SithTemplate

1.1a2

Generated by Doxygen 1.5.9

Sat Sep 5 21:15:43 2009

Contents

Chapter 1

SithTemplate - open-source template engine for PHP5

1.1 License

1.2 Introduction

SithTemplate is a flexible and extensible PHP templating engine, inspired primarily by Django.

This is documentation for 1.1 line of the library, which obsoletes older 0.1-1.0 line.

1.2.1 Why should I try?

SithTemplate has number of useful features, so I think you should at least try it, if you're looking for a template engine for PHP.

- SithTemplate is **small** - library itself is about ∼80kB of code, ∼1000 SLOC (without comments) in 12 classes (in 10 files), and standard library is single ∼45kB plugin file, ∼900 SLOC (without comments) [measure based on Mercurial head, as of 10-08-2009].
- SithTemplate has relatively **small overhead** - as library tries to shift the overhead from runtime part to the compiler, it has low memory usage. As for speed - it may not be the fastest one available (especially since it [ab]uses slowish PHP object system - no other engine I know generates PHP classes for templates) but it's not the slowest one either - speed overhead is (IMHO at least) reasonable and pretty stable (especially for repeated template rendering). And it's opcode-cache friendly.
- SithTemplate is **free** and **open-source**, with very permissive licensing - used **New BSD License** allows it to be included in both free and commercial, open- and closed-source projects.
- SithTemplate is **generic**, and can process any kind of plain text, not only (X)HTML, like some engines. Web layouts, e-mails, code, configuration files, anything is possible.
- SithTemplate is **extensible** - if included I/O drivers or tags/filters are not enough for your project, you can easily add your own.
- SithTemplate has an **easy API** - create one object, call one method and your template output is ready to be displayed. It also has a **simple, plain-text syntax**, based on Django. No XML writing required, ever!
- SithTemplate supports a **multi-zoned template inheritance** - feature ported from Django (Python web framework) that together with simple syntax makes templates more clear and maintainable, and hierarchical ones feel more natural (than using e.g. header/footer inclusion in every template).
- SithTemplate is **Unicode-aware**. It uses mbstring routines exclusively, both in the core and in the standard library, instead of the core PHP string manipulation functions. It introduces additional dependency, and maybe even slight performance degradation, but it also makes library safer when UTF-8-encoded data is manipulated. And standard filters are shorter and locale-independent (like `lower`).
- SithTemplate has been **tested in production**, and while internals and API has changed in 1.1, established concepts and template syntax are the same. Library is also tested by an automated unit test suite, with over 200 test cases, which keeps old bugs from reappearing. It is tested on three different PHP versions before releasing, ensuring that it will work without compatibility problems on all supported configurations.
- SithTemplate tries to stay **compatible with Django**, and while it introduces its own extensions to the standard library, there are just slight differences in the syntax (the most significant is in the variable access). So if you know Django, then you will have little or no problems using SithTemplate. Also most of the Django's standard tags and filters are implemented in the current version of the library.

1.2.2 Requirements

SithTemplate requires PHP5 (at least 5.1, but 5.2 is recommended) with SPL and mbstring enabled.

PHP versions 5.0 and 6.x are not supported. The library is tested with PHPUnit tests on PHP 5.1, 5.2 and 5.3 before releasing.

Test suite also requires PHPUnit 3.2.

Runtime part of the library requires at least ~200kB of free RAM in simplest cases, and the compiler requires at least 1MB.

1.3 Tutorial

If you want to try SithTemplate, follow tutorial on page SithTemplate tutorial to get yourself familiar with library's concepts and usage.

1.4 Standard library reference

See Standard library page for reference on standard tags and filters.

1.5 Extending SithTemplate

If you're interested in extending library's capabilities, see Extending SithTemplate page.

1.6 Reporting bugs, getting support

You're encouraged to file all found bugs to the project's `bugtracker`, even if you're not sure whether it's library's bug, or just your mistake.

