The operator class interface

Every method and argument the template engine uses on a template operator object - operatorList, namedParameterPerOperator, namedParameterList, modify with its eight arguments, operatorTemplateHints and element...

Source file: doc/eng-us/help/template-operators/operator-class-reference.md Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/template-operators/operator-class-reference.md

A template operator is any PHP object with the methods below; there is no base class or interface to implement. Everything here is read from `lib/eztemplate/classes/eztemplate.php` (`processOperator()`, `operatorParameterList()`, `loadAndRegisterOperators()`) and `lib/eztemplate/classes/eztemplatecompiler.php`. For a guided first operator see Creating an operator, step by step.

Contents

1. The sample class
2. The registration entry
3. operatorList
4. namedParameterPerOperator and namedParameterList
5. modify
6. operatorTemplateHints
7. Element transformations
8. operatorCompiledStaticData

The sample class

The examples use `C11ExampleOperators`, a class written for these pages. Not run on a development installation as an installed extension: the script that rendered the examples registered the class on the template engine in memory (the same entry an extension's `eztemplateautoload.php` would add), loaded from a scratch folder; compiled results were compiled into a scratch folder. Its operators:

Operator	What it shows
`exp_initials`	an operator that works on its piped input
`exp_reading_time( words_per_minute )`	an optional parameter with a default
`exp_clamp( min, max )`	required parameters
`exp_parameters_seen( first, second )`	what `modify()` receives
`exp_wrong_hint`	what a wrong hint does to compiled templates
`exp_twice`	an element transformation (inline PHP in compiled templates)

The registration entry

An entry of `$eZTemplateOperatorArray` in an `eztemplateautoload.php` (Registering operators):

Key	Meaning
`class`	the class to create (`new $class()`, or `new $class( $class_parameter )`)
`script`	file to `include_once` when the class does not exist yet; optional when the class is in the autoload arrays
`class_parameter`	optional, passed to the constructor
`function`	instead of `class`: a PHP function that returns the operator object (the kernel's `eZPHPOperatorInit`)
`operator_names`	the names this entry registers
`operator_names_function`	instead of `operator_names`: a PHP function that returns them (the kernel's `eZPHPOperatorNameInit` reads `template.ini`)

The engine stores the entry under each name and creates the object the first time a template uses one of them.

operatorList

PHP

```php
function operatorList()
{
    return array( 'exp_initials', 'exp_reading_time', 'exp_clamp', 'exp_parameters_seen', 'exp_wr
ong_hint', 'exp_twice' );
}
```

Required. When the object is created, the engine registers it under every name this returns, which can take names back from other classes (A loaded class takes its names back). Return exactly the names of your registration entry.

namedParameterPerOperator and namedParameterList

PHP

```
function namedParameterPerOperator()
{
    return true;
}

function namedParameterList()
{
    return array(
        'exp_initials' => array(),
        'exp_reading_time' => array(
            'words_per_minute' => array( 'type' => 'integer', 'required' => false, 'default' => 2
00 ) ),
        'exp_clamp' => array(
            'min' => array( 'type' => 'numeric', 'required' => true, 'default' => false ),
            'max' => array( 'type' => 'numeric', 'required' => true, 'default' => false ) ),
        // ...
    );
}
```

- With `namedParameterPerOperator()` returning `true`, `namedParameterList()` is keyed by operator name. Without it (or returning `false`) the list applies to every operator of the class.
- The template passes parameters **by position**; the names only label them in `$namedParameters`.
- `type` is documentation: the engine does not check or convert it. Check values in `modify()`.
- `required`: when the template leaves the parameter out, the engine writes the warning `Parameter '<name>' (<position>) missing` and still calls `modify()`, with the `default`.
- `default`: the value used when the parameter is left out. Leave it out of the definition and the engine passes `null` and warns `Operator '<operator>' has no default value for parameter '<name>'` (once per request in interpreted templates, at compile time for compiled ones).

