURNTRANSFER', 19913);
```

```
$handle = curl_init();
$options = [
    CURLOPT_URL => 'https://example.com',
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_2_0,
    CURLOPT_RETURNTRANSFER => true,
];
var_dump($options);
curl_setopt_array($handle, $options);
curl_exec($handle);
```

```
array(3) {
    [10002]=> string(19) "https://example.com"
    [84]=> int(3)
    [19913]=> bool(true)
}
```



```
define ('CURLOPT_URL', 10002);
define ('CURLOPT_HTTP_VERSION', 84);
define ('CURL_HTTP_VERSION_1_1', 2);
define ('CURL_HTTP_VERSION_2_0', 3);
define ('CURLOPT_RETURNTRANSFER', 19913);
```

```
$handle = curl_init();
$options = [
    CURLOPT_URL => 'https://example.com',
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_2_0,
    CURLOPT_RETURNTRANSFER => true,
];
var_dump($options);
curl_setopt_array($handle, $options);
curl_exec($handle);
```

```
array(3) {
    [10002] => string(19) "https://example.com"
    [84] => int(3)
    [19913] => bool(true)
}
```

```
function curl_setopt(CurlHandle $handle, int $option, mixed $value) : bool {}
```

```
function curl_setopt(CurlHandle $handle, int $option, mixed $value) : bool {}
```

```
function curl_setopt(CurlHandle $handle, int $option, mixed $value) : bool {}
```

```
curl_setopt($handle, 10002, 'https://example.com');  
10002 - CURLOPT_URL
```

```
curl_setopt($handle, 10003, 'https://example.com');
```

PHP Error: curl\_setopt(): Argument #2 (\$option) is not a valid cURL option in ... on line ...

```
curl_setopt($handle, 10004, 'https://example.com');  
10004 - CURLOPT_PROXY
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public string $status;  
  
    public function updateStatus(string $status): void {}  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public string $status;  
  
    public function updateStatus(string $status): void {}  
}
```

```
$post = new Post();  
$post->updateStatus(Post::POST_STATUS_PUBLISHED);
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public string $status;  
  
    public function updateStatus(string $status): void {}  
}
```

```
$post = new Post();  
$post->updateStatus('returned');
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public string $status;  
  
    public function updateStatus(string $status): void {}  
}
```

```
$post = new Post();  
$post->updateStatus('returned');
```



```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public string $status;  
  
    public function updateStatus(string $status): void {}  
}
```

```
class Post {
    public const POST_STATUS_DRAFT = 'draft';
    public const POST_STATUS_PENDING = 'pending';
    public const POST_STATUS_RETURNED = 'returned';
    public const POST_STATUS_PUBLISHED = 'published';

    public string $status;

    public function updateStatus(string $status): void {
        if ( $status !== static::POST_STATUS_DRAFT
            && $status !== static::POST_STATUS_PENDING
            && $status !== static::POST_STATUS_RETURNED
            && $status !== static::POST_STATUS_PUBLISHED
        ) {
            throw new InvalidArgumentException('Invalid state');
        }
    }
}
```

```
class Post {
    public const POST_STATUS_DRAFT = 'draft';
    public const POST_STATUS_PENDING = 'pending';
    public const POST_STATUS_RETURNED = 'returned';
    public const POST_STATUS_PUBLISHED = 'published';

    public string $status;

    public function updateStatus(string $status): void {
        if ( $status !== static::POST_STATUS_DRAFT
            && $status !== static::POST_STATUS_PENDING
            && $status !== static::POST_STATUS_RETURNED
            && $status !== static::POST_STATUS_PUBLISHED
        ) {
            throw new InvalidArgumentException('Invalid state');
        }
    }
}
```

[https://en.wikipedia.org/wiki/Open-closed\\_principle](https://en.wikipedia.org/wiki/Open-closed_principle)

# How Enums Can Help



```
type PostStatus = "draft" | "pending" | "returned" | "published";
```



```
type PostStatus = "draft" | "pending" | "returned" | "published";  
function updateStatus(status: PostStatus) {}
```



```
type PostStatus = "draft" | "pending" | "returned" | "published";  
function updateStatus(status: PostStatus) {}
```

```
type PostStatus = "draft" | "pending" | "returned" | "published";  
function updateStatus(status: PostStatus) {}
```

```
updateStatus("draft");
```



```
updateStatus("potato");
```

Argument of type '"potato"' is not assignable to parameter of type 'PostStatus'.