To get support or discuss about SithTemplate, you can visit project's IRC channel, `#sithtemplate` on Freenode network.

Remember to use English in both cases.

Chapter 2

SithTemplate tutorial

2.1 Overview

SithTemplate doesn't require any special installation or initialisation procedure. It uses an single entry point - SithTemplate.php file.

Library distinguishes between compilation time, and runtime - some parts may be used only in one of them (e.g. plugins - they run only during template compilation).

Your application will typically interact with TemplateEnviron class, and its members. This is "public API" or "client API" of the library, which is designed to be as simple as possible (I think nobody likes to dig through hundreds of pages of documentation, just to find right function to do something). Through this API you can create template instances, and retrieve template metadata. It also contains internal object cache, if you're lazy and repeatedly rendering same templates.

To create (and compile, if neccessary) the template object, you need to call either TemplateEnviron::get or TemplateEnviron::cachedGet. It constructs and return an template instance, which is always a subclass of the Template class.

Template objects are independent of the TemplateEnviron and context. Thus, you can create object once, and render it using different runtime environments and variables. To actually render the template, you need to call Template::render method on previously constructed object, passing context array (containing all variables that template uses) and TemplateEnviron object.

You can also render the template using TemplateEnviron::render, in case you don't use different environment objects.

```
<?php
require_once 'SithTemplate.php';

// 1. We create environment
$environ  = new TemplateEnviron;
// 2. Next, we create template object
// Library will take care of the (re)compilation.
// SithTemplate 1.1 introduced unified I/O system,
// which allows you to easily inline small templates in your PHP code.
$template = $environ->get('string://Hello world');
// 3. Finally, we render and display previously created template
// You may notice that display/fetch APIs are gone, replaced by
// generic ones - you need to display template output by yourself.
//
// You can also see that environment object is passed back to the template -
// it is used in several places, like {% include %}-generated code, but passing
// it here, and not during construction, keeps template object more lightweight
// and independent, as it doesn't carry reference to original environment.
// It also eliminates possibility of circular reference, when template object
// is stored in environ's internal cache.
echo $template->render(array(), $environ);

// If you don't want to cache the template object on your own, you can use
// chained calls to cachedGet and render:
$environ->cachedGet('string://Other')->render(array(), $environ);

// If you don't need the object at all, you can call TemplateEnviron::render inst
      ead.
// This call is the same as the chained call above, just shorter and less explici
      t.
$environ->render('string://Other', array());
```

2.2 I/O system

SithTemplate has its own extensible I/O system built-in.

All input and output is handled by so-called "I/O drivers". Library itself doesn't know, whether template is loaded/saved to filesystem, database or maybe across network. All I/O drivers follow ITemplateIODriver interface.

I/O system is transparent - to use non-default driver, all you have to do is to use URI-like DSN as template name, wherever it is supported (e.g. as TemplateEnviron::get argument). Library may refer to the DSNs as "template IDs". If you don't specify the driver (e.g. the part before ://), a default one will be used (`defaultIODriver` setting, see Environment settings).

```
<?php
require_once 'SithTemplate.php';

$environ = new TemplateEnviron;

// SithTemplate 1.1 comes with two I/O drivers bundled:
//
// - "file" I/O - a traditional template-from-file driver.
//   This driver uses "inputPrefix" as source directory with templates,
//   and "outputPrefix" as cache directory, to store metadata and
//   compiled templates' code.
echo $environ->get('template.html')->render(array(), $environ);
// - "string" I/O, which allows you to inline templates in code.
//   This driver uses only "outputPrefix" setting.
echo $environ->get('string://Hai')->render(array(), $environ);
//
// inputPrefix defaults to ./templates/
// outputPrefix defaults to ./templates_c/
```

If you are interested in creating your own, see Extending SithTemplate.

2.3 Environment settings

Every environment has a settings array associated with it. It determines library's behaviour during both runtime and compilation time.