Optional and required parameters on a development installation:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 89 ) )
     $text = 'one two three four five six seven eight nine ten eleven twelve'}
{$node.name|wash} -> {$node.name|exp_initials}
{$text|exp_reading_time} minute(s) at 200 words a minute, {$text|exp_reading_time( 5 )} at 5
{exp_initials( 'Bold Agency' )}
```

Output:

```
Fit &amp;amp; Healthy -> FH
1 minute(s) at 200 words a minute, 3 at 5
```

Template

```
{150|exp_clamp( 0, 100 )} {-5|exp_clamp( 0, 100 )} {42|exp_clamp( 0, 100 )}
{42|exp_clamp( 0 )}
```

Output:

```
100 0 42
42
```

Logged by the template engine:

```
WARNING eZTemplateOperatorElement: Parameter 'max' (1) missing
WARNING exp_clamp: min and max must be numbers
```

The second line passes only `min`: the engine warns about `max`, passes its default `false`, and the operator's own check refuses the call and leaves the input unchanged (`42`).

modify

PHP

```php
function modify( $tpl, $operatorName, $operatorParameters, $rootNamespace, $currentNamespace,
                 &$operatorValue, $namedParameters, $placement = false )
```

Argument	What it is
`$tpl`	the `eZTemplate` object: for `warning()`, `error()`, `elementValue()`, `variable()`
`$operatorName`	the name used in the template, for classes with several operators
`$operatorParameters`	the **unevaluated** parameter elements, one per parameter written in the template
`$rootNamespace`, `$currentNamespace`	the namespaces to evaluate elements and variables in
`&$operatorValue`	in: the piped input, `null` in the call form; out: the result. Assign, do not return
`$namedParameters`	the evaluated parameters by name, defaults filled in
`$placement`	where the call is in the template, for warnings

What `exp_parameters_seen` reports about its arguments, once called as a function and once piped:

Template

```
{def $b = 'bee'}
{exp_parameters_seen( 'a', $b )}
{'piped'|exp_parameters_seen}
```

Output:

```
value=NULL named={"first":"a","second":"bee"} raw count=2 first raw element type=1 second evaluat
ed again='bee' placement=[[2,0,18],[2,30,48],"var\/tmp\/examples\/ld-op\/03-what-modify-receives.
tpl"]
value='piped' named={"first":"first-default","second":"second-default"} raw count=0 first raw ele
ment type=none second evaluated again=NULL placement=[[3,0,51],[3,27,78],"var\/tmp\/examples\/ld-
op\/03-what-modify-receives.tpl"]
```

Output with the template compiler (compiled into a scratch folder):

```
value=NULL named={"first":"a","second":"bee"} raw count=2 first raw element type=1 second evaluat
ed again='bee' placement=false
value='piped' named={"first":"first-default","second":"second-default"} raw count=0 first raw ele
ment type=none second evaluated again=NULL placement=false
```

What the output shows:

- in the call form `$operatorValue` is `NULL`; piped, it is the input;
- `$namedParameters` holds the evaluated values with the defaults filled in;
- `$operatorParameters` holds element arrays (type `1` is `eZTemplate::TYPE_STRING`), and `$tpl->elementValue( $operatorParameters[1], $rootNamespace, $currentNamespace, $placement )` evaluates one again; use this only for operators that must evaluate a parameter lazily or several times;
- `$placement` is `[[start line, column, position], [end line, column, position], template file]` in an interpreted template and `false` in a compiled one. Pass it to `$tpl->warning( $operatorName, $text, $placement )` anyway: warnings of compiled templates then have no line number, but they are still written.

Rules for `modify()`: assign the result to `$operatorValue`; never echo; return text unescaped and let the template escape it with `|wash`; on bad input write a warning and leave `$operatorValue` as it is or set a neutral value; do not change content or settings from an operator (output is cached, and operators run while pages are compiled and rendered).

operatorTemplateHints

Optional. Tells the template compiler what an operator does, keyed by operator name:

PHP

