

---

# Twig-extensions Documentation

*Release latest*

May 24, 2017



---

# Contents

---

<b>1</b>	<b>The Text Extension</b>	<b>1</b>
1.1	Installation . . . . .	1
1.2	Wrapping Words . . . . .	1
1.3	Truncating Text . . . . .	2
<b>2</b>	<b>The i18n Extension</b>	<b>3</b>
2.1	Configuration . . . . .	3
2.2	Usage . . . . .	3
2.3	Complex Translations within an Expression or Tag . . . . .	5
2.4	Extracting Template Strings . . . . .	5
<b>3</b>	<b>The Intl Extension</b>	<b>7</b>
3.1	Installation . . . . .	7
3.2	localizeddate . . . . .	7
3.3	localizednumber . . . . .	8
3.4	localizedcurrency . . . . .	9
<b>4</b>	<b>The Array Extension</b>	<b>11</b>
4.1	Installation . . . . .	11
<b>5</b>	<b>The Date Extension</b>	<b>13</b>
5.1	time_diff . . . . .	13
<b>6</b>	<b>Installation</b>	<b>15</b>



---

## The Text Extension

---

The Text extension provides the following filters:

- truncate
- wordwrap

### Installation

First, *install the Extensions library*. Next, add the extension to Twig:

```
$twig->addExtension(new Twig_Extensions_Extension_Text());
```

### Wrapping Words

Use the `wordwrap` filter to split your text in lines with equal length.

```
{{ "Lorem ipsum dolor sit amet, consectetur adipiscing"|wordwrap(10) }}
```

This example would print:

```
Lorem ipsu  
m dolor si  
t amet, co  
nsectetur  
adipiscing
```

The default separator is “\n”, but you can easily change that by providing one:

```
{{ "Lorem ipsum dolor sit amet, consectetur adipiscing"|wordwrap(10, "zz\n") }}
```

This would result in:

```
Lorem ipsuzz  
m dolor sizz  
t amet, cozz  
nsectetur zz  
adipiscing
```

## Truncating Text

Use the `truncate` filter to cut off a string after limit is reached

```
{{ "Hello World!"|truncate(5) }}
```

The example would output `Hello...`, as `...` is the default separator.

You can also tell `truncate` to preserve whole words by setting the second parameter to `true`. If the last Word is on the the separator, `truncate` will print out the whole Word.

```
{{ "Hello World!"|truncate(7, true) }}
```

Here `Hello World!` would be printed.

If you want to change the separator, just set the third parameter to your desired separator.

```
{{ "Hello World!"|truncate(7, false, "??") }}
```

This example would print `Hello W??`.

### Configuration

The `i18n` extension adds `gettext` support to Twig. It defines one tag, `trans`.

To use it, first, *install the Extensions library*.

You need to register this extension before using the `trans` block:

```
$twig->addExtension(new Twig_Extensions_Extension_I18n());
```

Note that you must configure the `gettext` extension before rendering any internationalized template. Here is a simple configuration example from the PHP [documentation](#):

```
// Set language to French
putenv('LC_ALL=fr_FR');
setlocale(LC_ALL, 'fr_FR');

// Specify the location of the translation tables
bindtextdomain('myAppPhp', 'includes/locale');
bind_textdomain_codeset('myAppPhp', 'UTF-8');

// Choose domain
textdomain('myAppPhp');
```

**Caution:** The `i18n` extension only works if the PHP `gettext` extension is enabled.

### Usage

Use the `trans` block to mark parts in the template as translatable:

```
{% trans "Hello World!" %}

{% trans string_var %}

{% trans %}
    Hello World!
{% endtrans %}
```

In a translatable string, you can embed variables:

```
{% trans %}
    Hello {{ name }}!
{% endtrans %}
```

During the gettext lookup these placeholders are converted. `{{ name }}` becomes `%name%` so the gettext msgid for this string would be `Hello %name%!`.

---

**Note:** `{% trans "Hello {{ name }}!" %}` is not a valid statement.

---

If you need to apply filters to the variables, you first need to assign the result to a variable:

```
{% set name = name|capitalize %}

{% trans %}
    Hello {{ name }}!
{% endtrans %}
```

To pluralize a translatable string, use the plural block:

```
{% trans %}
    Hey {{ name }}, I have one apple.
{% plural apple_count %}
    Hey {{ name }}, I have {{ count }} apples.
{% endtrans %}
```

The plural tag should provide the count used to select the right string. Within the translatable string, the special count variable always contain the count value (here the value of `apple_count`).

