Callables

- Creation of callables

A callable is a reference to a function or method that is passed to another function as an argument. They are represented with the `callable` type declaration.

```
<?php
function foo(callable $callback) {
    $callback();
}
?>
```

Some functions accept callback functions as a parameter, e.g. `array_map`, `usort`, or `preg_replace_callback`.

Creation of callables

A callable is a type that represents something that can be invoked. Callables can be passed as arguments to functions or methods which expect a callback parameter or they can be invoked directly. The `callable` type cannot be used as a type declaration for class properties. Instead, use a `Closure` type declaration.

Callables can be created in several different ways:

- `Closure` object
- ENTITY containing the name of a function or a method
- ENTITY containing a class name or an `object` in index 0 and the method name in index 1
- ENTITY implementing the __invoke() magic method

A `Closure` object can be created using anonymous function syntax, arrow function syntax, first-class callable syntax, or the Closure::fromCallable method.

> The first-class callable syntax is only available as of PHP 8.1.0.

Callback example using a `Closure`

```
<?php
// Using anonymous function syntax
$double1 = function ($a) {
    return $a * 2;
};

// Using first-class callable syntax
function double_function($a) {
    return $a * 2;
}
$double2 = double_function(...);

// Using arrow function syntax
$double3 = fn($a) => $a * 2;

// Using Closure::fromCallable
$double4 = Closure::fromCallable('double_function');

// Use the closure as a callback here to
// double the size of each element in our range
$new_numbers = array_map($double1, range(1, 5));
print implode(' ', $new_numbers) . PHP_EOL;

$new_numbers = array_map($double2, range(1, 5));
print implode(' ', $new_numbers) . PHP_EOL;

$new_numbers = array_map($double3, range(1, 5));
print implode(' ', $new_numbers) . PHP_EOL;

$new_numbers = array_map($double4, range(1, 5));
print implode(' ', $new_numbers);

?>
```

ENTITY

```
2 4 6 8 10
2 4 6 8 10
2 4 6 8 10
2 4 6 8 10
```

A callable can also be a string containing the name of a function or a static method. Any built-in or user-defined function can be used, except language constructs such as: `array`, `echo`, `empty`, `eval`, `isset`, `list`, `print` or `unset`.

Static class methods can be used without instantiating an `object` of that class by either creating an array with the class name at index 0 and the method name at index 1, or by using the special syntax with the scope resolution operator `::`, as in `'ClassName::methodName'`.

A method of an instantiated `object` can be a callable when provided as an array with the `object` at index 0 and the method name at index 1.

The main difference between a `Closure` object and the `callable` type is that a `Closure` object is scope-independent and can always be invoked, whereas a callable type may be scope-dependent and may not be directly invoked. `Closure` is the preferred way to create callables. As of PHP 8.5.0, `Closure` is a proper subtype of `callable`.

> While `Closure` objects are bound to the scope where they are created, callables referencing class methods as strings or arrays are resolved in the scope where they are called. To create a callable from a private or protected method, which can then be invoked from outside the class scope, use Closure::fromCallable or the first-class callable syntax.

PHP allows the creation of callables which can be used as a callback argument but cannot be called directly. These are context-dependent callables which reference a class method in the inheritance hierarchy of a class, e.g. `'parent::method'` or `["static", "method"]`.

> As of PHP 8.2.0, context-dependent callables are deprecated. Remove the context dependency by replacing `'parent::method'` with `parent::class . '::method'` or use the first-class callable syntax.

Calling various types of callables with `call_user_func`

```
<?php
```

```php
// An example callback function
function my_callback_function() {
    echo 'hello world!', PHP_EOL;
}

// An example callback method
class MyClass {
    static function myCallbackMethod() {
        echo 'Hello World!', PHP_EOL;
    }
}

// Type 1: Simple callback
call_user_func('my_callback_function');

// Type 2: Static class method call
call_user_func(['MyClass', 'myCallbackMethod']);

// Type 3: Object method call
$obj = new MyClass();
call_user_func([$obj, 'myCallbackMethod']);

// Type 4: Static class method call
call_user_func('MyClass::myCallbackMethod');

// Type 5: Static class method call using ::class keyword
call_user_func([MyClass::class, 'myCallbackMethod']);

// Type 6: Relative static class method call
class A {
    public static function who() {
        echo 'A', PHP_EOL;
    }
}

class B extends A {
    public static function who() {
        echo 'B', PHP_EOL;
    }
```

```
}

call_user_func(['B', 'parent::who']); // deprecated as of PHP 8.2.0

// Type 7: Objects implementing __invoke can be used as callables
class C {
    public function __invoke($name) {
        echo 'Hello ', $name;
    }
}

$c = new C();
call_user_func($c, 'PHP!');
?>
```

ENTITY

```
hello world!
Hello World!
Hello World!
Hello World!
Hello World!

Deprecated: Callables of the form ["B", "parent::who"] are deprecated in script on line 44
A
Hello PHP!
```

ENTITY

□□□ #2
□ □□□ □□□ 2026-10-31 17:47:31 UTC
□ □□□ □□□ 2026-10-31 17:47:32 UTC