The settings array is stored as TemplateEnviron::$settings (and, as reference, in TemplateCompilerEx::$settings).

```
<?php
require_once 'SithTemplate.php';

// You can change default settings during TemplateEnviron construction,
// by passing associative array to the constructor.
$environ = new TemplateEnviron(array(
 'inputPrefix'  => './templates/',
 'outputPrefix' => './templates_c/',
));

// You can also load settings from INI file, using static named constructor
// See sample-configuration.ini for syntax.
$environ = TemplateEnviron::createFromINI('settings.ini');

// Finally, you can change settings in runtime, by modifying settings
// array directly. Note that some settings won't take effect if changed
// in that way. Refer to documentation for more information.
$environ->settings['recompilationMode'] = TemplateEnviron::RECOMPILE_ALWAYS;
```

2.4 Variables and context

A context is an associative array, that contains arbitarily nested scalars, objects, and other arrays. Context indexes (i.e. variable names) can be any Unicode string that doesn't contain any whitespace, except for "internal", "forloop" and "block" which are special variables used by the standard library (see Special variables).

Template variables are placeholders, that get replaced by appropriate values from the context array at runtime. They are the most basic template construct - without them templates would be quite useless.

This section documents simplest use of template variables - in the standalone `{{ variable expression }}` construct. Final result of that expression **MUST** evaluate to a scalar value (i.e. can be neither array [PHP would convert it to "Array" string] nor object [unless it can be converted to string - see PHP documentation for that]). Since this includes filters, it's important that the **last** filter in the chain produces scalar.

As mentioned above, variables can be filtered before displaying. This is done by appending **filter chain** to the entire expression. Filters can have their own arguments, and are separated in the chain by | (also known as pipe). Chain is executed in defined order (i.e. left to right), and have no length limit (but keep in mind that function calls in PHP are slowish, and filter chains are executed at runtime).

```
<?php
require_once 'SithTemplate.php';

$environ = new TemplateEnviron;

// Context array is passed as first argument to Template::render, or as second
// argument to TemplateEnviron::render.
$tpl = $environ->get('string://{{ foo }} ');

echo $tpl->render(array('foo' => 'first'),  $environ);
echo $tpl->render(array('foo' => 'second'), $environ);
// Will produce: "first second "

// Above is the simplest variable expression. To access nested elements, slightly
       more
// complex syntax is required, presented below, with equivalent PHP code:
//
// - accessing a named array element
//   {{ foo.bar }} is equivalent to $context['foo']['bar']
// - accessing a numeric array index
//   {{ foo.42 }} is equivalent to $context['foo'][42]
// - accessing a named or numeric array index, using value of another variable as
       key
//   {{ foo.[bar] }} is equivalent to $context['foo'][$context['bar']]
//
// Same syntax rules applies to object properties - you just use -> operator inst
      ead of ., e.g.
// {{ foo->bar }}.
//
// This syntax allows you to create very complex constructs, like:
//  {{ [one->[two]].three->four }} which is equivalent to
//  $context[ $context['one']->{$context['two']} ]['three']->four
//
// SithTemplate by default generates code to check whether variable really exists
       in the context
// before it is used, which triggers E_USER_WARNING if it doesn't. This can inter
      fere with "optional"
// variables (e.g. ones used with 'default' filter). You can tell compiler to omi
      t this code, by prefixing
// entire expression with @ sign:
//  {{ @non-existant-variable }}

// Filter chains are built with pipe operator. Filter arguments are comma-separat
```

```
      ed, passed after colon.
//  {{ variable|filter1|filter2:variable2,"foo" }}
// is roughly equivalent (if filters were simply functions) to
//  filter2(filter1($context['variable']), $context['variable2'], 'foo')
```

2.5 Template syntax

SithTemplate uses mostly Django-compatible syntax for templates, and follows its philosophy (but with PHP instead of Python as base language), both found in original docs at http://docs.djangoproject.com/en/dev/topics/templates/#topics-templates.

