The ElePHPant’s New Tricks
Features and deprecations that developers using PHP 7 need to be aware of

Cal Evans
cal@calevans.com
@calevans
Why stay up to date with PHP versions?
Why stay up to date with PHP versions?

1. Security
Why stay up to date with PHP versions?

1. Security
2. Speed
Why stay up to date with PHP versions?

1. Security
2. Speed
3. Features
Errors
Errors

Everything in PHP is moving away from “Fatal Error” and is moving to catchable Errors.
Errors

Long running projects can now catch errors and gracefully exit instead of just hitting a fatal error and bailing.
Errors

function call_method($obj) {
    $obj->method();
}

call_method(null); // OOPS!
Errors

```php
try {
    call_method(null); // OOPS!
} catch (\Error $e) {
    // Gracefully shutdown
}
Echo “Done!”;

function call_method($obj)
{
    $obj->method();
}
```
Errors

• Throwable Interface
  • Error Class
    • ArgumentCountError
    • ArithmeticError
    • AssertionError
    • DivisionByZeroError
    • ParseError
    • TypeError ← Strict type hints throw this one
      • You cannot subclass Error in Userland code!
  • Exception Class
    • Userland Subclasses
Multiple Exception catching

• Allows you to not copy and paste code within Catch
• Can come back to bite you
Multiple Exception catching

try {
    call_method(null); // OOPS!
} catch (ThisException | ThatException $e) {
    // Do something
}

Echo “Done!”;
JSON_THROW_ON_ERROR
JSON_THROW_ON_ERROR

A new flag for both `json_encode()` and `json_decode()`

Throws a JsonException if either `json_encode()` or `json_decode()` hits an error.

Exception contains the message that `json_last_error()` would have returned.
$output = json_encode(
    $payload,
    JSON_PRETTY_PRINT | JSON_THROW_ON_ERROR
);

JSON_THROW_ON_ERROR
Type Hinting
Type Hinting

Type Hints create self documented code.

If you don’t implement them for any other reason, implement them because they give others visual hints on how to use your methods.
Type Hinting

- PHP 7.0 Gave us **Scalar** Type Hints
- PHP 7.0 Gave us **Return** Type Hints
- PHP 7.1 Gave us **Nullable** Type Hints
- PHP 7.2 Gave us the **Object** Type Hint
- PHP 7.3 Gives us Widening

- **PHP 7.4 Gives us Typed Properties** 🎉 🎉 🎉
Type Hinting

Seven Eight new type hints

- int
- float
- string
- bool
- array
- callable
- iterable
- object
Type Hinting

public function post($url, $payload)
{
    // do something clever here
}

public function post(string $url, $payload)
{
    // do something clever here
}
Type Hinting

```php
public function post(string $url, array $payload)
{
    // do something clever here
}
```
Type Hinting

public function post(string $url, array $payload)
{
    if (!is_string($url)) {
        throw new \Exception('URL has to be a string');
    }
    // do something clever here
}
Type Hinting

Strict Types Hints reduce overall lines of code

Declaring strict typing only affects Scalar type hints

When PHP enforces type hints, you don’t have to in your code.
Type Hinting

```php
declare(strict_types=1);
```
Type Hinting

declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

try {
    echo "All ints  :" . sum(1,2) . "\n";
    echo "All Floats :" . sum(1.5,2.5) . "\n";
} catch (TypeError $e) {
    echo 'Error : ' . $e->getMessage();
}

echo "\nDone!\n";
Type Hinting

Nullable Type Hints allow us to say that we will pass in “Specifically this or a NULL.”
Type Hinting

```php
public function post(?string $url, ?array $payload)
{
    // do something clever here
}
```
Nullable Type Hints allow us to say that we will pass in “Specifically this or nothing at all.”

Nullable Type Hints are not the same as specifying a default value of NULL.
public function post(?string $url, ?array $payload=[]) {
    // do something clever here
}
Type Hinting

The object type hint

The eighth type of Scalar Type Hints
Type Hinting

```php
public function post(string $url, ?object $payload)
{
    // do something clever here
}

$obj = new MyFunc();
$response = $obj->post('https://example.com', json_decode('{}'));
```
Return Type Hints

Return Types help developers create self documenting code.
public function post(?string $url, ?array $payload=[]):
                      string
{
    // do something clever here
}
Return Type Hints

If specified, the Return Type MUST match the Return Type Hint. If they do not match, you receive a **TypeError**.
Return Type Hints

class Foo
{
    public function post(?string $url, ?array $payload=[]): int
    {
        return "Cal was here";
    }
}

$obj = new Foo();
$obj->bar(); // TypeError!!!
Return Type Hints

Nullable Return Types

Allows us to return “This type or nothing at all”.
Return Type Hints

public function post(?string $url, ?array $payload=[]) :?string
{
    // do something clever here
}
Void Return Type Hints

Self-documenting code. Let’s the next developer know explicitly that the method will not return anything.
Void Return Type Hints

```php
public function post(?string $url, ?array $payload=[]) : void
{
    // do something clever here
}
```
Void Return Type Hints

CANNOT be mixed with the nullable Return Type
Return Type Hints

class Foo
{
    public function post(?string $url, ?array $payload=[]) : ?void
    {
        return "Cal was here";
    }
}

PHP Fatal error: Void type cannot be nullable
Typed Properties

Coming in PHP 7.4!
class User {
    public int $id;
    public string $name;

    public function __construct(int $id, string $name) {
        $this->id = $id;
        $this->name = $name;
    }
}

Typed Properties
Typed Properties

• Types on static properties supported
• VOID and CALLABLE not supported
• Nullable (?) supported
• Without **strict_types**=1, it is coercive, not strict.
• Non-Private properties are Invariant (Cannot change between inheritance)
Typed Properties - Invariance

```csharp
class A {
    private bool $a;
    public int $b;
    public ?int $c;
}

class B extends A {
    public string $a; // legal, because A::$a is private
    public ?int $b;   // ILLEGAL
    public int $c;    // ILLEGAL
}
```
Typed Properties - Default Values

- Default values have to match they type specified
- Default values cannot be null unless null is specified.
- Properties with type hints that do not have default values and do not allow nulls will throw a `TypeError` upon `Read`. 
Typed Properties

This is a large and complex RFC.

