FREE eBook

LEARNING

twig

Free unaffiliated eBook created from
Stack Overflow contributors.

#twig

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: twig

It is an unofficial and free twig ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official twig.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with twig

Remarks

This section provides an overview of what twig is, and why a developer might want to use it.

It should also mention any large subjects within twig, and link out to the related topics. Since the Documentation for twig is new, you may need to create initial versions of those related topics.

Examples

Basic API Usage

It can also be installed by downloading the source code and placing it in a directory of your project. However there are many benefits to using composer.

```
require '/path/to/lib/Twig/Autoloader.php';
Twig_Autoloader::register();

$loader = new Twig_Loader_Filesystem('/path/to/templates');

$options = array(
    'strict_variables' => false,
    'debug' => false,
    'cache'=> false
);

$twig = new Twig_Environment($loader, $options);
```

When creating a new `Twig_Environment` instance, you can pass an array of options as the constructor's second argument. Here is a list of the available options:

- debug (*boolean*, default `false`)

  When set to true, the generated templates have a `__toString()` method that you can use to display the generated nodes.

- charset (*string*, default `utf-8`)

  The charset used by the templates.

- base_template_class (*string*, default `Twig_Template`)

  The base template class to use for generated templates.

- cache (*string or `false`*, default `false`)

  An absolute path where to store the compiled templates, or false to disable caching (which is the default).

- auto_reload (*boolean*, default inherited from *debug*)

  When developing with Twig, it's useful to recompile the template whenever the source code changes. If you don't provide a value for the auto_reload option, it will be determined automatically based on the debug value.

- strict_variables (*boolean*, default `false`)

  If set to false, Twig will silently ignore invalid variables (variables and or attributes/methods that do not exist) and replace them with a null value. When set to true, Twig throws an exception instead.

- autoescape (*string or boolean*, default`true`)

  If set to true, HTML auto-escaping will be enabled by default for all templates.

  As of Twig 1.8, you can set the escaping strategy to use (html, js, false to disable).

  As of Twig 1.9, you can set the escaping strategy to use (css, url, html_attr, or a PHP callback that takes the template "filename" and must return the escaping strategy to use -- the callback cannot be a function name to avoid collision with built-in escaping strategies).

  As of Twig 1.17, the filename escaping strategy determines the escaping strategy to use for a template based on the template filename extension (this strategy does not incur any overhead at runtime as auto-escaping is done at compilation time.)

- optimizations (*integer*, default `-1`)

  A flag that indicates which optimizations to apply:

  ```
  set to -1 to enabled all optimalizations
  set o 0 to disable all optimalitazations
  ```

Official Twig Installation Guide

A Twig PHP extension (written in C) can also be compiled and installed, and the PHP package will automatically take advantage of that for optimizing some common routines.

What is Twig?

Twig is a templating language that compiles to optimized PHP code. It is primarily used for outputting HTML, but can also be used to output any other text-based format. It is a standalone component that can be easily integrated into any PHP project.

It provides many excellent features:

- Autoescaping for HTML (helps to prevent XSS)
- Syntax designed with templating in mind (based on Django templates)
- Template inheritance
- Macros

Official Twig Templating Documentation

Example of Twig's syntax:

```
{% extends "base.html" %}

{% block sidebar %}
  {{ parent() }}
  <span>Sidebar content specific to this page</span>
{% endblock sidebar %}

{% block body %}
  <p>Select an item:</p>
  <ul>
    {% for item in list %}
      <li><a href="/items/{{ item.id }}">{{ item.name }}</a>
    {% else %}
      <li>No items yet.
    {% endfor %}
  </ul>
{% endblock body %}
```

Introduction

If you have any exposure to other text-based template languages, such as Smarty, Django, or Jinja, you should feel right at home with Twig. It's both ***designer and developer friendly*** by sticking to PHP's principles and adding functionality useful for templating environments.

The key-features are...

- Fast: Twig compiles templates down to plain optimized PHP code. The overhead compared to regular PHP code was reduced to the very minimum.
- Secure: Twig has a **sandbox mode** to evaluate untrusted template code. This allows Twig to be used as a template language for applications where users may modify the template design.
- Flexible: Twig is powered by a flexible **lexer** and **parser**. This allows the developer to define their own **custom tags** and **filters**, and to create their own DSL.