```
type PostStatus = "draft" | "pending" | "returned" | "published";
```

```
function updateStatus(status: PostStatus) {}
```

```
updateStatus("p");
```

🔑 pending pending

🔑 published

TS

```
enum PostStatus {  
    DRAFT,  
    PENDING,  
    PUBLISHED,  
    RETURNED,  
};  
  
function updateStatus(status: PostStatus) {  
}  
  
updateStatus(PostStatus.DRAFT);
```

```
enum PostStatus {  
    DRAFT = "draft",  
    PENDING = "pending",  
    PUBLISHED = "published",  
    RETURNED = "draft",  
};  
  
function updateStatus(status: PostStatus) {  
}  
  
updateStatus(PostStatus.DRAFT);
```



```
enum PostStatus {  
    DRAFT = "draft",  
    PENDING = "pending",  
    PUBLISHED = "published",  
    RETURNED = "draft",  
};
```



```
enum PostStatus {  
    DRAFT = "draft";  
    PENDING = "pending";  
    PUBLISHED = "published";  
    RETURNED = "draft";  
};
```



```
enum PostStatus {  
  
}
```



```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```



```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public string $status;  
  
    public function updateStatus(string $status): void {  
        if ( $status !== static::POST_STATUS_DRAFT  
            && $status !== static::POST_STATUS_PENDING  
            && $status !== static::POST_STATUS_RETURNED  
            && $status !== static::POST_STATUS_PUBLISHED  
        ) {  
            throw new InvalidArgumentException('Invalid state');  
        }  
    }  
}  
  
$post = new Post();  
$post->updateStatus(\Post::POST_STATUS_PUBLISHED);
```

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public string $status;  
  
    public function updateStatus(string $status): void {  
        if ( $status !== static::POST_STATUS_DRAFT  
            && $status !== static::POST_STATUS_PENDING  
            && $status !== static::POST_STATUS_RETURNED  
            && $status !== static::POST_STATUS_PUBLISHED  
        ) {  
            throw new InvalidArgumentException('Invalid state');  
        }  
    }  
}  
  
$post = new Post();  
$post->updateStatus(\Post::POST_STATUS_PUBLISHED);
```



```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public string $status;  
  
    public function updateStatus(string $status): void {  
        if ( $status !== static::POST_STATUS_DRAFT  
            && $status !== static::POST_STATUS_PENDING  
            && $status !== static::POST_STATUS_RETURNED  
            && $status !== static::POST_STATUS_PUBLISHED  
        ) {  
            throw new InvalidArgumentException('Invalid state');  
        }  
    }  
}  
  
$post = new Post();  
$post->updateStatus(\Post::POST_STATUS_PUBLISHED);
```

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public PostStatus $status;  
  
    public function updateStatus(string $status): void {  
        if ( $status !== static::POST_STATUS_DRAFT  
            && $status !== static::POST_STATUS_PENDING  
            && $status !== static::POST_STATUS_RETURNED  
            && $status !== static::POST_STATUS_PUBLISHED  
        ) {  
            throw new InvalidArgumentException('Invalid state');  
        }  
    }  
}  
  
$post = new Post();  
$post->updateStatus(\Post::POST_STATUS_PUBLISHED);
```

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public PostStatus $status;  
  
    public function updateStatus(PostStatus $status): void {  
        if ( $status !== static::POST_STATUS_DRAFT  
            && $status !== static::POST_STATUS_PENDING  
            && $status !== static::POST_STATUS_RETURNED  
            && $status !== static::POST_STATUS_PUBLISHED  
        ) {  
            throw new InvalidArgumentException('Invalid state');  
        }  
    }  
}  
  