Every template is just a plain text, with special commands for the compiler:

- variables, already described in Variables and context
- tags, which control template logic (e.g. conditionals, loops)
- comments

SithTemplate includes a plugin called `StdLibEx`, which implements standard library of tags and filters - see Standard library page for reference.

```
{{ this.is.variable }}

There are two kinds of tags:
{% inline %} and {% block %} which require an ending tag {% endblock %}

{# this is single-line comment - they are ignored by the compiler #}
{% comment %}
 and this is
 multi-line comment
 which is implemented as built-in block tag
{% endcomment %}
```

2.6 Template inheritance

One of key concepts of SithTemplate is template inheritance, which allows you to build hierarchy of templates, e.g. the three-level approach Django docs mentions:

1. A base template, containing general layout of the site
2. A section template, containing more specific layout bits for the site section
3. A detail templates, containing the most specific bits for every page type

Template inheritance increases maintainability and readability, and the hierarchy feels more natural than with e.g. header and footer included in every template (which is also error-prone).

SithTemplate uses so-called "multi-zoned template inheritance", which means that parent template defines any number of blocks (just like parent class would define a number of methods), and children templates override them with their own content (again, like children classes would override methods), optionally including parent block's code within new block (using {{ block }} special variable). Blocks are created using standard library's {% block %} tag, and inheritance is done by {% extends %} tag.

An example inheritance (parent and then child):

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <link rel="stylesheet" href="style.css">
  <title>{% block title %}Awesome HTML5 site{% endblock %}</title>
 </head>
 <body>
  <div id="main">
  {% block contents %}
   <p>
    This is default content of the block.
   </p>
   <p>
    It will be used in case no children template will override it,
    or when parent template will be rendered directly.
   </p>
  {% endblock %}
  </div>

  <div id="somethingelse">
  {% block other %}
   <p>
    Foo.
   </p>
  {% endblock %}
  </div>
</body>
</html>


{# This is the most important line: #}
{% extends "05_inheritance_parent.html" %}

{% block contents %}
 <p>
  And here is overriden contents of this block!
 </p>
{% endblock %}

{% block other %}
 <p>
  This one shows how to use parent block's contents.
 </p>
 {{ block.super }}
{% endblock %}
```

The result of rendering child template would be (actually whitespace would be different, but it's irrelevant here):

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <link rel="stylesheet" href="style.css">
  <title>Awesome HTML5 site</title>
 </head>
 <body>
  <div id="main">
   <p>
    And here is overriden contents of this block!
   </p>
  </div>

  <div id="somethingelse">
   <p>
    This one shows how to use parent block's contents.
   </p>
   <p>
```

```
    Foo.
   </p>
  </div>
</body>
</html>
```

