Properties



- Type declarations
- Readonly properties
- Dynamic properties

Class member variables are called *properties*. They may be referred to using other terms such as *fields*, but for the purposes of this reference *properties* will be used. They are defined by using at least one modifier (such as ???, ???, or, as of PHP 8.1.0, readonly), optionally (except for `readonly` properties), as of PHP 7.4, followed by a type declaration, followed by a normal variable declaration. This declaration may include an initialization, but this initialization must be a constant value.

> [!NOTE] An obsolete way of declaring class properties is by using the `var` keyword instead of a modifier.

> [!NOTE] A property declared without a ??? modifier will be declared as `public`.

Within class methods non-static properties may be accessed by using `-ENTITY` (Object Operator): `$this-ENTITYproperty` (where `property` is the name of the property). Static properties are accessed by using the `::` (Double Colon): `self::$property`. See ??? for more information on the difference between static and non-static properties.

The pseudo-variable `$this` is available inside any class method when that method is called from within an object context. `$this` is the value of the calling object.

Property declarations

```
<?php
class SimpleClass
{
   public $var1 = 'hello ' . 'world';
```

```
   public $var2 = <<<EOD
hello world
EOD;
   public $var3 = 1+2;
   // invalid property declarations:
   public $var4 = self::myStaticMethod();
   public $var5 = $myVar;

   // valid property declarations:
   public $var6 = myConstant;
   public $var7 = [true, false];

   public $var8 = <<<'EOD'
hello world
EOD;

   // Without visibility modifier:
   static $var9;
   readonly int $var10;
}
?>
```

> [!NOTE] There are various functions to handle classes and objects. See the Class/Object Functions reference.

Type declarations

As of PHP 7.4.0, property definitions can include ???, with the exception of `callable`.

Example of typed properties

```
<?php

class User
```

```
{
    public int $id;
    public ?string $name;

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

$user = new User(1234, null);

var_dump($user->id);
var_dump($user->name);

?>
```

ENTITY

```
int(1234)
NULL
```

Typed properties must be initialized before accessing, otherwise an `Error` is thrown.

Accessing properties

```
<?php

class Shape
{
    public int $numberOfSides;
    public string $name;

    public function setNumberOfSides(int $numberOfSides): void
    {
        $this->numberOfSides = $numberOfSides;
```

```
    }

    public function setName(string $name): void
    {
        $this->name = $name;
    }

    public function getNumberOfSides(): int
    {
        return $this->numberOfSides;
    }

    public function getName(): string
    {
        return $this->name;
    }
}

$triangle = new Shape();
$triangle->setName("triangle");
$triangle->setNumberOfSides(3);
var_dump($triangle->getName());
var_dump($triangle->getNumberOfSides());

$circle = new Shape();
$circle->setName("circle");
var_dump($circle->getName());
var_dump($circle->getNumberOfSides());
?>
```

ENTITY

```
string(8) "triangle"
int(3)
string(6) "circle"

Fatal error: Uncaught Error: Typed property Shape::$numberOfSides must not be accessed before
```

```
initialization
```

Readonly properties

As of PHP 8.1.0, a property can be declared with the `readonly` modifier, which prevents modification of the property after initialization. Prior to PHP 8.4.0 a `readonly` property is implicitly private-set, and may only be written to from the same class. As of PHP 8.4.0, `readonly` properties are implicitly `protected(set)`, so may be set from child classes. That may be overridden explicitly if desired.

Example of readonly properties

```
<?php

class Test {
   public readonly string $prop;

   public function __construct(string $prop) {
       // Legal initialization.
       $this->prop = $prop;
   }
}

$test = new Test("foobar");
// Legal read.
var_dump($test->prop); // string(6) "foobar"

// Illegal reassignment. It does not matter that the assigned value is the same.
$test->prop = "foobar";
// Error: Cannot modify readonly property Test::$prop
?>
```

> [!NOTE] The readonly modifier can only be applied to typed properties. A readonly property without type constraints can be created using the ??? type.