$post = new Post();  
$post->updateStatus(\Post::POST_STATUS_PUBLISHED);
```

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public PostStatus $status;  
  
    public function updateStatus(PostStatus $status): void {  
        if ( $status !== static::POST_STATUS_DRAFT  
            && $status !== static::POST_STATUS_PENDING  
            && $status !== static::POST_STATUS_RETURNED  
            && $status !== static::POST_STATUS_PUBLISHED  
        ) {  
            throw new InvalidArgumentException('Invalid state');  
        }  
    }  
}  
  
$post = new Post();  
$post->updateStatus(\Post::POST_STATUS_PUBLISHED);
```

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public PostStatus $status;  
  
    public function updateStatus(PostStatus $status): void {  
        if ( $status !== static::POST_STATUS_DRAFT  
            && $status !== static::POST_STATUS_PENDING  
            && $status !== static::POST_STATUS_RETURNED  
            && $status !== static::POST_STATUS_PUBLISHED  
        ) {  
            throw new InvalidArgumentException('Invalid state');  
        }  
    }  
}  
  
$post = new Post();  
$post->updateStatus(\Post::POST_STATUS_PUBLISHED);
```

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public const POST_STATUS_DRAFT = 'draft';  
    public const POST_STATUS_PENDING = 'pending';  
    public const POST_STATUS_RETURNED = 'returned';  
    public const POST_STATUS_PUBLISHED = 'published';  
  
    public PostStatus $status;  
  
    public function updateStatus(PostStatus $status): void {  
        if ( $status !== static::POST_STATUS_DRAFT  
            && $status !== static::POST_STATUS_PENDING  
            && $status !== static::POST_STATUS_RETURNED  
            && $status !== static::POST_STATUS_PUBLISHED  
        ) {  
            throw new InvalidArgumentException('Invalid state');  
        }  
    }  
}  
  
$post = new Post();  
$post->updateStatus(PostStatus::PUBLISHED);
```

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
class Post {  
    public PostStatus $status;  
  
    public function updateStatus(PostStatus $status): void {  
    }  
}  
  
$post = new Post();  
$post->updateStatus(\PostStatus::PUBLISHED);
```

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
function setIsSponsored(bool $sponsored): void {  
}
```

```
function isSponsored(): bool {  
}
```

```
setIsSponsored(true);  
setIsSponsored(false);
```



# Enums in PHP 8.1



- 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;  
}
```

- 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: 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**

```
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 = '♣';
```

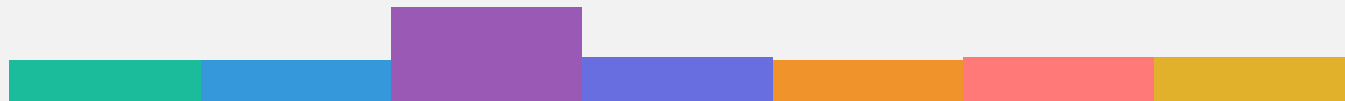
```
    case Diamonds = '♦';
```

```
    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) "♣"
```

# Unit Enums

```
enum PostStatus {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```



# Unit Enums

```
enum PostStatus implements UnitEnum {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

# Unit Enums

```
interface UnitEnum {  
    public static function cases(): array;  
}
```

```
enum PostStatus implements UnitEnum {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

# Unit Enums

```
interface UnitEnum {  
    public static function cases(): array;  
}
```

```
enum PostStatus implements UnitEnum {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

# Unit Enums

```
interface UnitEnum {  
    public static function cases(): array;  
}
```

```
enum PostStatus implements UnitEnum {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

# Unit Enums

```
interface UnitEnum {  
    public static function cases(): array;  
}
```

```
enum PostStatus implements UnitEnum {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
PostStatus::cases();
```

# Unit Enums

```
interface UnitEnum {  
    public static function cases(): array;  
}
```

```
enum PostStatus implements UnitEnum {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
PostStatus::cases();
```

```
array(4) {  
    [0]=> enum(PostStatus::DRAFT)  
    [1]=> enum(PostStatus::PENDING)  
    [2]=> enum(PostStatus::RETURNED)  
    [3]=> enum(PostStatus::PUBLISHED)  
}
```

