SFW Security Monitoring

Twig | The flexible, fast, and secure template engine for PHP

ABOUT DOCUMENTATION DEVELOPMENT

You are reading the documentation for Twig 2.x. Switch to the documentation for Twig 1.x.

extends¶

The `extends` tag can be used to extend a template from another one.

> Like PHP, Twig does not support multiple inheritance. So you can only have one extends tag called per rendering. However, Twig supports horizontal reuse.

Let's define a base template, `base.html`, which defines a simple HTML skeleton document:

```
<!DOCTYPE html>
<html>
    <head>
        {% block head %}
            <link rel="stylesheet" href="style.css" />
            <title>{% block title %}{% endblock %} - My Webpage</title>
        {% endblock %}
    </head>
    <body>
        <div id="content">{% block content %}{% endblock %}</div>
        <div id="footer">
            {% block footer %}
                &copy; Copyright 2011 by <a href="http://domain.invalid/">
            {% endblock %}
        </div>
    </body>
</html>
```

In this example, the block tags define four blocks that child templates can fill in.

All the `block` tag does is to tell the template engine that a child template may override those portions of the template.

Questions & Feedback

Found a typo or an error?
Want to improve this document? Edit it.

Need support or have a technical question?
Ask support on Stack Overflow.

Child Template¶

A child template might look like this:

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

{% block title %}Index{% endblock %}
{% block head %}
    {{ parent() }}
    <style type="text/css">
```

```
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
    <h1>Index</h1>
    <p class="important">
        Welcome on my awesome homepage.
    </p>
{% endblock %}
```

The `extends` tag is the key here. It tells the template engine that this template "extends" another template. When the template system evaluates this template, first it locates the parent. The extends tag should be the first tag in the template.

Note that since the child template doesn't define the `footer` block, the value from the parent template is used instead.

You can't define multiple `block` tags with the same name in the same template. This limitation exists because a block tag works in "both" directions. That is, a block tag doesn't just provide a hole to fill - it also defines the content that fills the hole in the *parent*. If there were two similarly-named `block` tags in a template, that template's parent wouldn't know which one of the blocks' content to use.

If you want to print a block multiple times you can however use the `block` function:

```
<title>{% block title %}{% endblock %}</title>
<h1>{{ block('title') }}</h1>
{% block body %}{% endblock %}
```

Parent Blocks¶

It's possible to render the contents of the parent block by using the parent function. This gives back the results of the parent block:

```
{% block sidebar %}
    <h3>Table Of Contents</h3>
    ...
    {{ parent() }}
{% endblock %}
```

Named Block End-Tags¶

Twig allows you to put the name of the block after the end tag for better readability:

```
{% block sidebar %}
    {% block inner_sidebar %}
        ...
    {% endblock inner_sidebar %}
{% endblock sidebar %}
```

Of course, the name after the `endblock` word must match the block name.

Block Nesting and Scope¶

Blocks can be nested for more complex layouts. Per default, blocks have access to variables from outer scopes:

```
{% for item in seq %}
    <li>{% block loop_item %}{{ item }}{% endblock %}</li>
{% endfor %}
```

Block Shortcuts¶

For blocks with little content, it's possible to use a shortcut syntax. The following constructs do the same thing:

```
{% block title %}
    {{ page_title|title }}
{% endblock %}
```

```
{% block title page_title|title %}
```

Dynamic Inheritance¶

Twig supports dynamic inheritance by using a variable as the base template:

```
{% extends some_var %}
```

If the variable evaluates to a `\Twig\Template` or a `\Twig\TemplateWrapper` instance, Twig will use it as the parent template:

```
// {% extends layout %}

$layout = $twig->load('some_layout_template.twig');

$twig->display('template.twig', ['layout' => $layout]);
```

You can also provide a list of templates that are checked for existence. The first template that exists will be used as a parent:

```
{% extends ['layout.html', 'base_layout.html'] %}
```

Conditional Inheritance¶

As the template name for the parent can be any valid Twig expression, it's possible to make the inheritance mechanism conditional:

```
{% extends standalone ? "minimum.html" : "base.html" %}
```

In this example, the template will extend the "minimum.html" layout template if the `standalone` variable evaluates to `true`, and "base.html" otherwise.

How do blocks work?¶

A block provides a way to change how a certain part of a template is rendered but it does not interfere in any way with the logic around it.

Let's take the following example to illustrate how a block works and more importantly, how it does not work:

```
{# base.twig #}

{% for post in posts %}
    {% block post %}
        <h1>{{ post.title }}</h1>
        <p>{{ post.body }}</p>
    {% endblock %}
{% endfor %}
```

If you render this template, the result would be exactly the same with or without the `block` tag. The `block` inside the `for` loop is just a way to make it overridable by a child template:

```
{# child.twig #}

{% extends "base.twig" %}

{% block post %}
    <article>
        <header>{{ post.title }}</header>
        <section>{{ post.text }}</section>
    </article>
{% endblock %}
```

Now, when rendering the child template, the loop is going to use the block defined in the child template instead of the one defined in the base one; the executed template is then equivalent to the following one:

```
{% for post in posts %}
    <article>
        <header>{{ post.title }}</header>
        <section>{{ post.text }}</section>
    </article>
{% endfor %}
```

Let's take another example: a block included within an `if` statement:

```
{% if posts is empty %}
    {% block head %}
        {{ parent() }}

        <meta name="robots" content="noindex, follow">


    {% endblock head %}
{% endif %}
```

Contrary to what you might think, this template does not define a block conditionally; it just makes overridable by a child template the output of what will be rendered when the condition is `true`.

If you want the output to be displayed conditionally, use the following instead: