Array operators

The 24 operators of eZTemplateArrayOperator (array, hash, append, prepend, merge, contains, compare, extract, extract_left, extract_right, begins_with, ends_with, implode, explode, repeat, reverse, insert, remove...

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

Arrays and hashes (associative arrays) are built and changed with operators: the template language has no literal `[1, 2, 3]` syntax. All 24 operators on this page are registered by `eZTemplateArrayOperator` (`lib/eztemplate/classes/eztemplatearrayoperator.php`) through `lib/eztemplate/classes/eztemplateautoload.php`.

Most of them work on **arrays and on strings**: `modify()` first looks at the input. When the piped input is an array (or the first parameter is an array) it uses the array branch; when the input is a string it uses the string branch, which works on characters with the multibyte `mb_*` functions. The sections say which branches an operator has. No operator changes the variable it is applied to: store the result with `{set $list = $list|append( 4 )}`.

The examples were rendered by the template interpreter and again by the template compiler on a development installation (template compiling is enabled there). Where the two disagreed, the section says so.

Contents

1. Building: array and hash
2. Adding: append, prepend and merge
3. Testing: contains, compare, begins_with and ends_with
4. Slicing: extract, extract_left and extract_right
5. Joining and splitting: implode and explode
6. Reshaping: repeat, reverse and unique
7. Editing: insert, remove and replace
8. array_sum
9. Deprecated names: array_append, array_prepend and array_merge
10. A worked example: collecting names in a loop
11. Related pages

Building: array and hash

array

Returns a new list of all its parameters, in order. Takes no input.

Parameter	Type	Required	Default
any number of values	mixed	no	none (an empty array)

Generated reference: array.

hash

Returns an associative array from pairs of parameters: key, value, key, value. A key must be a string or a number; any other key type logs "Unknown hash key type" and that pair is skipped. A trailing key without a value is ignored.

Parameter	Type	Required	Default
key, value pairs	string or number, mixed	no	none (an empty array)

Generated reference: hash.

Template

```
{def $list = array( 'red', 'green', 'blue' )
     $map  = hash( 'name', 'Fit & Healthy', 'node_id', 89 )}
array: {$list|implode( ', ' )} ({$list|count} items)
hash: {foreach $map as $key => $value}{$key}={$value|wash} {/foreach}
```

Output:

```
array: red, green, blue (3 items)
hash: name=Fit &amp;amp; Healthy node_id=89
```

Adding: append, prepend and merge

append

- Array: returns the input array with every parameter added at the end. Without an input, the first parameter is the array and the rest are added (`append( $list, 9 )`). With no item to add it logs "Requires at least one item!".
- String: returns the input with every parameter appended.

Parameter	Type	Required	Default
items	mixed	yes, at least one	none

Generated reference: append.

prepend

- Array: like `append`, but the items go in front, in the order given.
- String: returns the named parameter `prepend_string` followed by the input. Only the first parameter is used.

Parameter	Type	Required	Default
`prepend_string` (string input) or items (array input)	mixed	no	`false`

Generated reference: prepend.

Template

```
{def $list = array( 2, 3 )}
$list|append( 4, 5 ) = {$list|append( 4, 5 )|implode( ',' )}
$list|prepend( 0, 1 ) = {$list|prepend( 0, 1 )|implode( ',' )}
append( $list, 9 ) = {append( $list, 9 )|implode( ',' )}
'Hello'|append( ', ', 'world' ) = {'Hello'|append( ', ', 'world' )}
'world'|prepend( 'Hello ' ) = {'world'|prepend( 'Hello ' )}
```

Output:

```
$list|append( 4, 5 ) = 2,3,4,5
$list|prepend( 0, 1 ) = 0,1,2,3
append( $list, 9 ) = 2,3,9
'Hello'|append( ', ', 'world' ) = Hello, world
'world'|prepend( 'Hello ' ) = Hello world
```

Use the piped form. Compiled on a development installation, the call form `append( $list, 9 )` printed `Array,Array` instead of `2,3,9`; the compiled code (`mergeTrans()`, with the input always passed as a parameter) does not treat the first parameter as the target array. All other lines were identical.

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

```
$list|append( 4, 5 ) = 2,3,4,5
$list|prepend( 0, 1 ) = 0,1,2,3
append( $list, 9 ) = Array,Array
'Hello'|append( ', ', 'world' ) = Hello, world
'world'|prepend( 'Hello ' ) = Hello world
```

merge

Array only: PHP `array_merge()` of the input (or an empty array when there is none) with every parameter that is not `null`. Numeric keys are renumbered, string keys of later arrays overwrite earlier ones, so merging hashes is the way to change one key. On a string input it logs "merge works only with arrays.".