> [!NOTE] Readonly static properties are not supported.

A readonly property can only be initialized once, and only from the scope where it has been declared. Any other assignment or modification of the property will result in an `Error` exception.

Illegal initialization of readonly properties

```
<?php
class Test1 {
    public readonly string $prop;
}

$test1 = new Test1;
// Illegal initialization outside of private scope.
$test1->prop = "foobar";
// Error: Cannot initialize readonly property Test1::$prop from global scope
?>
```

> [!NOTE] Readonly properties can only be initialized by a direct assignment. An uninitialized readonly property cannot be initialized by reference, not even from the scope where it has been declared: passing it to a function parameter that expects to be passed by reference, or assigning it by reference, results in an Error exception.

```
<?php

class Test {
    public readonly array $matches;

    public function __construct(string $subject) {
        // Illegal initialization by reference.
        preg_match('/\d+/', $subject, $this->matches);
        // Error: Cannot indirectly modify readonly property Test::$matches
    }
}
```

```
new Test('abc 42');
?>
```

> [!NOTE] Specifying an explicit default value on readonly properties is not allowed, because a readonly property with a default value is essentially the same as a constant, and thus not particularly useful.
>
> ```
> <?php
>
> class Test {
>     // Fatal error: Readonly property Test::$prop cannot have default value
>     public readonly int $prop = 42;
> }
> ?>
> ```

> [!NOTE] Readonly properties cannot be `unset` once they are initialized. However, it is possible to unset a readonly property prior to initialization, from the scope where the property has been declared.

Modifications are not necessarily plain assignments, all of the following will also result in an `Error` exception:

```
<?php

class Test {
    public function __construct(
        public readonly int $i = 0,
        public readonly array $ary = [],
    ) {}
}

$test = new Test;
```

```
$test->i += 1;
$test->i++;
++$test->i;
$test->ary[] = 1;
$test->ary[0][] = 1;
unset($test->ary[0]);
$ref =& $test->i;
$test->i =& $ref;
byRef($test->i);
foreach ($test as &$prop);
?>
```

However, readonly properties do not preclude interior mutability. Objects (or resources) stored in readonly properties may still be modified internally:

```
<?php

class Test {
    public function __construct(public readonly object $obj) {}
}

$test = new Test(new stdClass);
// Legal interior mutation.
$test->obj->foo = 1;
// Illegal reassignment.
$test->obj = new stdClass;
?>
```

As of PHP 8.3.0, readonly properties can be reinitialized when cloning an object using the __clone() method.

Readonly properties and cloning

```
<?php
class Test1 {
    public readonly ?string $prop;
```

```
    public function __clone() {
        $this->prop = null;
    }

    public function setProp(string $prop): void {
        $this->prop = $prop;
    }
}

$test1 = new Test1;
$test1->setProp('foobar');

$test2 = clone $test1;
var_dump($test2->prop); // NULL
?>
```

> [!NOTE] As of PHP 8.4.0, indirect modification of a readonly property within __clone(), such as acquiring a reference to it (`$ref = ENTITY$this->prop`), is no longer allowed. This was already forbidden during readonly initialization.

Dynamic properties

If trying to assign to a non-existent property on an ENTITY, PHP will automatically create a corresponding property. This dynamically created property will *only* be available on this class instance.

Dynamic properties are supported by stdClass and its subclasses. Other classes may enable similar behavior with AllowDynamicProperties or property overloading.

> [!WARNING] Creating a dynamic property is deprecated as of PHP 8.2.0, unless the class is marked with the `#[\AllowDynamicProperties]` attribute. It is recommended to declare the property instead. To handle arbitrary property names, the class should implement the magic methods __get() and __set().

> Marking the class with the attribute remains an option, but is generally considered a last resort.

□□□ #2
□ □□□ □□□ 2026-10-31 17:40:38 UTC
□ □□□ □□□ 2026-10-31 17:40:40 UTC