2.7 Security settings in SithTemplate

SithTemplate includes several settings that are referred to as "security settings", and implemented by the compiler and the standard library. These include variable autoescaping (using escape and safe filter), I/O restrictions (if used, templates will be bound to the originally used I/O driver), element whitelists and blacklists (you can sandbox templates by restricting access to plugins, tags, filters, and plain PHP functions), and {{ internal }} access restrictions. See TemplateEnviron::$settings for reference.

```
<?php
require_once 'SithTemplate.php';

$environ = new TemplateEnviron;

// All security settings are set using environment's setting array.
// Some of them may be enforced at runtime, and some at compile time,
// see TemplateEnviron::$settings documentation for reference.

// The most common is variable autoescaping, which applies "escape" filter
// to all stand-alone variables (i.e. {{ vars }}), unless they are marked
// with "safe" pseudofilter.
// Autoescaping is turned on with "autoEscape" boolean setting.
$environ->settings['autoEscape'] = true;
$environ->render('string://{{ var }}', array('var' => '<b>'));      // will retur
      n "&lt;b&gt;"
$environ->render('string://{{ var|safe }}', array('var' => '<b>')); // will retur
      n "<b>"

// Next, there are I/O restriction settings. They allow you to enforce specific I
      /O driver,
// e.g. when you load template using your own db:// driver, and you don't want lo
      aded template
// to use any other I/O driver, like file:// or string://.
// Note that this is a bit primitive, and may be replaced sometime in the future.

// I/O restrictions are turned on by "restrictIncludeIO" and "restrictExtendIO" b
      oolean settings.
$environ->settings['restrictIncludeIO'] = true;
$environ->render('string://{% include "string://test" %}', array());    // will r
      eturn "test"
$environ->render('string://{% include "file://test.html" %}', array()); // will r
      aise TemplateError

// Next, there are {{ internal }} access restrictions (again, a bit primitive and
       boolean only).
// Since {{ internal }} allows template to access global constants and supergloba
      l arrays
// (like $_SERVER or $_ENV), it may introduce security risk in sandboxed environm
      ent
// (e.g. when templates are loaded from DB, and users can edit them).
// {{ internal }} restrictions can be set by turning off "allowInternalRequest"
// and/or "allowInternalConstants" boolean settings.
// Since this is boolean-only and a bit inconsistent, it may get replaced.
$environ->render('string://{{ internal.request.ENV.PATH.0 }}', array()); // will
      return $_ENV['PATH'][0]
$environ->settings['allowInternalRequest'] = false;
$environ->render('string://{{ internal.request.ENV.PATH.0 }}', array()); // will
      raise TemplateError
```

```
// Finally, there are security lists, that allows you to handpick plugins, tags,
      filters and
// plain PHP functions that templates are allowed to use. Lists are the most comp
      lex of security
// settings, as they support multiple modes of evaluation (allow all, deny; allow
      , deny; deny, allow; deny all, allow),
// and wildcards (TemplateEnviron::SECURITY_MATCH_EVERYTHING).
// Evaluation mode is controlled by "securityEvalMode" enumerative setting, and l
      ists themselves
// are stored in several array settings: "allowedPlugins", "allowedTags", "allowe
      dFilters", "allowedFunctions"
// and their "disallowed*" counterparts.
$environ->settings['securityEvalMode'] = TemplateEnviron::SECURITY_DENY_ALL; // m
      ost restrictive setting
$environ->settings['allowedTags']      = array('block'); // you don't have to spe
      cify ending tags
$environ->render('string://{% block foo %}foo{% endblock %}', array()); // will r
      eturn "foo"
$environ->render('string://{% comment %}foo{% endcomment %}', array()); // will r
      aise TemplateError
```

2.8 Error handling

SithTemplate uses PHP exception mechanism to report errors (and standard `trigger_error` to report warnings). It uses single exception class - TemplateError, which defines several class constants that indicates error groups.

```
<?php
require_once 'SithTemplate.php';

$environ = new TemplateEnviron;

// You should always remember about error handling
// If error occurs during template compilation, exception message
// may contain template file and approx. line of the mistake.

// Errors are grouped - every group has it's own errorcode, specified
// as class constants in TemplateError.
try {
 $environ->render('string://{% bkock foo %}Typos are evil.{% endblock %}', array(
      ));
} catch (TemplateError $e) {
 echo $e->getMessage(); // Unknown tag ...
 echo $e->getCode();    // TemplateError::E_UNKNOWN_TAG
}
```

Chapter 3

Extending SithTemplate

TODO: describe extending SithTemplate, compiler's API, etc.

3.1 Extending: AST nodes

TODO: describe AST, and how tags may affect it

3.2 Extending: handlers

TODO: describe handlers for tags, filters and hooks

3.3 Extending: hooks

TODO: describe available hooks

Chapter 4

Standard library

SithTemplate comes with one plugin - `StdLibEx`.

It implements so-called **standard library**, that is set of tags and filters always available (by default, at least, as you can opt to not use it at all) to all templates. Standard library tries to be Django-compatbile, while providing several SithTemplate-specific extensions.