# Unit Enums

```
interface UnitEnum {  
    public static function cases(): array;  
}
```

```
enum PostStatus implements UnitEnum {  
    case DRAFT;  
    case PENDING;  
    case RETURNED;  
    case PUBLISHED;  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
PostStatus::cases();
```

```
array(4) {  
    [0]=> enum(PostStatus::DRAFT)  
    [1]=> enum(PostStatus::PENDING)  
    [2]=> enum(PostStatus::RETURNED)  
    [3]=> enum(PostStatus::PUBLISHED)  
}
```

# Backed Enums

Backed Enums extend Unit Enums

```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```



# Backed Enums

Backed Enums extend Unit Enums

```
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(int|string $value): static;  
    public static function tryFrom(int|string $value): ?static;  
}
```

```
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

# Backed Enums

Backed Enums `extend Unit Enums`

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;  
// "draft"
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;  
// "draft"
```

```
PostStatus::tryFrom('draft');  
PostStatus::from('draft');
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;  
// "draft"
```

```
PostStatus::tryFrom('draft');  
PostStatus::from('draft');
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;  
// "draft"
```

```
PostStatus::tryFrom('draft');  
PostStatus::from('draft');
```



# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;  
// "draft"
```

```
PostStatus::tryFrom('draft');  
PostStatus::from('draft');
```

```
enum(PostStatus::DRAFT)
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;  
// "draft"
```

```
PostStatus::tryFrom('draft');  
PostStatus::from('draft');  
enum(PostStatus::DRAFT)
```

```
PostStatus::tryFrom('potato');  
PostStatus::from('potato');
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;  
// "draft"
```

```
PostStatus::tryFrom('draft');  
PostStatus::from('draft');  
enum(PostStatus::DRAFT)
```

```
PostStatus::tryFrom('potato');  
PostStatus::from('potato');
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {  
    public static function from(  
        int|string $value  
    ): static;  
  
    public static function tryFrom(  
        int|string $value  
    ): ?static;  
}  
  
enum PostStatus: string implements BackedEnum {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
echo PostStatus::DRAFT->name;  
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;  
// "draft"
```

```
PostStatus::tryFrom('draft');  
PostStatus::from('draft');  
enum(PostStatus::DRAFT)
```

```
PostStatus::tryFrom('potato');  
// null
```

```
PostStatus::from('potato');  
Uncaught ValueError: "potato" is not a valid backing value  
for enum "PostStatus"
```

# Backed Enums

Backed Enums extend Unit Enums

```
interface BackedEnum extends UnitEnum {
    public static function from(
        int|string $value
    ): static;

    public static function tryFrom(
        int|string $value
    ): ?static;
}

enum PostStatus: string implements BackedEnum {
    case DRAFT = 'draft';
    case PENDING = 'pending';
    case RETURNED = 'returned';
    case PUBLISHED = 'published';
}
```

```
echo PostStatus::DRAFT->name;
// "DRAFT"
```

```
echo PostStatus::DRAFT->value;
// "draft"
```

```
PostStatus::tryFrom('draft');
PostStatus::from('draft');
enum(PostStatus::DRAFT)
```

```
PostStatus::tryFrom('potato');
// null
```

```
PostStatus::from('potato');
Uncaught ValueError: "potato" is not a valid backing value
for enum "PostStatus"
```

# Enum Semantics



# Enum

Enumerated type that contains a fixed number of members.



# Enum

Enumerated type that contains a fixed number of members.

A type that is supported as parameter, return, and property type in PHP,  
and the type is enforced by PHP itself.



# Enum

Enumerated type that contains a fixed number of members.

All members are contained within a declared Enum.



# Enum

Enumerated type that contains a fixed number of members.

Members of an Enum is fixed at the declaration time.

An enumerated member is identical to the same member everywhere.

Enums must not contain state.