Parameter	Type	Required	Default
arrays	array	yes, at least one	none

Generated reference: merge.

Template

```
{def $a = array( 1, 2 )
     $b = array( 3 )
     $h = hash( 'color', 'red', 'size', 'L' )}
$a|merge( $b, array( 4 ) ) = {$a|merge( $b, array( 4 ) )|implode( ',' )}
merge( $a, $b ) = {merge( $a, $b )|implode( ',' )}
$h|merge( hash( 'size', 'XL' ) ) = {foreach $h|merge( hash( 'size', 'XL' ) ) as $k => $v}{$k}={$v
} {/foreach}
```

Output:

```
$a|merge( $b, array( 4 ) ) = 1,2,3,4
merge( $a, $b ) = 1,2,3
$h|merge( hash( 'size', 'XL' ) ) = color=red size=XL
```

Testing: contains, compare, begins_with and ends_with

All four return `true` or `false`. In the examples, `choose( 'false', 'true' )` turns the boolean into a word.

contains

- Array: PHP `in_array( match, input )` (loose comparison).
- String: whether the input contains the `match` text (`mb_strpos`).

Use it piped: the array or string must be the input.

Parameter	Type	Required	Default
match	string	yes	false

Generated reference: contains.

compare

- Array: true when the input and the `compare` array hold the same values, in any order (two `array_diff()` calls; keys and duplicates are not compared).
- String: true when `strcmp()` finds the strings identical.

Parameter	Type	Required	Default
compare	mixed	yes	false

Generated reference: compare.

Template

```
{def $list = array( 'news', 'blog', 'event' )}
$list|contains( 'blog' ) = {$list|contains( 'blog' )|choose( 'false', 'true' )}
$list|contains( 'shop' ) = {$list|contains( 'shop' )|choose( 'false', 'true' )}
'Fit & Healthy'|contains( 'Healthy' ) = {'Fit & Healthy'|contains( 'Healthy' )|choose( 'false', '
true' )}
$list|compare( array( 'event', 'news', 'blog' ) ) = {$list|compare( array( 'event', 'news', 'blog
' ) )|choose( 'false', 'true' )}
'abc'|compare( 'abc' ) = {'abc'|compare( 'abc' )|choose( 'false', 'true' )}
```

Output:

```
$list|contains( 'blog' ) = true
$list|contains( 'shop' ) = false
'Fit & Healthy'|contains( 'Healthy' ) = true
$list|compare( array( 'event', 'news', 'blog' ) ) = true
'abc'|compare( 'abc' ) = true
```

begins_with

- Array: true when the first elements of the input equal the parameters, one by one (`begins_with( 1, 2 )`).
- String: true when the input starts with the `match` text.

Parameter	Type	Required	Default
`match` (string) or the elements to test (array)	mixed	yes	`false`

Generated reference: begins_with.

ends_with

- Array: true when the last elements of the input equal the parameters, in order.
- String: true when the input ends with the `match` text (`mb_strrpos`).

Parameter	Type	Required	Default
`match` (string) or the elements to test (array)	mixed	yes	`false`

Generated reference: ends_with.

Template

```
{def $list = array( 1, 2, 3, 4 )}
$list|begins_with( 1, 2 ) = {$list|begins_with( 1, 2 )|choose( 'false', 'true' )}
$list|ends_with( 3, 4 ) = {$list|ends_with( 3, 4 )|choose( 'false', 'true' )}
$list|ends_with( 2 ) = {$list|ends_with( 2 )|choose( 'false', 'true' )}
'recipes/pasta'|begins_with( 'recipes/' ) = {'recipes/pasta'|begins_with( 'recipes/' )|choose( 'f
alse', 'true' )}
'image.jpg'|ends_with( '.jpg' ) = {'image.jpg'|ends_with( '.jpg' )|choose( 'false', 'true' )}
```

Output:

```
$list|begins_with( 1, 2 ) = true
$list|ends_with( 3, 4 ) = true
$list|ends_with( 2 ) = false
'recipes/pasta'|begins_with( 'recipes/' ) = true
'image.jpg'|ends_with( '.jpg' ) = true
```

Slicing: extract, extract_left and extract_right

extract

- Array: PHP `array_slice( input, extract_start, extract_length )`; without a length, to the end.
- String: `mb_substr()` with the same parameters.

