

# Twig

The flexible, fast, and secure  
template language for PHP

Fabien Potencier



# Fabien Potencier

- Serial entrepreneur and developer by passion
- Founder of Sensio
  - Web Agency (France and USA)
  - Since 1998
  - 70 people
  - Open-Source Specialists
  - Big corporate customers
  - Consulting, training, development, web design, ... and more
  - Sponsor of a lot of Open-Source projects like symfony and Doctrine



# Fabien Potencier

- Creator and lead developer of **symfony**
- and creator and lead developer of some more:
  - symfony components
  - Swift Mailer : Powerful component based mailing library for PHP
  - Twig : Flexible, fast, and secure template language for PHP
  - Pirum : Simple PEAR Channel Server Manager
  - Sismo : PHP continuous integration server
  - Lime : Easy to use unit testing library for PHP
  - Twitto : A web framework in a tweet
  - Twittee : A Dependency Injection Container in a tweet
  - Pimple : A small PHP 5.3 dependency injection container



# Fabien Potencier

- Read my technical blog: <http://fabien.potencier.org/>
- Follow me on Twitter: @fabpot
- Fork my code on Github: <http://github.com/fabpot/>



# TWIG

The flexible, fast, and secure  
template language for PHP

ABOUT

INSTALLATION

DOCUMENTATION

BLOG

DEVELOPMENT

## Twig is a modern template language for PHP

- **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 templating 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 its own custom tags and filters, and create its own DSL.

LEARN MORE →

INSTALL NOW ↓

<http://www.twig-project.org/>



# Twig history

I was looking for a good template engine

I don't like to reinvent the wheel

I started to look around for existing ones



# Twig history

But I found none that satisfied all *my personal* needs:

- Fast (templates should be compiled to PHP)
- Secure (flexible output escaping)
- Sandboxed (whitelist system)
- Not tied to only HTML templates (emails, ...)
- Syntax similar to Django & Jinja (but flexible, nothing hardcoded)
- Embeddable
- “Modern” features (such as template inheritance)
- Easily extensible (should be able to override everything)
- *Clean OO code* (“real” lexer, parser, and compiler, DI)



# Twig history

- ... except Twig, which was quite close and had a solid architecture
  - written by Armin Ronacher (of Jinja fame – a python template engine)
- ... but
  - not available as a standalone project (packaged with chypr, a blog platform)
  - never really maintained
- I stumbled upon it by chance



# Twig history

- I started hacking the code, found it immediately beautiful and flexible
- I wrote an email to Armin
- and I took over the maintenance of it
  - I'm the lead developer since October 1<sup>st</sup>
  - kept the source under a BSD license
  - created a dedicated website, mailing-list, wrote some documentation... on October 7<sup>th</sup>
  - announced Twig on my blog:
    - <http://fabien.potencier.org/article/34/templating-engines-in-php>
    - <http://fabien.potencier.org/article/35/templating-engines-in-php> follow-up



# Twig Installation

- Download it
  - `http://github.com/fabpot/Twig/tarball/master`
- Use the Subversion repository
  - `http://svn.twig-project.org/trunk/`
- Get it on Github.com
  - `http://github.com/fabpot/twig`
- Install it via PEAR
  - `pear channel-discover pear.twig-project.org`
  - `pear install twig/Twig-beta`



How does Twig looks like?



# Template side

```
{# Twig template #}
```

comment

```
{% for item in items %}
```

tag

```
* {{ item.value }}
```

variable

```
{% endfor %}
```

filter

```
Hello {{ name|title }}
```

filter with arguments

```
{{ items|join(', ') }}
```

# Template side

```
{{ item.value }}
```

Can be any of the following

```
$item['value']  
$item->value()  
$item->getValue()  
$item->value
```

mask the implementation



# PHP side

```
require_once '/path/to/Twig/Autoloader.php';  
Twig_Autoloader::register();  
  
$loader = new Twig_Loader_Filesystem(  
    '/path/to/templates', '/path/to/cache');  
$twig = new Twig_Environment($loader);  
$twig->addExtension(new Twig_Extension_Escaper());
```

loader

extension

```
$template = $twig->loadTemplate('index.twig');  
$params = array('name' => 'Fabien');  
  
$template->display($params);
```