# Enumerated types

```
enum Suit {  
    case Spades;  
    case Hearts;  
    case Clubs;  
    case Diamonds;  
}  
  
function play_card(Suit $suit, string $card) {}  
  
function pick_a_suit(): Suit {  
    return Suit::Spades;  
}  
  
play_card(Suit::Spades, 'A');  
var_dump(pick_a_suit());  
// enum(Suit::Spades)
```

# Enumerated types

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

```
function play_card(Suit $suit, string $card) {}
```

```
function pick_a_suit(): Suit {  
    return Suit::Spades;  
}
```

```
play_card(Fruits::Apple);  
play_card(Languages::English);  
play_card('potato');
```

Fatal error: Uncaught TypeError: play\_card(): Argument #1 (\$suit) must be of type Suit, string given

# Closed Set

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

# Fixed Members

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

Suit::*Spades* === Suit::*Spades*

# Fixed Members

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

```
enum RussianSuit extends Suit {}
```

Parse error: syntax error, unexpected token "extends",  
expecting "{"

# No Properties Allowed

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

Fatal error: Enums may not include properties



# Backed Enums *must* assign values for all cases

```
enum HTTPMethods: string {  
    case GET;  
    case POST;  
}
```

Fatal error: Case GET of backed enum HTTPMethods must have a value

# Enum cases and values *must* be unique

```
enum Test {  
    case F00;  
    case F00;  
}
```

Fatal error: Cannot redefine class constant Test::F00

```
enum Test: string {  
    case F00 = 'baz';  
    case BAR = 'baz';  
}
```

Fatal error: Duplicate value in enum Test for cases F00 and BAR

# 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;  
    }  
  
}
```

# Usage Examples



```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
class Post {  
    private int $id;  
    private string $title;  
    private PostStatus $status;  
  
    public function __construct(  
        int $id,  
        string $title,  
        PostStatus $status  
    ) {  
        // ...  
    }  
  
    public function getStatus(): PostStatus {  
        return $this->status;  
    }  
}
```

```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
class Post {  
    private int $id;  
    private string $title;  
    private PostStatus $status;  
  
    public function __construct(  
        int $id,  
        string $title,  
        PostStatus $status  
    ) {  
        // ...  
    }  
  
    public function getStatus(): PostStatus {  
        return $this->status;  
    }  
}
```

```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
class Post {  
    private int $id;  
    private string $title;  
    private PostStatus $status;  
  
    public function __construct(  
        int $id,  
        string $title,  
        PostStatus $status  
    ) {  
        // ...  
    }  
  
    public function getStatus(): PostStatus {  
        return $this->status;  
    }  
}
```



```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
class Post {  
    private int $id;  
    private string $title;  
    private PostStatus $status;  
  
    public function __construct(  
        int $id,  
        string $title,  
        PostStatus $status  
    ) {  
        // ...  
    }  
  
    public function getStatus(): PostStatus {  
        return $this->status;  
    }  
}
```

```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
class Post {  
    private int $id;  
    private string $title;  
    private PostStatus $status;  
  
    public function __construct(  
        int $id,  
        string $title,  
        PostStatus $status  
    ) {  
        // ...  
    }  
  
    public function getStatus(): PostStatus {  
        return $this->status;  
    }  
}
```

```
$stmt = $pdo->prepare("  
    SELECT *  
    FROM posts  
    WHERE post_status=?");  
$stmt->execute([  
    PostStatus::PUBLISHED->value  
]);  
$post = $stmt->fetch();
```

```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
class Post {  
    private int $id;  
    private string $title;  
    private PostStatus $status;  
  
    public function __construct(  
        int $id,  
        string $title,  
        PostStatus $status  
    ) {  
        // ...  
    }  
  
    public function getStatus(): PostStatus {  
        return $this->status;  
    }  
}
```

```
$stmt = $pdo->prepare("  
    SELECT *  
    FROM posts  
    WHERE post_status=?");  
$stmt->execute([  
    PostStatus::PUBLISHED->value  
]);  
$post = $stmt->fetch();
```

```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
class Post {  
    private int $id;  
    private string $title;  
    private PostStatus $status;  
  
    public function __construct(  
        int $id,  
        string $title,  
        PostStatus $status  
    ) {  
        // ...  
    }  
  
    public function getStatus(): PostStatus {  
        return $this->status;  
    }  
}
```