Parameter	Type	Required	Default
`extract_start`	integer	yes	`false`
`extract_length`	integer	no	`false` (to the end)

Generated reference: extract.

extract_left

The first `length` elements, or characters.

Parameter	Type	Required	Default
length	integer	yes	false

Generated reference: extract_left.

extract_right

The last `length` elements, or characters.

Parameter	Type	Required	Default
length	integer	yes	false

Generated reference: extract_right.

Template

```
{def $list = array( 'a', 'b', 'c', 'd', 'e' )}
$list|extract( 1, 3 ) = {$list|extract( 1, 3 )|implode( '' )}
$list|extract( 2 ) = {$list|extract( 2 )|implode( '' )}
$list|extract_left( 2 ) = {$list|extract_left( 2 )|implode( '' )}
$list|extract_right( 2 ) = {$list|extract_right( 2 )|implode( '' )}
'Exponential'|extract( 0, 3 ) = {'Exponential'|extract( 0, 3 )}
'Exponential'|extract_left( 4 ) = {'Exponential'|extract_left( 4 )}
'Exponential'|extract_right( 4 ) = {'Exponential'|extract_right( 4 )}
```

Output:

```
$list|extract( 1, 3 ) = bcd
$list|extract( 2 ) = cde
$list|extract_left( 2 ) = ab
$list|extract_right( 2 ) = de
'Exponential'|extract( 0, 3 ) = Exp
'Exponential'|extract_left( 4 ) = Expo
'Exponential'|extract_right( 4 ) = tial
```

Joining and splitting: implode and explode

implode

Array only: PHP `implode( separator, input )`, a string. On a string input it logs "implode only works with arrays".

Parameter	Type	Required	Default
separator	string	yes	false

Generated reference: implode.

explode

The two branches do different things:

- Array: splits the array in two at position `explode_first` and returns `array( first part, rest )`.
- String: PHP `explode( explode_first, input )`, a list of the pieces between the

separators.

Parameter	Type	Required	Default
explode_first	mixed (position or separator)	yes	false

Generated reference: explode.

Template

```
{def $list = array( 'a', 'b', 'c', 'd' )
     $parts = $list|explode( 1 )}
$list|implode( ' / ' ) = {$list|implode( ' / ' )}
$list|explode( 1 ) = [{$parts[0]|implode( ',' )}] [{$parts[1]|implode( ',' )}]
'red,green,blue'|explode( ',' ) = {'red,green,blue'|explode( ',' )|implode( ' | ' )}
```

Output:

```
$list|implode( ' / ' ) = a / b / c / d
$list|explode( 1 ) = [a] [b,c,d]
'red,green,blue'|explode( ',' ) = red | green | blue
```

Reshaping: repeat, reverse and unique

repeat

- Array: the elements repeated `repeat_times` times, one after the other.
- String: PHP `str_repeat()`.

Parameter	Type	Required	Default
repeat_times	integer	no	1

Generated reference: repeat.

reverse

Array: `array_reverse()`. String: `strrev()`, which reverses bytes, so it breaks multibyte characters. No parameters. Generated reference: reverse.

unique

Array only: `array_unique()`; the keys of the first occurrences are kept. On a string it logs "unique works only with arrays.". No parameters. Generated reference: unique.

Template

```
array( 1, 2 )|repeat( 3 ) = {array( 1, 2 )|repeat( 3 )|implode( ',' )}
'-'|repeat( 10 ) = {'-'|repeat( 10 )}
array( 1, 2, 3 )|reverse = {array( 1, 2, 3 )|reverse|implode( ',' )}
'abc'|reverse = {'abc'|reverse}
array( 'a', 'b', 'a', 'c', 'b' )|unique = {array( 'a', 'b', 'a', 'c', 'b' )|unique|implode( ',' )
}
```

Output:

```
array( 1, 2 )|repeat( 3 ) = 1,2,1,2,1,2
'-'|repeat( 10 ) = ----------
array( 1, 2, 3 )|reverse = 3,2,1
'abc'|reverse = cba
array( 'a', 'b', 'a', 'c', 'b' )|unique = a,b,c
```

Editing: insert, remove and replace

insert

- Array: inserts every parameter after the first at position `insert_position`.
- String: inserts `insert_string` at `insert_position`.