Twig is used by many Open-Source projects like Symfony, Drupal, eZPublish and many frameworks have support for it as well like Slim, Yii, Laravel, Codeigniter,silex and Kohana — just to name a few.

Installation

The recommended way to install Twig is via Composer:

For **php 5.x** users

```
composer require "twig/twig:~1.0"
```

For **php 7.x** users

```
composer require "twig/twig:~2.0"
```

Read Getting started with twig online: https://riptutorial.com/twig/topic/1100/getting-started-with-twig

Chapter 2: Advanced usage

Remarks

Please note that the Twig extension is not compatible with PHP7 and although there is a pull request to resolve this situation it was not yet made part of version 1.x. With the most recent Twig 2.x release the C extension was removed but might be part of 2.1. Some benchmarks reveal minimal improvements with the C extension under PHP7.

Examples

Building the C Extension

The C extension is an optional feature of Twig that offers some performance improvements of template rendering. The source code for the extension is located in the Twig source code directory at `ext/twig`. It compiles like any other PHP extentension:

```
cd ext/twig
phpize
./configure
make
make install
```

Enable the extension in the `php.ini` file. Once the extension is installed, Twig will automatically detect and use it at runtime.

Read Advanced usage online: https://riptutorial.com/twig/topic/8696/advanced-usage

Chapter 3: Basic template syntax

Introduction

Explanation of basic template syntax constructs

Examples

Accessing variables

In Twig templates variables can be accessed using double curly braces notation `{{ variableName }}`.

Basic example of greeting user

```
<!DOCTYPE html>
<html>
  <body>
    <span>Hello {{ name }}</span>
  </body>
</html>
```

Accessing array elements

Twig as a parameter can receive array. To access a specific element of array you can use regular php array access bracket notation `{{ array[key] }}`.

Previous example modified to use array as a parameter

```
<!DOCTYPE html>
<html>
  <body>
    <span>Hello {{ user['name'] }}</span>
  </body>
</html>
```

Accessing object properties

Objects also can be passed as a parameter to template. 'Dot' (.) notation is used to access specific object properties `{{ object.propertyName }}`.

Same example with object as a parameter

```
<!DOCTYPE html>
<html>
  <body>
    <span>Hello {{ user.name }}</span>
```

```
  </body>
</html>
```

Filters

Variables can be modified using filters. To apply filter to variable follow variable name with pipe `|` and filter name:

```
{{ variable|filterName }}
```

For example to display variable value in uppercase use construct.

```
{{ variable|upper }}
```

Filters can be parametrized. Filter parameters are passed inside parentheses as a comma(,) separated list :

```
{{ variable|filterName(param1, param2, ...) }}
```

To round number to given precision we can use filter `round`, it accepts up to 2 parameters. First one specifies precision (default: 0), second one rounding method (default: common).

To round number to 1 decimal place using common method you can use `{{ number|round(1, 'common') }}` or `{{ number|round(1) }}` as common is default method.

Filters can also be used on embedded objects & array variables:

```
{{ array['key'] | upper }} {{ object.text | upper }}
```

Filters can also be concatenated:

```
{% set array = "3,1,2"|split(',') %}

{{ array | sort | first }}
```

List of basic filters is available here

Conditional blocks

Parts of template can be displayed conditionally. `If` statement is used for this purpose. It's similar to `if` statement in programing languages. Contents of block are executed/displayed if an expression evaluates to `true`.

```
{% if enabled == false %}
    Disabled
{% endif %}
```

Disabled will be displayed only when `enabled` will be equal `false`.

Multiple branches can be created using `elseif` and `else`.

```
{% if temperature < 10 %}
    It's cold
{% elseif temperature < 18 %}
    It's chilly
{% elseif temperature < 24 %}
    It's warm
{% elseif temperature < 32 %}
    It's hot
{% else %}
    It's very hot
{% endif %}
```

For Loop

For loops can be really useful in TWIG, allowing the creation of data-dynamic web pages.