# Compiled template

```
class __TwigTemplate_5f8fa007771745904051c1db27ffc4e0 extends Twig_Template
{
    public function display(array $context)
    {
        // line 1
        echo "\n\nHello ";

        // line 3
        echo twig_escape_filter($this->getEnvironment(),
            (isset($context['name']) ? $context['name'] : null));
        echo "\n\n";

        // ...
    }
}
```

compiled as a class

debug information

# Template inheritance

```
{# Template #}  
{% extends "layout.twig" %}
```

template extends another one

```
{% block title "Cool..." %}
```

template override  
parent blocks

```
{% block content %}  
  Hello {{ name }}  
  {% parent %}  
{% endblock %}
```

parent content

```
{# Layout #}  
<html>  
  <head>  
    <title>{% block title '' %}</title>  
  </head>  
  <body>  
    {% block content '' %}  
  </body>  
</html>
```

define blocks with default values

# Template inheritance

```
class __TwigTemplate_8a7c3cb36a4064e9c740c094a108a5a4 extends __TwigTemplate_c61404957758dfda283709e89376ab3e
{
    public function block_title($context)
    {
        // ...
    }

    public function block_content($context)
    {
        // ...
    }
}
```



```
class __TwigTemplate_c61404957758dfda283709e89376ab3e extends Twig_Template
{
    public function display(array $context)
    {
        echo "\n <html>\n     <head>\n         <title>";
        $this->block_title($context);
        echo "</title>\n     </head>\n     <body>\n         ";
        $this->block_content($context);
        echo "\n     </body>\n </html>";
    }

    public function block_title($context) {}

    public function block_content($context) {}
}
```

# Why Twig compiles templates to PHP classes?

- Deep template inheritance becomes trivial to implement as PHP does all the job for us
- Very fast in common cases
  - For better reusability, small templates are created and included often (think partials in symfony)

```
{% for item in items %}  
  * {% include "item_partial.twig" %}  
{% endfor %}
```

- As each template is a stateless instance of a class, inclusion is just a method call away (no PHP include involved)



# Macros

A macro is a reusable and configurable snippet of HTML

```
{# forms.twig #}
{% macro input(name, value, type, size) %}
    <input type="{{ type|default('text') }}"
        name="{{ name }}" value="{{ value|e }}"
        size="{{ size|default(20) }}" />
{% endmacro %}

{# In a template #}
{% import "forms.twig" as forms %}

{{ forms.input('username') }}
{{ forms.input('password', none, 'password') }}
```

# Expressions

For display logic (not for business logic of course)

```
{% if 1 < foo < 4 and (not bar or foobar) %}  
  {{ foo + 4 }}  
{% endif %}
```



# Extensibility



# Extensibility

- Everything in the core can be changed
  - Uses dependency injection
  - The syntax is not hardcoded
- Built-in extension mechanism for day-to-day enhancements
  - Adding tags
  - Adding filters
  - Adding node transformers
- Extensions are used for the core features



# Core extensions

- Everything is an extension in the core
  - `Twig_Extension_Core`: Provides the core features (tags and filters)
  - `Twig_Extension_Escaper`: Adds automatic output-escaping and the possibility to escape/unescape blocks of code
  - `Twig_Extension_Sandbox`: Adds a sandbox mode to the default Twig environment, making it safe to evaluate untrusted code



# Registering an extension

- The Core extension is automatically registered
- Core and user extensions are managed in the same way

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



# Output Escaping



# XSS protection

```
{# Manually #}
```

```
{{ post.author|escape }}
```

escape variable

```
{{ foo|escape('js') }}
```

escaping strategy

```
{# A whole block #}
```

```
{% autoescape on %}
```

escape all variables of the block

```
  {{ post.author }}
```

```
  {{ post.title|escape }}
```

won't be double-escaped

```
  {{ post.html|safe }}
```

```
{% endautoescape %}
```

marked this as safe

```
{# Automatic #}
```

```
{{ post.author }}
```

```
{{ post.html|safe }}
```

# XSS protection

- Everything is done during compilation (see node transformers later on)
- Nothing is decided on execution
- No overhead between escaping by hand and auto-escaping



# Sandbox



# When do you need a sandbox?

- When you need to evaluate non-trusted code; a user who can edit templates, typically a webmaster in a web backend
- Security is managed by a policy instance  
(`Twig_Sandbox_SecurityPolicy` by default)
- The code evaluated in a sandbox must respect the policy
- The default policy works as follows
  - Tags must be explicitly allowed
  - Filters must be explicitly allowed
  - Method calls must be explicitly allowed on a method basis



