PHP string functions: transform

The transforming operators of eZTemplateStringsOperator (substr, substr_replace, ristring, rstring, strtr, str_split, str_word_count, str_getcsv, strtok, ltrim, rtrim, lcfirst, chunk_split, addslashes, stripslashes...

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

This page covers the operators of `eZTemplateStringsOperator` (`lib/eztemplate/classes/eztemplatestringsoperator.php`) that cut, replace, split and escape strings. The mapping rule is the same for all of them: the piped input is the PHP function's main string, the template parameters are its remaining arguments in order (how the functions are mapped).

These functions work on **bytes**. For text with accented or other multibyte characters, the array operators `extract`, `insert`, `remove` and `replace` work on characters (Array operators).

Contents

1. Cutting: substr and substr_replace
2. Replacing: ristring, rstring and strtr
3. Splitting: str_split, str_word_count, str_getcsv and strtok
4. Trimming and case: ltrim, rtrim, lcfirst and chunk_split
5. Slashes: addslashes, stripslashes, addcslashes, stripcslashes and quotemeta
6. strip_tags and the registration trap
7. str_shuffle and hebrev
8. str_increment and str_decrement
9. Related pages

Cutting: substr and substr_replace

substr

PHP `substr()`. Without a start it logs "Missing required parameter: start".

Parameter	Type	Required	Default
start	integer (negative counts from the end)	yes	none
length	integer	no	to the end

Returns a string. Generated reference: substr.

substr_replace

PHP `substr_replace()`: replaces the part from start (for length bytes) with the replacement.

Parameter	Type	Required	Default
replace	string	yes	none
start	integer	yes	none
length	integer	no	to the end

Returns a string. Generated reference: substr_replace.

Template

```
{def $s = 'Healthy eating'}
substr( 8 ) = {$s|substr( 8 )}
substr( 0, 7 ) = {$s|substr( 0, 7 )}
substr( -6 ) = {$s|substr( -6 )}
substr_replace( 'Clean', 0, 7 ) = {$s|substr_replace( 'Clean', 0, 7 )}
substr_replace( '!', 14 ) = {$s|substr_replace( '!', 14 )}
```

Output:

```
substr( 8 ) = eating
substr( 0, 7 ) = Healthy
substr( -6 ) = eating
substr_replace( 'Clean', 0, 7 ) = Clean eating
substr_replace( '!', 14 ) = Healthy eating!
```

Replacing: ristring, rstring and strtr

ristring

PHP `str_replace( search, replace, input )`: **case-sensitive** replacement, despite the "i" in its name. Search and replace may be arrays, as in PHP.

Parameter	Type	Required	Default
search	string or array	yes	none
replace	string or array	yes	none

Returns a string. Generated reference: ristring.

rstring

PHP `str_ireplace()`: the **case-insensitive** replacement. Same parameters as `ristring`. Generated reference: rstring.

The names are the opposite of what one would guess; the class comment documents them as `ristring: str_replace` and `rstring: str_ireplace`, and the example confirms it. On a development installation the extension owsimpleoperator also offers a plain `str_replace` operator (owsimpleoperator).

strtr

PHP `strtr()`, in both of its forms: two strings (character by character translation) or one hash (whole strings, longest first). Anything else logs "Requires (from_string, to_string) or a single associative array".

Parameter	Type	Required	Default
from, or a hash of replacements	string or hash	yes	none
to	string	with a from string	none

Returns a string. Generated reference: strtr.

Template

```
{def $s = 'Fit and healthy. Stay Healthy.'}
ristring( 'healthy', 'strong' ) = {$s|ristring( 'healthy', 'strong' )}
rstring( 'healthy', 'strong' ) = {$s|rstring( 'healthy', 'strong' )}
ristring( array( 'Fit', 'Stay' ), array( 'Be', 'Keep' ) ) = {$s|ristring( array( 'Fit', 'Stay' ),
 array( 'Be', 'Keep' ) )}
strtr( 'aeiou', 'AEIOU' ) = {$s|strtr( 'aeiou', 'AEIOU' )}
strtr( hash( 'Fit', 'Strong', 'healthy', 'happy' ) ) = {$s|strtr( hash( 'Fit', 'Strong', 'healthy
', 'happy' ) )}
```

