String operators

The 15 operators of eZTemplateStringOperator (upcase, downcase, upfirst, upword, count_words, count_chars, trim, simplify, pad, shorten, wrap, break, wash, ord, chr), every wash type (xhtml, email, javascript, pdf), and...

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

The text operators a template uses most, including `wash`, the escaping operator every template that prints user content needs. All 15 are registered by `eZTemplateStringOperator` (`lib/eztemplate/classes/eztemplatestringoperator.php`) through `lib/eztemplate/classes/eztemplateautoload.php`.

All of them work on the **piped input**. An input of `null` (an attribute that is not set) is treated as an empty string. Where an operator has a multibyte variant (`mb_strtoupper`, `mb_strlen`, `mb_substr`), it is used when the `mbstring` extension is loaded, so accented letters are handled as one character.

The PHP string functions such as `strlen`, `substr` or `sprintf` are a different family: PHP string functions.

Contents

1. Case: upcase, downcase, upfirst and upword
2. Counting: count_words and count_chars
3. Cleaning: trim, simplify and pad
4. shorten
5. Line breaks: wrap and break
6. wash
7. Code points: ord and chr
8. Related pages

Case: upcase, downcase, upfirst and upword

upcase

The input in upper case (`mb_strtoupper`). No parameters. Returns a string. Generated reference: upcase.

downcase

The input in lower case (`mb_strtolower`). No parameters. Returns a string. Generated reference: downcase.

upfirst

The first character in upper case, the rest unchanged. With `[CharacterSettings] MBStringExtension=enabled` in `i18n.ini` it works on the internal charset with `mb_*` functions, otherwise with `ucfirst()`. No parameters. Generated reference: upfirst.

upword

The first character of every word (words split at single spaces) in upper case, with the same multibyte rule. No parameters. Generated reference: upword.

Template

```
'fit & healthy'|upcase = {'fit & healthy'|upcase|wash}
'ÜBER Exponential'|downcase = {'ÜBER Exponential'|downcase}
'élan vital'|upfirst = {'élan vital'|upfirst}
'healthy eating for runners'|upword = {'healthy eating for runners'|upword}
```

Output:

```
'fit & healthy'|upcase = FIT &amp;amp; HEALTHY
'ÜBER Exponential'|downcase = über exponential
'élan vital'|upfirst = Élan vital
'healthy eating for runners'|upword = Healthy Eating For Runners
```

Counting: count_words and count_chars

count_words

The number of words, counted as runs of word characters (`preg_match_all( "#(\w+)#" )`), so `&` is not a word. No parameters. Returns an integer. Generated reference: count_words.

count_chars

The number of characters (`mb_strlen`). Unlike the arithmetic `count`, which returns the byte length of a string, `count_chars` counts a multibyte character once. No parameters. Returns an integer. Generated reference: count_chars.

Template

```
'Fit & Healthy recipes'|count_words = {'Fit & Healthy recipes'|count_words}
'Grüße'|count_chars = {'Grüße'|count_chars}
'Grüße'|count = {'Grüße'|count} (the arithmetic count: bytes)
```

Output:

```
'Fit & Healthy recipes'|count_words = 3
'Grüße'|count_chars = 5
'Grüße'|count = 7 (the arithmetic count: bytes)
```

Cleaning: trim, simplify and pad

trim

PHP `trim()`: removes whitespace, or the characters in `chars_to_remove`, from both ends.

Parameter	Type	Required	Default
chars_to_remove	string	no	false (whitespace)

Generated reference: trim.

simplify

Replaces runs of two or more of the same character by one. Without a parameter it collapses spaces. The parameter is put into a regular expression as it is, so pass a plain character, not one that has a meaning in a pattern.

Parameter	Type	Required	Default
char	string	no	false (a space)

Generated reference: simplify.

pad

PHP `str_pad()` to the right, but only when the input is shorter than `desired_length` (measured in bytes).

Parameter	Type	Required	Default
desired_length	integer	no	80
pad_sequence	string	no	' '

Generated reference: pad.

Template

```
[{'   padded   '|trim}]
[{'--title--'|trim( '-' )}]
[{'too    many     spaces'|simplify}]
[{'a---b--c'|simplify( '-' )}]
[{'7'|pad( 3, '0' )}]
[{'Name'|pad( 10 )}]
[{'already long'|pad( 4 )}]
```

Output:

```
[padded]
[title]
[too many spaces]
[a-b-c]
[700]
[Name      ]
[already long]
```

shorten

shorten

Shortens a text that is longer than `chars_to_keep` characters so that the result, including `str_to_append`, has at most that length. With `trim_type` `right` (the default) it cuts the end and trims trailing whitespace before appending; with `middle` it keeps the start and the end and puts `str_to_append` in between. A text that is short enough is returned unchanged.

