PHP string functions: encode and format

The encoding and formatting operators of eZTemplateStringsOperator (bin2hex, hex2bin, convert_uuencode, convert_uudecode, quoted_printable_encode, quoted_printable_decode, htmlentities, html_entity_decode...

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

The last group of `eZTemplateStringsOperator` (`lib/eztemplate/classes/eztemplatestringsoperator.php`): encodings, HTML entities and printf-style formatting. The piped input is the PHP function's main argument, the template parameters the rest, in order (how the functions are mapped).

PHP flag constants such as `ENT_QUOTES` do not exist in templates. Where a function takes flags, the class supplies a sensible default; to pass other flags, give their integer value (`ENT_QUOTES` is 3, `ENT_NOQUOTES` 0).

Contents

1. Hex, uuencode and quoted-printable
2. HTML entities
3. sprintf and vsprintf
4. number_format
5. Related pages

Hex, uuencode and quoted-printable

None of these takes a parameter. All return a string, except the decoders, which return `false` for input they cannot decode (the PHP warning is suppressed).

bin2hex and hex2bin

`bin2hex()`: every byte as two hex digits. `hex2bin()`: the reverse, or `false`. Generated references: bin2hex, hex2bin.

convert_uuencode and convert_uudecode

PHP's uuencoding and decoding (`convert_uudecode` returns `false` on bad input). Generated references: convert_uuencode, convert_uudecode.

quoted_printable_encode and quoted_printable_decode

The quoted-printable encoding of mail bodies (RFC 2045). Generated references: quoted_printable_encode, quoted_printable_decode.

Template

```
'Fit'|bin2hex = {'Fit'|bin2hex}
'466974'|hex2bin = {'466974'|hex2bin}
'zz'|hex2bin = [{'zz'|hex2bin|choose( 'false', 'true' )}]
'Fit'|convert_uuencode|convert_uudecode = {'Fit'|convert_uuencode|convert_uudecode}
'Grüße'|quoted_printable_encode = {'Grüße'|quoted_printable_encode}
'Gr=C3=BC=C3=9Fe'|quoted_printable_decode = {'Gr=C3=BC=C3=9Fe'|quoted_printable_decode}
```

Output:

```
'Fit'|bin2hex = 466974
'466974'|hex2bin = Fit
'zz'|hex2bin = [false]
'Fit'|convert_uuencode|convert_uudecode = Fit
'Grüße'|quoted_printable_encode = Gr=C3=BC=C3=9Fe
'Gr=C3=BC=C3=9Fe'|quoted_printable_decode = Grüße
```

HTML entities

htmlentities

PHP `htmlentities()`: every character that has a named entity is converted. Without parameters it uses `ENT_QUOTES` and UTF-8. For output escaping, `wash` is the operator to use; `htmlentities` also rewrites letters such as `é`, which is rarely wanted in UTF-8 pages.

Parameter	Type	Required	Default
flags	integer	no	`ENT_QUOTES` (3)
encoding	string	no	`'UTF-8'`
double_encode	boolean	no	PHP's `true`

Generated reference: htmlentities.

html_entity_decode

PHP `html_entity_decode()`: entities back to characters. Without parameters it uses `ENT_QUOTES | ENT_HTML5` and UTF-8. A decoded value must be washed again before it is printed in HTML.

Parameter	Type	Required	Default
flags	integer	no	`ENT_QUOTES | ENT_HTML5`
encoding	string	no	`'UTF-8'`

Generated reference: html_entity_decode.

htmlspecialchars_decode

PHP `htmlspecialchars_decode()`: only `&`, `"`, `'`, `<` and `>`, decoded once. Default flags `ENT_QUOTES`; with 0 (`ENT_NOQUOTES`) quotes stay encoded.

Parameter	Type	Required	Default
flags	integer	no	`ENT_QUOTES` (3)

Generated reference: htmlspecialchars_decode.

Template

```
'Café & "Bar"'|htmlentities = {'Café & "Bar"'|htmlentities}
'Caf&eacute; &amp; &quot;Bar&quot;'|html_entity_decode = {'Caf&eacute; &amp; &quot;Bar&quot;'|html_entity_decode}
'&lt;b&gt; &amp;amp;'|htmlspecialchars_decode = {'&lt;b&gt; &amp;amp;'|htmlspecialchars_decode}
'it&#039;s'|htmlspecialchars_decode( 0 ) = {'it&#039;s'|htmlspecialchars_decode( 0 )}
```

Output:

```
'Café & "Bar"'|htmlentities = Caf&eacute; &amp; &quot;Bar&quot;
'Caf&eacute; &amp; &quot;Bar&quot;'|html_entity_decode = Café & "Bar"
'&lt;b&gt; &amp;amp;'|htmlspecialchars_decode = <b> &amp;
'it&#039;s'|htmlspecialchars_decode( 0 ) = it&#039;s
```

sprintf and vsprintf

sprintf

The piped input is the **format**; every parameter is one argument. Internally it calls `vsprintf( format, parameters )`, so all printf conversions work, including argument numbers (`%2$s`). Returns a string. A format with more conversions than parameters makes PHP throw an `ArgumentCountError`, so keep them in step.

Parameter	Type	Required	Default
arguments	mixed	no	none

Generated reference: sprintf.

vsprintf

The same with the arguments in one array (a non-array parameter is wrapped in one). Without the array it logs "Missing required parameter: args_array".

Parameter	Type	Required	Default
arguments	array	yes	none

Generated reference: vsprintf.

The first line uses the Recipes category of a development installation (node 144):

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 144 ) )}
{'%s has %d children'|sprintf( $node.name, $node.children_count )}
{'%05.1f%%'|sprintf( 42.25 )}
{'%2$s, %1$s'|sprintf( 'first', 'second' )}
{'%s and %s'|vsprintf( array( 'fit', 'healthy' ) )}
```

Output:

```
Recipes has 18 children
042.2%
second, first
fit and healthy
```

`sprintf` does not escape: wash a value that comes from content, either before passing it or the whole result.

number_format

number_format

PHP `number_format()` on the input cast to a float. The parameter mapping has one quirk: with **two** parameters, the thousands separator is fixed to `,`, so `number_format( 2, ',' )` puts commas on both sides. Give all three to choose both separators.

Parameter	Type	Required	Default
decimals	integer	no	0
decimal separator	string	no	.
thousands separator	string	no	,

Returns a string. For numbers in the visitor's locale use `l10n( 'number' )` instead (Locale operators). Generated reference: number_format.

Template

```
1234567.891|number_format = {1234567.891|number_format}
1234567.891|number_format( 2 ) = {1234567.891|number_format( 2 )}
1234567.891|number_format( 2, ',' ) = {1234567.891|number_format( 2, ',' )}
1234567.891|number_format( 2, ',', '.' ) = {1234567.891|number_format( 2, ',', '.' )}
1234567.891|number_format( 2, '.', ' ' ) = {1234567.891|number_format( 2, '.', ' ' )}
```

Output:

```
1234567.891|number_format = 1,234,568
1234567.891|number_format( 2 ) = 1,234,567.89
1234567.891|number_format( 2, ',' ) = 1,234,567,89
1234567.891|number_format( 2, ',', '.' ) = 1.234.567,89
1234567.891|number_format( 2, '.', ' ' ) = 1 234 567.89
```

Related pages

- PHP string functions: search and compare
- PHP string functions: transform
- String operators: `wash`
- Locale operators: numbers, currency and dates for a locale
- 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.