Parameter	Type	Required	Default
`insert_position`	integer	yes	`false`
`insert_string` (and, for arrays, more items)	mixed	yes	`false`

Generated reference: insert.

remove

Removes `length` elements (or characters) starting at `offset`.

Parameter	Type	Required	Default
`offset`	integer	yes	`false`
`length`	integer	no	1

Generated reference: remove.

replace

Removes `length` elements (or characters) at `offset` and puts every parameter after the second in their place (for a string, those parameters joined).

Parameter	Type	Required	Default
`offset`	integer	yes	`false`
`length`	integer	no	`false`
replacement items	mixed	no	none

Generated reference: replace.

Template

```
{def $list = array( 'a', 'b', 'c', 'd' )}
$list|insert( 2, 'X', 'Y' ) = {$list|insert( 2, 'X', 'Y' )|implode( ',' )}
$list|remove( 1 ) = {$list|remove( 1 )|implode( ',' )}
$list|remove( 1, 2 ) = {$list|remove( 1, 2 )|implode( ',' )}
$list|replace( 1, 2, 'X' ) = {$list|replace( 1, 2, 'X' )|implode( ',' )}
'Exponential'|insert( 3, '-' ) = {'Exponential'|insert( 3, '-' )}
'Exponential'|remove( 0, 3 ) = {'Exponential'|remove( 0, 3 )}
'Exponential'|replace( 0, 3, 'IM' ) = {'Exponential'|replace( 0, 3, 'IM' )}
```

Output:

```
$list|insert( 2, 'X', 'Y' ) = a,b,X,Y,c,d
$list|remove( 1 ) = a,c,d
$list|remove( 1, 2 ) = a,d
$list|replace( 1, 2, 'X' ) = a,X,d
'Exponential'|insert( 3, '-' ) = Exp-onential
'Exponential'|remove( 0, 3 ) = onential
'Exponential'|replace( 0, 3, 'IM' ) = IMonential
```

array_sum

array_sum

PHP `array_sum()` of the piped array. Any other input logs "Unknown input type, can only work with arrays". No parameters. Generated reference: array_sum.

Template

```
array( 1, 2, 3.5 )|array_sum = {array( 1, 2, 3.5 )|array_sum}
```

Output:

```
array( 1, 2, 3.5 )|array_sum = 6.5
```

Deprecated names: array_append, array_prepend and array_merge

`lib/eztemplate/classes/eztemplateautoload.php` marks `array_prepend`, `array_append` and `array_merge` as "DEPRECATED/OBSOLETE", replaced by `prepend`, `append` and `merge`. On arrays they still do the same as the new names:

Template

```
array( 1 )|array_append( 2 ) = {array( 1 )|array_append( 2 )|implode( ',' )}
array( 2 )|array_prepend( 1 ) = {array( 2 )|array_prepend( 1 )|implode( ',' )}
array( 1 )|array_merge( array( 2 ) ) = {array( 1 )|array_merge( array( 2 ) )|implode( ',' )}
```

Output:

```
array( 1 )|array_append( 2 ) = 1,2
array( 2 )|array_prepend( 1 ) = 1,2
array( 1 )|array_merge( array( 2 ) ) = 1,2
```

On a string input the string branch knows only the new names, so the old ones log "Unknown operatorname" and return the input unchanged:

Template

```
'abc'|array_append( 'd' ) = {'abc'|array_append( 'd' )}
```

Output:

```
'abc'|array_append( 'd' ) = abc
```

Logged by the template engine:

```
WARNING array_append: Unknown operatorname: array_append
```

The compiled template is more lenient here: compiled on a development installation, the same line printed `abcd`. Do not rely on either; use `append`.

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

```
'abc'|array_append( 'd' ) = abcd
```

A worked example: collecting names in a loop

Building a list in a loop with `set` and `append`, then joining it, on the Recipes category of a development installation (node 144):

Template

```
{def $names = array()}
{foreach fetch( 'content', 'list', hash( 'parent_node_id', 144, 'limit', 3, 'sort_by', array( 'na
me', true() ) ) ) as $child}
{set $names = $names|append( $child.name )}
{/foreach}
Recipes (first three by name): {$names|implode( ', ' )|wash}
```

Output:

```
Recipes (first three by name): Almond French Toast, Avocado Kale Pesto Grilled Cheese, Gluten-Fre
e Shrimp Pad Thai with Tofu
```

Related pages

- Array operators for designers: the everyday uses
- Arithmetic operators: `count` counts arrays
- Foreach loops
- All template operators (generated)
- 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.