To add notes for translators, use the notes block:

```
{% trans %}
    Hey {{ name }}, I have one apple.
{% plural apple_count %}
    Hey {{ name }}, I have {{ count }} apples.
{% notes %}
    This is shown in the user menu. This string should be shorter than 30 chars
{% endtrans %}
```

You can use notes with or without plural. Once you get your templates compiled you should configure the gettext parser to get something like this: `xgettext --add-comments=notes`

Within an expression or in a tag, you can use the `trans` filter to translate simple strings or variables:

```
{{ var|default(default_value|trans) }}
```



## Complex Translations within an Expression or Tag

Translations can be done with both the `trans` tag and the `trans` filter. The filter is less powerful as it only works for simple variables or strings. For more complex scenario, like pluralization, you can use a two-step strategy:

```
{# assign the translation to a temporary variable #}
{% set default_value %}
    {% trans %}
        Hey {{ name }}, I have one apple.
    {% plural apple_count %}
        Hey {{ name }}, I have {{ count }} apples.
    {% endtrans %}
{% endset %}

{# use the temporary variable within an expression #}
{{ var|default(default_value|trans) }}
```

## Extracting Template Strings

If you use the Twig I18n extension, you will probably need to extract the template strings at some point. Unfortunately, the `xgettext` utility does not understand Twig templates natively. But there is a simple workaround: as Twig converts templates to PHP files, you can use `xgettext` on the template cache instead.

Create a script that forces the generation of the cache for all your templates. Here is a simple example to get you started:

```
$tplDir = dirname(__FILE__).'/templates';
$tmpDir = '/tmp/cache/';
$loader = new Twig_Loader_FileSystem($tplDir);

// force auto-reload to always have the latest version of the template
$twig = new Twig_Environment($loader, array(
    'cache' => $tmpDir,
    'auto_reload' => true
));
$twig->addExtension(new Twig_Extensions_Extension_I18n());
// configure Twig the way you want

// iterate over all your templates
foreach (new RecursiveIteratorIterator(new RecursiveDirectoryIterator($tplDir),
↳RecursiveIteratorIterator::LEAVES_ONLY) as $file)
{
    // force compilation
    if ($file->isFile()) {
        $twig->loadTemplate(str_replace($tplDir.'/', '', $file));
    }
}
```

Use the standard `xgettext` utility as you would have done with plain PHP code:

```
xgettext --default-domain=messages -p ./locale --from-code=UTF-8 -n --omit-header -L_
↳PHP /tmp/cache/*.php
```

Another workaround is to use [Twig Gettext Extractor](#) and extract the template strings right from Poedit.



The *Intl* extensions provides the `localizeddate`, `localizednumber` and `localizedcurrency` filters.

### Installation

First, *install the Extensions library*. Next, add the extension to Twig:

```
$twig->addExtension(new Twig_Extensions_Extension_Intl());
```

### localizeddate

Use the `localizeddate` filter to format dates into a localized string representing the date.

```
{{ post.published_at|localizeddate('medium', 'none', locale) }}
```

The `localizeddate` filter accepts strings (it must be in a format supported by the `strtotime` function), `DateTime` instances, or `Unix timestamps`.

---

**Note:** Internally, Twig uses the PHP `IntlDateFormatter::create()` function for the date.

---

### Arguments

- `date_format`: The date format. Choose one of these formats:
  - `'none'`: `IntlDateFormatter::NONE`
  - `'short'`: `IntlDateFormatter::SHORT`
  - `'medium'`: `IntlDateFormatter::MEDIUM`

- ‘long’: `IntlDateFormatter::LONG`
- ‘full’: `IntlDateFormatter::FULL`
- `time_format`: The time format. Same formats possible as above.
- `locale`: The locale used for the format. If `NULL` is given, Twig will use `Locale::getDefault()`
- `timezone`: The date timezone
- `format`: Optional pattern to use when formatting or parsing. Possible patterns are documented in the [ICU user guide](#).
- `calendar`: Calendar to use for formatting. The default value is ‘gregorian’, which corresponds to `IntlDateFormatter::GREGORIAN`. Choose one of these formats:
  - ‘gregorian’: `IntlDateFormatter::GREGORIAN`
  - ‘traditional’: `IntlDateFormatter::TRADITIONAL`

For the following calendars should use ‘traditional’:

- Japanese
- Buddhist
- Chinese
- Persian
- Indian
- Islamic
- Hebrew
- Indian
- Coptic
- Ethiopic

Also for non-Gregorian calendars need to be specified in `locale`. Examples might include `locale="fa_IR@calendar=PERSIAN"`.