# How to enable the sandbox mode?

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

$tags = array('if');
$filters = array('upper');
$methods = array(
    'Article' => array('getTitle', 'getBody'),
);

$policy = new Twig_Sandbox_SecurityPolicy($tags, $filters, $methods);
$sandbox = new Twig_Extension_Sandbox($policy);

$twig->addExtension($sandbox);
```



# Using the sandbox

Enable it globally

```
$sandbox = new Twig_Extension_Sandbox($policy, true);
```

Or on a template basis

```
{% include "user.twig" sandboxed %}
```



# When to use Twig?

- When users you don't trust can edit some templates
  - a webmaster can edit the look and feel of some templates
- When you need to create a simple language for a domain specific problem
  - Templates of a blog, or of an ecommerce software
- Advantages of Twig in such cases
  - You can use as little or as much features as you want
  - Security
  - Sandbox
  - Possibility to create a DSL (a set of pre-defined tags, macros, and filters specific to the domain)



# DSL

- One of the goal of Twig is to provide the architecture that allows the developer to create its own language targeted at his specific domain
  - Configure the syntax (delimiters)
  - Override all built-in elements (tags, filters, ...)
  - Create your own tags, macros, and filters
- All without having to write complex code
  - The core classes (lexer, parser, and compiler) provides a simple API to let you do your job without understanding the nitty-gritty details of the internals



# Under the hood



# From a template to a PHP file

Request

Something decide which template to render (typically a controller)

Twig\_Loader

if already compiled in cache (on the filesystem)

Twig\_Lexer

Twig\_Parser

Twig\_Compiler

Template source (string)

Stream of tokens (Twig\_Token)

Tree of nodes (Twig\_Node)

Compiled template (PHP class)

happens once

even faster  
if APC is enabled

HTML



# The lexer

The lexer tokenizes the template source and returns a stream of tokens

```
$stream =  
    $twig->tokenize('Hello {{ name|title }}');  
echo $stream;
```

```
TEXT_TYPE(Hello )  
VAR_START_TYPE()  
NAME_TYPE(name)  
VAR_END_TYPE()  
EOF_TYPE()
```

# The parser

The parser converts the stream of tokens into a tree of nodes

```
$nodes= $twig->parse($stream);  
echo $nodes;
```

```
Twig_Node_Module(  
    Twig_Node_Text(Hello )  
    Twig_Node_Print(  
        Twig_Node_Expression_Name(name)  
    )  
)  
)
```

# The parser

```
// with automatic output escaping
Twig_Node_Module(
  Twig_Node_Text(Hello )
  Twig_Node_Print(
    Twig_Node_Expression_Filter(
      Twig_Node_Expression_Name(name)
      (escape())
    )
  )
)
```

added by manipulating  
the AST

# Node transformer classes

- Manipulate the AST (Abstract Syntax Tree), the tree of nodes, before compilation
- Each transformer can visit the tree and do something with each node
  - remove it
  - visit its children
  - change it by another node
  - wrap it (with a filter for instance)



# Node transformer classes

- Used by the core extensions
  - Escaper
    - adds the escape filter to `Twig_Node_Print` nodes
    - except when the node is already filtered with `escape` (to avoid double-escaping), or already filtered with `safe` (to avoid escaping safe HTML strings)
  - Sandbox
    - To collect calls to filters and tags in a template
  - Filter blocks
    - To apply a set of filters to a whole block



# The compiler

The compiler eventually converts the tree of nodes to PHP code

```
class __TwigTemplate_5f8fa007771745904051c1db27ffc4e0 extends Twig_Template
{
    public function display(array $context)
    {
        // line 1
        echo "Hello ";

        echo (isset($context['name']) ? $context['name'] : null);
    }
}
```



# Integration



# Integration with other frameworks

- symfony
  - sfTwigPlugin
- Zend Framework
  - [http://svn.twig-project.org/branches/zend\\_adapter/](http://svn.twig-project.org/branches/zend_adapter/)
- Yii
  - <http://www.yiiframework.com/extension/twig-view-renderer/>
- Kohana
  - <http://github.com/shadowhand/kohana-twig>
- ... and more to come



# Questions?



Sensio S.A.  
92-98, boulevard Victor Hugo  
92 115 Clichy Cedex  
FRANCE

Tél. : +33 1 40 99 80 80

Contact  
Fabien Potencier  
fabien.potencier at sensio.com

<http://www.sensiolabs.com/>  
<http://www.symfony-project.org/>  
<http://fabien.potencier.org/>