Parameter	Type	Required	Default
chars_to_keep	integer	no	80
str_to_append	string	no	'...'
trim_type	string (right or middle)	no	'right'

Returns a string. `shorten` counts characters, not words, and can cut a word or an HTML tag in half: shorten plain text and `wash` it afterwards. Generated reference: shorten.

The first line uses the name of the Privacy Policy article on a development installation (node 90):

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 90 ) )}
{$node.name|shorten( 10 )}
{'A long title that does not fit'|shorten( 20 )}
{'A long title that does not fit'|shorten( 20, ' [more]' )}
{'A long title that does not fit'|shorten( 20, '...', 'middle' )}
{'Short'|shorten( 20 )}
```

Output:

```
Privacy...
A long title that...
A long title [more]
A long t...s not fit
Short
```

Line breaks: wrap and break

wrap

PHP `wordwrap()`. Each parameter is passed only when the one before it is given and not empty.

Parameter	Type	Required	Default
wrap_at_position	integer	no	false (PHP's 75)
break_sequence	string	no	false (PHP's "\n")
cut	boolean	no	false

Generated reference: wrap.

break

PHP `nl2br()`: an HTML line break before every newline. No parameters. `break` does not escape: wash first, `$text|wash|break`, or the input's HTML goes through. Generated reference: break.

Template

```
{'The quick brown fox jumps over the lazy dog'|wrap( 15 )}
---
{'The quick brown fox'|wrap( 10, '<br />' )}
---
{'Supercalifragilistic'|wrap( 6, '|', true() )}
---
{"line one
line two"|break}
```

Output:

```
The quick brown
fox jumps over
the lazy dog
---
The quick<br />brown fox
---
Superc|alifra|gilist|ic
---
line one<br />
line two
```

wash

wash

Makes text safe for one kind of output. The type is the only parameter.

Parameter	Type	Required	Default
type	string	no	'xhtml'

Type	What it does (`eZTemplateStringOperator::wash()`)
xhtml	`htmlspecialchars()` with ENT_QUOTES: <, >, &, " and ' become entities
email	the xhtml escaping, then . and @ replaced by `[WashSettings] EmailDotText` and `EmailAtText` of `template.ini`, to hide addresses from harvesters
javascript	`eZTemplateStringOperator::javascriptString()`: the content of a JavaScript string literal; backslash, both quotes, <, >, &, line breaks and the Unicode line and paragraph separators are escaped
pdf	for the PDF export: HTML entities decoded, then spaces, tabs and newlines turned into <C:callSpace>, <C:callTab> and <C:callNewLine> callbacks of the PDF library

Any other type returns the input **unchanged**, without escaping and without a warning, so a typo in the type is a security problem. On a development installation `EmailAtText` is set to a `spamfilter` span, as the example shows.

A compiled template turns `wash` with a constant type into inline PHP (`washTransformation()`); the compiled `pdf` branch only replaces whitespace and does not decode entities first.

Generated reference: wash.

Template

```
xhtml:      {'<b>"Fit" & \'Healthy\'</b>'|wash}
email:      {'info@example.com'|wash( 'email' )}
javascript: {'He said "hi" </script>'|wash( 'javascript' )}
pdf:        {'85 g spinach    end'|wash( 'pdf' )}
unknown:    {'<b>raw</b>'|wash( 'no-such-type' )}
```

Output:

```
xhtml:      &lt;b&gt;&quot;Fit&quot; &amp; &#039;Healthy&#039;&lt;/b&gt;
email:      info<span class="spamfilter">SPAMFILTER</span>@example.com
javascript: He said \042hi\042 \x3C/script\x3E
pdf:        85<C:callSpace>g spinach<C:callTab>end
unknown:    <b>raw</b>
```

Code points: ord and chr

ord

Converts a string into a list of Unicode code points, through the `unicode` codec of `eZTextCodec`. No parameters. Returns an array of integers. Generated reference: ord.

chr

The reverse: a list of code points to a string in the internal charset. No parameters. Returns a string. Generated reference: chr.

Template

```
{def $codes = 'Añ€'|ord}
'Añ€'|ord = {$codes|implode( ', ' )}
back with chr = {$codes|chr}
array( 72, 105 )|chr = {array( 72, 105 )|chr}
```

Output:

```
'Añ€'|ord = 65, 241, 8364
back with chr = Añ€
array( 72, 105 )|chr = Hi
```

Related pages

- String operators for designers
- Wash and escaping: which type to use where
- PHP string functions: search and compare
- Text operators: `concat` and `indent`
- 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.