We’ve only scratched the surface here.

I recommend you read up on it before PHP 7.4 gets here so you are ready for it.

https://wiki.php.net/rfc/typed_properties_v2
Covariant Returns and Contravariant Parameters
interface A {
    function m(B $z);
}

interface B extends A {
    // permitted
    function m(B$z): A;
}
Covariant Returns and Contravariant Properties

```java
interface X {
    function m(Y $z): X;
}

interface Y extends X {
    // not permitted but type-safe
    function m(X $z): Y;
}
```
Covariant Returns and Contravariant Properties

```java
interface Factory {
    function make(): object;
}

class UserFactory implements Factory {
    function make(): User;
}
```
interface Concatable {
    function concat(Iterator $input);
}

class Collection implements Concatable {
    // accepts all iterables, not just Iterator
    function concat(iterable $input) {
      /* . . . */
    }
}
Null Coalesce Operator
Null Coalesce Operator

• Allows developers to test a variable and return either its value or a default value if it is null.

• Syntactical sugar for a ternary IF

  $thatString = isset($myString) ? $myString : "";

  is now

  $thatString = $myString ?? "";
Null Coalescing Assignment Operator

- Syntactical sugar for the null coalesce operator

```php
$mystring = $myString ?? "value";
```

is now

```php
$mystring ??= "value";
```
Flexible Heredoc and Nowdoc Syntaxes
Flexible Heredoc and Nowdoc Syntaxes

<table>
<thead>
<tr>
<th>Works</th>
<th>Does Not Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>```php</td>
<td>```php</td>
</tr>
</tbody>
</table>
| class foo {
|     public $bar = <<<EOT bar EOT;         | class foo {
| }                                          |     public $bar = <<<EOT bar EOT;         |
| }                                          | }                                  |
Flexible Heredoc and Nowdoc Syntaxes

// no indentation
echo <<<END
  a
  b
  c
END;
/*
  a
  b
  c
*/

// 4 spaces of indentation
echo <<<END
  a
  b
  c
  END;
/*
  a
  b
  c
*/
CSPRNG
CSPRNG

• Reliable, userland Cryptographically Secure Pseudo Random Number Generator

• Until PHP 7.0, PHP developers did not have easy access to cryptographically strong pseudo random numbers. This led to a lot of developers trying to roll their own. (bad idea)

• Starting in PHP 7.0 We now have access to the underlying operating system’s random number generator.
CSPRNG

Two new functions:

```php
c$randomBytes = random_bytes(16);
c$randomInt = random_int(1, 20);
```
$randomString = substr(bintohe\text{x}(\text{random\_bytes(10)}),0,10);
Argon2 Password Hashing
Argon2 Password Hashing

Argon2, the recommended password hashing algorithm by the Password Hashing Competition, is a modern algorithm for securely hashing passwords
Argon2 Password Hashing

Three Parameters to tweak

1. A memory cost that defines memory usage of the algorithm
2. A time cost that defines the execution time of the algorithm and the number of iterations
3. And a parallelism factor, which defines the number of parallel threads
Argon2 Password Hashing

// Argon2i with default cost factors
password_hash('password', PASSWORD_ARGON2ID);

// Argon2i by name with custom cost factors
password_hash('password', PASSWORD_ARGON2ID, ['memory_cost' => 1<<17, 'time_cost' => 4, 'threads' => 2]);
Argon2 Password Hashing

`PASSWORD_ARGON2ID`

Recommended password hashing algorithm

as of PHP 7.3
Debugging PDO Prepared Statements
Debugging PDO prepared statements.

/* Execute a prepared statement by binding PHP variables */
$calories = 150; $colour = 'red';
$sth = $dbh->prepare('SELECT name,
                     colour,
                     calories
FROM fruit
WHERE calories < ?
   AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->bindValue(2, $colour, PDO::PARAM_STR);
$sth->execute();
$sth->debugDumpParams();
Debugging PDO prepared statements.

Output:
SQL: [82] SELECT name,
    colour,
    calories
FROM fruit
WHERE calories < ?
    AND colour = ?

Sent SQL: [88] SELECT name,
    colour,
    calories
FROM fruit
WHERE calories < 150
    AND colour = 'red'

Params: 2
Key: Position #0: paramno=0 name=[0] "" is_param=1
param_type=1
Key: Position #1: paramno=1 name=[0] "" is_param=1
param_type=2
One More Thing...
Preloading

https://wiki.php.net/rfc/preload

By: Dmitry Stogov
Preloading

The problem
Preloading

The solution
Questions?

Cal Evans

- cal@calevans.com
- @calevans
- https://blog.calevans.com
- https://nexcess.net

Managers, if you call your developers “resources”, they get to call you “overhead”.