Say we create a simple array of numbers:

```
{% set array = "3,1,2" %}
```

We can then iterate over the array and print out whatever we want. Any data within the array block will be outputted in relation to the amount of data within the array. This example would print out three h1 elements with the array data concatenated.

```
{% for current in array %}
    <h1>This is number {{ current }} in the array </h1>
{% endear %}
```

Note that `{{ current }}` was used to access the data and not a version of `array`

See this working: https://twigfiddle.com/mxwkea/2

A further example of this could be using objects. For example, say we have an Entity object with the field 'name' along with its relevant *getters and setters*. If a number of these entities are also stored within the Array, they can be accessed like any other Objects within TWIG:

```
{% for currentObject in ArrayOfObjects %}
    {{ currentObject.name }}
{% endfor %}
```

See this working with JSON data: https://twigfiddle.com/mxwkea

Ternary Operator (Shorthand If-Then-Else) & Null-Coalescing Operator

The ternary operator (`?:`)

Support for the extended ternary operator was added in **Twig 1.12.0**.

```
{{ foo ? 'yes' : 'no' }}
```

Evaluates:

> if `foo` echo `yes` else echo `no`

```
{{ foo ?: 'no' }}
```

or

```
{{ foo ? foo : 'no' }}
```

Evaluates:

> if `foo` echo it, else echo `no`

```
{{ foo ? 'yes' }}
```

or

```
{{ foo ? 'yes' : '' }}
```

Evaluates:

> if `foo` echo `yes` else echo nothing

The null-coalescing operator (`??:`)

```
{{ foo ?? 'no' }}
```

Evaluates:

> Returns the value of `foo` if it **is defined** and **not null**, `no` otherwise

Whitespace handling

To remove whitespace (spaces, tabs, newlines…) between HTML tags use `spaceless` tag:

```
{% spaceless %}

    <div>
        <span>foo bar </span>
    </div>
{% endspaceless %}
{# produces output <div><strong>foo bar </strong></div> #}
```

If you need to remove whitespace on a per tag level, use *whitespace control modifier* i.e. hyphen (-). Using it, you can trim leading and or trailing whitespace:

```
{% set value = 'foo bar' %}

<span>  {{- value }}  </span>
{# produces '<span>foo bar  </span>' #}

<span>  {{ value -}}  </span>
{# produces '<span>  foo bar</span>' #}

<span>  {{- value -}}  </span>
{# produces '<span>foo bar</span>' #}

<span {%- if true  %} class="foo"{% endif %}>
{# produces '<span class="foo">' #}

<span {%- if false %} class="foo"{% endif %}>
{# produces '<span>' #}
```

Read Basic template syntax online: https://riptutorial.com/twig/topic/8697/basic-template-syntax

Chapter 4: Extending twig

Remarks

Twig already has some built-in filters and functions, but what if the built-in features are lacking or you have to access some default `PHP` functions in a template

Examples

Adding custom filters/functions

Here are some example on how to add new filters/functions to `twig`,
the synax for adding `Twig_Function`s are the same as the `Twig_Filter` ones, just change the keywords accordingly

```
<?php
   $twig = new Twig_Environment($loader);

   /* You can chain a global function */
   $twig->addFilter(new Twig_SimpleFilter('floor', 'floor'));

   /* You can specify a custom function */
   $twig->addFilter(new Twig_SimpleFilter('money', function($value, $currency, $prefix =
false, $decimals = 2, $dec_point = "." , $thousands_sep = ",") {
       $value = number_format($value, $decimals, $dec_point, $thousands_sep);
       if ($prefix) return $currency.' '.$value;
       return $value.' '.$prefix;
   });

   /* You can chain an object's method */
   $twig->addFilter(new Twig_SimpleFilter('foo_bar', array($foo, 'bar')));
```

Creating a Twig_Extension

You can group all your custom functions/filters/tests/... inside a custom `Twig_Extension` class:

ProjectTwigExtension

```
class ProjectTwigExtension extends Twig_Extension {

    public function getFunctions() {
        return array(
            new Twig_SimpleFunction('twig_function_name', array($this,
'getTwigFunctionName')),
            new Twig_SimpleFunction('twig_function_foo', array($this, 'getTwigFunctionFoo')),

        );



    }
```

```
    public function getFilters() {
        return array(
            new Twig_SimpleFilter('twig_filter_name' , array($this, 'getTwigFilterName')),
            new Twig_SimpleFilter('twig_filter_foo' , array($this, 'getTwigFilterFoo')),
        );
    }

    public function getName() {
        return 'ProjectTwigExtension';
    }
}
```

Register extension in twig

```
$twig = new Twig_Environment($loader);
$twig->addExtension(new ProjectTwigExtension());
```

More options can be found on the official docs

Simple Date of Birth to age filter

How to ...

1 - use twig extension class that extends

```
use \Twig_Extension

class dobToAge extends \Twig_Extension {
```

2 - Add the appropriate filter by overriding getFilters() method

```
 public function getFilters() {
        return array(
            'age' => new \Twig_Filter_Method($this, 'getAge'),
        );
 }
```

3 - Add some logic to get the age of a given Date of Birth

```
 public function getAge($date)
     {
        if (!$date instanceof \DateTime) {
        // turn $date into a valid \DateTime object or let return
        return null;
         }

     $referenceDate = date('01-01-Y');
     $referenceDateTimeObject = new \DateTime($referenceDate);
     $diff = $referenceDateTimeObject->diff($date);
     return $diff->y;
    }
}
```

Then, call your filter as follow,

```
{{ yourDateOfBirthInstance | age }}
```

Read Extending twig online: https://riptutorial.com/twig/topic/4923/extending-twig

Chapter 5: Template inheritance

Examples

Using a base template

base.twig.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>{{ title | default('Hello World') }}</title>
        <link rel="stylesheet" type="text/css" href="theme.css">
        {% block css %}
        {% endblock %}
    </head>
    <body>
    {% block body %}
        <nav>
        {% block navigation %}
            <a href="#">Link</a>
            <a href="#">Link</a>
            <a href="#">Link</a>
        {% endblock navigation %}
        </nav>
        <section id="container">
            <section id="content">
            {% block content %}
            <p>
            Lorem ipsum dolor sit amet.
            </p>
        </section>
        {% endblock content %}
    </section>
    {% endblock body %}
    </body>
</html>
```

page.twig.html

```
{% extends base.twig.html %}
{% block navigation %}
<a href="page2.html">Page 2</a>
<a href="page3.html">Page 3</a>
<a href="page4.html">Page 4</a>
{% endblock %}
{% block content %}
This is my first page
{% endblock content %}
```

Change the content of an included template

article.twig.html

```
<article>
    <h1>{{ article.title }}</h1>
    {% block content %}
    <p>{{ article.content }}</p>
    {% endblock %}
</article>
```

articles.twig.html

```
{# use default template for article #}
{% for article in articles %}
    {% include "article.twig.html" %}
{% endfor %}
```

```
{# use custom template for article #}
{% for article in articles %}
    {% embed "article.twig.html" %}
        {% block content %}
            <img src="{{ article.image }}" alt="{{ article.title }}" />
            {{ parent() }}
        {% endblock %}
    {% endembed %}
{% endfor %}
```

Variable inheritance

parent.html.twig

```
<!DOCTYPE html>
<html>
    <head>
        <title>{{ title_variable | default('Normal Page') }}</title>
    </head>
    <body>
         <h1>This is the {{ title_variable }} Page</h1>
    </body>
</html>
```

---

child.html.twig

```
{% extends "parent.html.twig" %}
{% set title_variable = "Child" %}
```

Comparison of Include, Extends, Use, Macro, Embed

There are various types of inheritance and code reuse in Twig:

Include

The main goal is **code reuse**. Consider using `header.html.twig` & `footer.html.twig` inside `base.html.twig` as an example.

header.html.twig

```
<nav>
   <div>Homepage</div>
   <div>About</div>
</nav>
```

---

base.html.twig