Output:

```
ristring( 'healthy', 'strong' ) = Fit and strong. Stay Healthy.
rstring( 'healthy', 'strong' ) = Fit and strong. Stay strong.
ristring( array( 'Fit', 'Stay' ), array( 'Be', 'Keep' ) ) = Be and healthy. Keep Healthy.
strtr( 'aeiou', 'AEIOU' ) = FIt And hEAlthy. StAy HEAlthy.
strtr( hash( 'Fit', 'Strong', 'healthy', 'happy' ) ) = Strong and happy. Stay Healthy.
```

Splitting: str_split, str_word_count, str_getcsv and strtok

str_split

PHP `str_split()`: the string in chunks of length bytes (default 1). Returns an array. Generated reference: str_split.

str_word_count

PHP `str_word_count()`: with no parameter the number of words; with format 1 a list of the words, with 2 a hash of position to word. An optional second parameter lists extra characters that count as letters.

Parameter	Type	Required	Default
format	integer (0, 1 or 2)	no	0
characters	string	no	none

Generated reference: str_word_count.

str_getcsv

PHP `str_getcsv()`: one line of CSV as an array. The escape character defaults to `''` (none), which avoids the PHP 8.4 deprecation of the old default.

Parameter	Type	Required	Default
separator	string	no	,
enclosure	string	no	"
escape	string	no	''

Generated reference: str_getcsv.

strtok

PHP `strtok( input, token )`: the first token. Because each operator call starts a new `strtok()` run, a template can only get the first piece; use `explode` for all of them. Parameter: token (string, required). Generated reference: strtok.

Template

```
'abcdefg'|str_split( 3 ) = {'abcdefg'|str_split( 3 )|implode( ' | ' )}
'one two three'|str_word_count = {'one two three'|str_word_count}
'one two three'|str_word_count( 1 ) = {'one two three'|str_word_count( 1 )|implode( ',' )}
'a,"b,c",d'|str_getcsv = {'a,"b,c",d'|str_getcsv|implode( ' | ' )}
'a;b;c'|str_getcsv( ';' ) = {'a;b;c'|str_getcsv( ';' )|implode( ' | ' )}
'first second third'|strtok( ' ' ) = {'first second third'|strtok( ' ' )}
```

Output:

```
'abcdefg'|str_split( 3 ) = abc | def | g
'one two three'|str_word_count = 3
'one two three'|str_word_count( 1 ) = one,two,three
'a,"b,c",d'|str_getcsv = a | b,c | d
'a;b;c'|str_getcsv( ';' ) = a | b | c
'first second third'|strtok( ' ' ) = first
```

Trimming and case: ltrim, rtrim, lcfirst and chunk_split

ltrim and rtrim

PHP `ltrim()` and `rtrim()`: whitespace, or the given characters, removed from the left or the right end. Parameter: characters (string, optional). Generated references: ltrim, rtrim. Both ends at once: trim.

lcfirst

PHP `lcfirst()`: the first byte in lower case. No parameters. Generated reference: lcfirst.

chunk_split

PHP `chunk_split()`: inserts the end string after every chunk (default 76 bytes and `"\r\n"`), including after the last one.

Parameter	Type	Required	Default
length	integer	no	PHP's 76
separator	string	no	PHP's "\r\n"

Generated reference: chunk_split.

Template

```
[{'--slug--'|ltrim( '-' )}] [{'--slug--'|rtrim( '-' )}] [{'  both  '|ltrim}]
'Healthy'|lcfirst = {'Healthy'|lcfirst}
'abcdefgh'|chunk_split( 3, '-' ) = {'abcdefgh'|chunk_split( 3, '-' )}
```

Output:

```
[slug--] [--slug] [both  ]
'Healthy'|lcfirst = healthy
'abcdefgh'|chunk_split( 3, '-' ) = abc-def-gh-
```

Slashes: addslashes, stripslashes, addcslashes, stripcslashes and quotemeta

Operator	PHP function	Parameters
addslashes	addslashes(): backslash before ', ", \ and NUL	none
stripslashes	stripslashes()	none
addcslashes	addcslashes(): backslash before the characters of a list (ranges such as A..Z allowed)	characters (required)
stripcslashes	stripcslashes(): C-style escapes such as \t become the characters	none
quotemeta	quotemeta(): backslash before . \ + * ? [^] $ ( )	none

None of these is an escaping method for HTML or JavaScript output; use `wash` for that (String operators). Generated references: addslashes, stripslashes, addcslashes, stripcslashes, quotemeta.