```
$sql = "  
    INSERT INTO  
        posts (id, title, post_status)  
    VALUES  
        (:id, :title, :post_status)";  
$stmt= $pdo->prepare($sql);  
$stmt->execute([  
    'id' => $post->getId(),  
    'title' => $post->getTitle(),  
    'post_status' => $post->getStatus()->value,  
]);
```

```
enum PostStatus: string {  
    case DRAFT = 'draft';  
    case PENDING = 'pending';  
    case RETURNED = 'returned';  
    case PUBLISHED = 'published';  
}
```

```
class Post {  
    private int $id;  
    private string $title;  
    private PostStatus $status;  
  
    public function __construct(  
        int $id,  
        string $title,  
        PostStatus $status  
    ) {  
        // ...  
    }  
  
    public function getStatus(): PostStatus {  
        return $this->status;  
    }  
}
```

```
$sql = "  
    INSERT INTO  
        posts (id, title, post_status)  
    VALUES  
        (:id, :title, :post_status)";  
$stmt= $pdo->prepare($sql);  
$stmt->execute([  
    'id' => $post->getId(),  
    'title' => $post->getTitle(),  
    'post_status' => $post->getStatus()->value,  
]);
```

```
enum PostStatus: string {
    case DRAFT = 'draft';
    case PENDING = 'pending';
    case RETURNED = 'returned';
    case PUBLISHED = 'published';
}

class Post {
    private int $id;
    private string $title;
    private PostStatus $status;

    public function __construct(
        int $id,
        string $title,
        PostStatus $status
    ) {
        // ...
    }

    public function getStatus(): PostStatus {
        return $this->status;
    }

    public function updateStatus(
        PostStatus $status
    ): void {
    }
}
```

```
$result = [
    'id' => 42,
    'title' => 'PHP Enums',
    'post_status' => 'published',
];

$post = new Post(
    $result['id'],
    $result['title']
);

$post->updateStatus(
    PostStatus::from($result['post_status'])
);
```

```
enum PostStatus: string {
    case DRAFT = 'draft';
    case PENDING = 'pending';
    case RETURNED = 'returned';
    case PUBLISHED = 'published';
}

class Post {
    private int $id;
    private string $title;
    private PostStatus $status;

    public function __construct(
        int $id,
        string $title,
        PostStatus $status
    ) {
        // ...
    }

    public function getStatus(): PostStatus {
        return $this->status;
    }

    public function updateStatus(
        PostStatus $status
    ): void {
    }
}
```

```
$result = [
    'id' => 42,
    'title' => 'PHP Enums',
    'post_status' => 'published',
];
```

```
$post = new Post(
    $result['id'],
    $result['title']
);
```

```
$post->updateStatus(
    PostStatus::from($result['post_status'])
);
```

# Trying out Enums today





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

3v4l.org 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
```



# 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>

# Backwards Compatibility



# Enums is a new syntax

Enums is a new syntax introduced in PHP 8.1, and not supported in older PHP versions.

Parse error: syntax error, unexpected identifier "PostStatus"

# User-land PHP implementations

<https://github.com/myclabs/php-enum>

```
use MyCLabs\Enum\Enum;  
  
class PostStatus extends Enum {  
    private const DRAFT = 'draft';  
    private const PENDING = 'pending';  
    private const RETURNED = 'returned';  
    private const PUBLISHED = 'published';  
}
```

# Further Resources

- <https://aye.sh/talk/dpc2021-php-enums>
- <https://php.watch/versions/8.1/enums>
- <https://php.watch/versions/8.1>
- <https://wiki.php.net/rfc/enumerations>
- <https://phpinternals.news/73>
- <https://github.com/php/php-src/pull/6489/>
- <https://externals.io/message/112626>
- <https://github.com/phpdaily/php>
- <https://3v4l.org/>
- <https://php.watch/articles/compile-php-ubuntu>

# Questions?

No question is too small.



**#DPC21 @Ayeshlive** [ayesh@php.watch](mailto:ayesh@php.watch)

<https://aye.sh/talk/dpc2021-php-enums>



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

# PHP 8.1 Enums

#DPC21



Ayesh Karunaratne | <https://aye.sh/talk/dpc2021-php-enums>