## localizednumber

Use the `localizednumber` filter to format numbers into a localized string representing the number.

```
{{ product.quantity|localizednumber }}
```

---

**Note:** Internally, Twig uses the PHP `NumberFormatter::create()` function for the number.

---

## Arguments

- `style`: Optional date format (default: ‘decimal’). Choose one of these formats:
  - ‘decimal’: `NumberFormatter::DECIMAL`
  - ‘currency’: `NumberFormatter::CURRENCY`

- 'percent': `NumberFormatter::PERCENT`
- 'scientific': `NumberFormatter::SCIENTIFIC`
- 'spellout': `NumberFormatter::SPELLOUT`
- 'ordinal': `NumberFormatter::ORDINAL`
- 'duration': `NumberFormatter::DURATION`
- `type`: Optional formatting type to use (default: 'default'). Choose one of these types:
  - 'default': `NumberFormatter::TYPE_DEFAULT`
  - 'int32': `NumberFormatter::TYPE_INT32`
  - 'int64': `NumberFormatter::TYPE_INT64`
  - 'double': `NumberFormatter::TYPE_DOUBLE`
  - 'currency': `NumberFormatter::TYPE_CURRENCY`
- `locale`: The locale used for the format. If `NULL` is given, Twig will use `Locale::getDefault()`

## localizedcurrency

Use the `localizedcurrency` filter to format a currency value into a localized string.

```
{{ product.price|localizedcurrency('EUR') }}
```

---

**Note:** Internally, Twig uses the PHP `NumberFormatter::create()` function for the number.

---

## Arguments

- `currency`: The 3-letter ISO 4217 currency code indicating the currency to use.
- `locale`: The locale used for the format. If `NULL` is given, Twig will use `Locale::getDefault()`



---

## The Array Extension

---

The Array extensions provides the following filters:

- `shuffle`

### Installation

First, *install the Extensions library*. Next, add the extension to Twig:

```
$twig->addExtension(new Twig_Extensions_Extension_Array());
```





---

## The Date Extension

---

The *Date* extension provides the `time_diff` filter.

You need to register this extension before using the `time_diff` filter:

```
$twig->addExtension(new Twig_Extensions_Extension_Date());
```

### `time_diff`

Use the `time_diff` filter to render the difference between a date and now.

```
{{ post.published_at|time_diff }}
```

The example above will output a string like `4 seconds ago` or `in 1 month`, depending on the filtered date.

---

**Note:** Internally, Twig uses the PHP `DateTime::diff()` method for calculating the difference between dates, this means that PHP 5.3+ is required.

---

### Arguments

- `date`: The date for calculate the difference from now. Can be a string or a `DateTime` instance.
- `now`: The date that should be used as now. Can be a string or a `DateTime` instance. Do not set this argument to use current date.

### Translation

To get a translatable output, give a `Symfony\Component\Translation\TranslatorInterface` as constructor argument. The returned string is formatted as `diff.ago.XXX` or `diff.in.XXX` where `XXX` can be any valid unit: second, minute, hour, day, month, year.

The Twig Extensions is a library that provides several useful extensions for Twig. You can find it's code at [GitHub.com/twigphp/Twig-extensions](https://github.com/twigphp/Twig-extensions).

This library can be installed via Composer running the following from the command line:

```
composer require twig/extensions
```

- *Text*: Provides useful filters for text manipulation;
- *Intl*: Adds internationalization support via the `gettext` library;
- *Intl*: Adds a filter for localization of `DateTime` objects, numbers and currency;
- *Array*: Provides useful filters for array manipulation;
- *Date*: Adds a filter for rendering the difference between dates.