```
{% include 'header.html.twig' %}
<main>{% block main %}{% endblock %}</main>
```

Extends

The main goal is **vertical inheritance**. Consider extending `base.html.twig` inside `homepage.html.twig` and `about.html.twig` as an example.

base.html.twig

```
{% include 'header.html.twig' %}
<main>{% block main %}{% endblock %}</main>
```

---

homepage.html.twig

```
{% extends 'base.html.twig' %}

{% block main %}
<p>You are at the homepage</p>
{% endblock %}
```

---

about.html.twig

```
{% extends 'base.html.twig' %}

{% block main %}
<p>You are at the about page</p>
{% endblock %}
```

Use

The main goal is **horizontal reuse**. Consider using `sidebar.product.html.twig` inside `single.product.html.twig` (extends `product.layout.html.twig`) and `single.service.html.twig` (extends 'service.layout.html.page') pages. (it's like macros, but for blocks)

sidebar.html.twig

```
<aside>{% block sidebar %}{% endblock %}</aside>
```

---

single.product.html.twig

```
{% extends 'product.layout.html.twig' %}

{% use 'sidebar.html.twig' %}
{% block main %}
<p>You are at the product page for product number 123</p>
{% endblock %}
```

---

single.service.html.twig

```
{% extends 'service.layout.html.twig' %}

{% use 'sidebar.html.twig' %}
{% block main %}
<p>You are at the service page for service number 456</p>
{% endblock %}
```

Macro

The main goal is **having reusable markup across many templates with variables**. Consider a function which gets some variables and outputs some markup.

form.html.twig

{% macro input(name, value, type) %} <input type="{{ type|default('text') }}" name="{{ name }}" value="{{ value|e }}" }}" /> {% endmacro %}

---

profile.service.html.twig

```
{% import "forms.html.twig" as forms %}

<div>{{ forms.input('username') }}</div>
```

Embed

The main goal is **block overriding**. It has functionality of both `Use` & `Include` together. Consider embedding `pagination.html.twig` in `product.table.html.twig` & `service.table.html.twig`.

pagination.html.twig

```
<div>
    <div>{% block first %}{% endblock %}</div>
    {% for i in (min + 1)..(max - 1) %}
        <div>{{ i }}</div>
    {% endfor %}
    <div>{% block last %}{% endblock %}</div>
</div>
```

---

product.table.html.twig

```
{% set min, max = 1, products.itemPerPage %}

{% embed 'pagination.html.twig' %}
    {% block first %}First Product Page{% endblock %}
    {% block last %}Last Product Page{% endblock %}
{% endembed %}
```

---

service.table.html.twig

```
{% set min, max = 1, services.itemPerPage %}

{% embed 'pagination.html.twig' %}
    {% block first %}First Service Page{% endblock %}
    {% block last %}Last Service Page{% endblock %}
{% endembed %}
```

Please note that embedded file (`pagination.html.twig` here) has access to the current context (`min`, `max` variables here). Also you may pass extra variables to the embedded file:

pagination.html.twig

```
<p>{{ count }} items</p>
<div>
    <div>{% block first %}{% endblock %}</div>
    {% for i in (min + 1)..(max - 1) %}
        <div>{{ i }}</div>
    {% endfor %}
    <div>{% block last %}{% endblock %}</div>
</div>
```

---

product.table.html.twig

```
{% set min, max = 1, products|length %}

{% embed 'pagination.html.twig' with {'count': products|length } %}
    {% block first %}First Product Page{% endblock %}
    {% block last %}Last Product Page{% endblock %}
{% endembed %}
```

Read Template inheritance online: https://riptutorial.com/twig/topic/2922/template-inheritance

Credits

S. No	Chapters	Contributors
1	Getting started with twig	4444, Community, DarkBee, Icode4food, insertusernamehere, mleko, Object, Paulpro, user1950896
2	Advanced usage	Icode4food, mleko, Ricardo Velhote
3	Basic template syntax	jkucharovic, mleko, PastyAndPeas, Trix
4	Extending twig	DarkBee, Object
5	Template inheritance	DarkBee, Ḟ̄ḁḿíṅḡ ỏḿƀíé, nicolallias, PastyAndPeas, Trix