In the template source, a quote inside a single-quoted string is written `\'` and a backslash `\\`:

Template

```
'It\'s "fit"'|addslashes = {'It\'s "fit"'|addslashes}
'It\\\'s'|stripslashes = {'It\\\'s'|stripslashes}
'FOO[bar]'|addcslashes( 'A..Z' ) = {'FOO[bar]'|addcslashes( 'A..Z' )}
'a\\tb'|stripcslashes|bin2hex = {'a\\tb'|stripcslashes|bin2hex}
'1+1=2?'|quotemeta = {'1+1=2?'|quotemeta}
```

Output:

```
'It\'s "fit"'|addslashes = It\'s \"fit\"
'It\\\'s'|stripslashes = It's
'FOO[bar]'|addcslashes( 'A..Z' ) = \F\O\O[bar]
'a\\tb'|stripcslashes|bin2hex = 610962
'1+1=2?'|quotemeta = 1\+1=2\?
```

strip_tags and the registration trap

strip_tags

PHP `strip_tags()`: removes HTML and PHP tags. In `eZTemplateStringsOperator` an optional parameter lists tags to keep (a string such as `'<b><i>'` or an array of names).

Parameter	Type	Required	Default
allowed tags	string or array	no	none

Returns a string. `strip_tags` is not a sanitizer: it keeps attributes of allowed tags and does not escape &. Wash the result before you print it. Generated reference: strip_tags.

Template

```
{def $html = '<p>Fit <b>and</b> <a href="/x">healthy</a></p>'}
strip_tags = {$html|strip_tags}
```

Output:

```
strip_tags = Fit and healthy
```

The registration trap. On a development installation the extension ezmultiupload adds `PHPOperatorList[strip_tags]=strip_tags` to `template.ini` (`extension/ezmultiupload/settings/template.ini.append.php`). The kernel's `kernel/common/eztemplateautoload.php` registers the `PHPOperatorList` names through `eZTemplatePHPOperator`, which passes only one value to the PHP function, so that version ignores the allowed tags. Which of the two classes answers depends on what the request has used before: the template engine loads operator classes lazily, and when any operator of `eZTemplateStringsOperator` (here `strlen`) is used first, loading that class registers all of its names again, including `strip_tags`, and from then on the version with parameters answers. The same template line gives two results:

Template

```
{def $html = '<p>Fit <b>and</b> healthy</p>'}
first use: [{$html|strip_tags( '<b>' )}]

{def $len = 'x'|strlen}
after strlen: [{$html|strip_tags( '<b>' )}]
```

Output:

```
first use: [Fit and healthy]
after strlen: [Fit <b>and</b> healthy]
```

If you need the allowed tags, do not rely on either: strip all tags and add the markup in the template, or register an operator of your own under another name (Registration order and overrides).

str_shuffle and hebrev

str_shuffle

PHP `str_shuffle()`: the bytes in random order (not for anything secret). No parameters. Generated reference: str_shuffle.

hebrev

PHP `hebrev()`: converts logical Hebrew text to visual text, with an optional maximum characters per line (integer). Text without Hebrew characters is returned unchanged. Generated reference: hebrev.

Template

```
{def $s = 'abcdef'|str_shuffle}
str_shuffle keeps the letters: {$s|str_split( 1 )|unique|count} distinct letters, length {$s|strl
en}
hebrev of a Latin text returns it unchanged: {'plain text'|hebrev}
```

Output:

```
str_shuffle keeps the letters: 6 distinct letters, length 6
hebrev of a Latin text returns it unchanged: plain text
```

str_increment and str_decrement

The class adds `str_increment` and `str_decrement` (PHP 8.3) to its operator list when PHP has them, but they are not in the `operator_names` list of `lib/eztemplate/classes/eztemplateautoload.php`. So, like the trap above, they exist only after some other operator of the class has loaded it. Alone, the engine reports them as not registered:

Template

```
[{'a9'|str_increment}]
```

Output:

```
[a9]
```

Logged by the template engine:

```
WARNING : Operator 'str_increment' is not registered
```

After `strlen` has loaded the class:

Template

```
{def $len = 'x'|strlen}
'a9'|str_increment = {'a9'|str_increment}
'Ab'|str_decrement = {'Ab'|str_decrement}
```

Output:

```
'a9'|str_increment = b0
'Ab'|str_decrement = Aa
```

Do not depend on them in a template that has to work everywhere.

Related pages

- PHP string functions: search and compare
- PHP string functions: encode and format
- PHP function operators: `PHPOperatorList`
- Array operators: the multibyte-safe string editing operators
- 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.