```php
function operatorTemplateHints()
{
    return array(
        // correct: takes the piped value, returns a value, no parameters
        'exp_initials' => array( 'input' => true, 'output' => true, 'parameters' => false ),
        // wrong on purpose: the operator uses its piped value but says it takes none
        'exp_wrong_hint' => array( 'input' => false, 'output' => true, 'parameters' => false ),
        // ...
    );
}
```

Key	Meaning for the compiler
`input`	the operator uses its piped value. With `input false` and `output true`, the compiler **drops the piped value**
`output`	the operator produces a value. `false` marks the result as null
`parameters`	the operator takes parameters. `false`: the compiler drops any written in the template
`output-type`	the type of the result, for later optimizations
`element-transformation`	the operator can write its own PHP into the compiled template
`element-transformation-func`	the method that does it (default `templateElementTransformation`)
`transform-parameters`	compile the parameters first and hand them to that method
`input-as-parameter`	hand the piped value to the method as the first parameter (`true`: when there is one; `'always'`)
`static`	the result is constant: the compiler calls `operatorCompiledStaticData()` once

The interpreter ignores hints; the compiler trusts them. A wrong hint is therefore a bug that only compiled templates show, which on a default installation means every visitor:

Template

```
{'fit & healthy'|exp_initials} {'fit & healthy'|exp_wrong_hint}
```

Output:

```
FH FIT & HEALTHY
```

Output with the template compiler (compiled into a scratch folder):

```
FH
```

An operator without `operatorTemplateHints()` is compiled into a run-time call of `$tpl->processOperator()`, which calls `modify()` like the interpreter does. That is always correct, only slower; leave hints out until you need the speed.

Element transformations

An operator whose work is simple PHP can write that PHP into the compiled template, so no operator object is called at run time. `exp_twice` multiplies its input by two:

PHP

```
// hints
'exp_twice' => array( 'input' => true, 'output' => true, 'parameters' => false,
                      'element-transformation' => true, 'transform-parameters' => true,
                      'input-as-parameter' => 'always',
                      'element-transformation-func' => 'twiceTransformation' ),

// the method the compiler calls
function twiceTransformation( $operatorName, &$node, $tpl, &$resourceData,
                              $element, $lastElement, $elementList, $elementTree, &$parameters )
{
    $values = array( $parameters[0] );
    return array( eZTemplateNodeTool::createCodePieceElement( "%output% = %1% * 2; // exp_twice,
compiled\n", $values ) );
}
```

`%1%` is replaced by the code of the first value in `$values` (here the piped input), `%output%` by the variable that receives the result. Return `false` (not an array) when the call cannot be transformed, for example when a parameter is not constant (`eZTemplateExecuteOperator::fetchTransform()` does that); the compiler then keeps a run-time call. `modify()` must still implement the operator for interpreted templates:

Template

```
{def $n = 21}
{$n|exp_twice} {4|exp_twice|exp_twice}
```

Output:

```
42 16
```

Output with the template compiler (compiled into a scratch folder):

```
42 16
```

An excerpt of the PHP the compiler generated on a development installation for the second line (`{4|exp_twice|exp_twice}`): the constant was folded into the first multiplication and no operator object is called:

PHP

```php
$var1 = 4 * 2; // exp_twice, compiled
if (! isset( $var1 ) ) $var1 = NULL;
while ( is_object( $var1 ) and method_exists( $var1, 'templateValue' ) )
    $var1 = $var1->templateValue();
$var = $var1 * 2; // exp_twice, compiled
```

The interpreted and compiled results must always be the same; test both. The library's operators are the best reference: `eZTemplateStringOperator` (`phpMap` and `customMap` turn PHP functions into code) and `eZTemplateArithmeticOperator`.

operatorCompiledStaticData

With `'static' => true` in the hints, the compiler calls `operatorCompiledStaticData( $operatorName )` once and writes the returned value into the compiled template as a constant (the template library uses it for `true()` and `false()` in `eZTemplateLogicOperator`). Use it only for values that never change between compilation and the next cache clear, never for user, time or content data.

Related pages

- Creating an operator, step by step
- Registering operators
- Overriding kernel operators
- The template compiler
- Writing template operators (application creators' introduction)
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

> **Become an Exponential Partner**
>
> Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.
>
> Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.