4.1 Standard tags

4.1.1 {% autoescape %}

`{% autoescape on|off %} ... {% endautoescape %}`

Activates or deactivates variable auto-escaping inside the block. Also see escape and safe filters.

```
{# assumptions: variable = "<b>foo</b>", global autoescape is off #}

{{ variable }} {# => <b>foo</b> #}

{% autoescape on %}
 {{ variable }}      {# => &lt;b&gt;foo&lt;/b&gt; #}
 {{ variable|safe }} {# => <b>foo</b> #}
{% endautoescape %}
```

4.1.2 {% block %}

`{% block <name> %} ... {% endblock %}`

`{% block <name> store %} ... {% endblock %}` **(non-Django extension)**

Creates a new overridable, named block (see Template inheritance). `name` argument must be non-quoted string. You can access {{ block }} special variable from within the block.

If `store` is used, then block contents will be remembered, but not displayed (see {% putblock %}).

```
{% block foo %}
 <p>Hello</p> {# will be outputted #}
{% endblock %}

{% block bar store %}
 <p>world</p> {# will not be outputted #}
{% endblock %}
```

4.1.3 {% call %}

Non-Django tag.

`{% call <callback> [<argument> [<argument> [...]]] %}`

`{% call <callback> [<argument> [<argument> [...]]] as <variable> %}`

Calls raw PHP function, specified by `callback` (checked against security lists - see Security settings in SithTemplate), and either displays it (first signature) or stores in a new context variable (second signature).

`callback` can be a quoted string constant, or a variable expression (that must evaluate to `call_user_func_array` compatible callback value). You can pass as many arguments as you need; every `argument` can be either constant or variable. If second version is used, `variable` must be a simple variable name (i.e. it's not parsed as an expression, and cannot contain access operators, or filters).

```
{% call "sha1" "foo" %}            {# => 0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33
       #}
{% call "sha1" "foo" as fooHash %} {# will create fooHash variable instead of out
      putting #}

<p>{{ fooHash }}</p> {# => 0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33 #}
```

4.1.4 {% cycle %}

`{% cycle <value> <value> [<value> [...]] %}`

`{% cycle <value> <value> [<value> [...]] as <name> %}`

`{% cycle <name> %}`

Cycles through given list of values. It can be used in two ways: inside the loop (first signature), or outside, as named cycle (second signature creates named cycle, third calls already created one).

`value` can be either constant or filtered variable expression. `name` must be non-quoted constant string.

```
{% for entry in entries %}
 {% cycle "red" "blue" "green" %}
{% endfor %}

{% cycle "red" "blue" "green" as rgbCycle %} {# => red #}
{% cycle rgbCycle %}                         {# => blue #}
{% cycle rgbCycle %}                         {# => green #}
{% cycle rgbCycle %}                         {# => red #}
```

4.1.5 {% debug %}

`{% debug %}`

As for now, it only `var_dump's` the context. Maybe will be extended in the future.

4.1.6 {% extends %}

`{% extends <template> %}`

Extends given template. `template` must be a quoted constant string, and be a correct template ID (see I/O system). Argument is tested against `restrictExtendIO` setting (see Security settings in SithTemplate).

If you use `{% extends %}` more than once in one template, an error will be raised (see Error handling).

For more complex example on inheritance, see Template inheritance.

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

{# extending from string is allowed, but very limited ATM #}
{# extending from a different I/O driver is perfectly fine, though #}
{% extends "string://another" %}

{# note that in one template, there may be only one {% extends %} #}
```

4.1.7 {% filter %}

`{% filter <filters> %} ... {% endfilter %}`

Filters the block contents. `filters` must be a correct filter chain (i.e. a variable expression, but without variable part; see Variables and context), and contain at least one filter.

```
{% filter lower|cut:"foo" %}
 FoO bAr FoO {# => bar #}
{% endfilter %}
```

4.1.8 {% firstof %}

`{% firstof <variable> <variable> [<variable> [...]] [<fallback>] %}`

Outputs first variable that exists and evaluates to `true`, or a fallback value (unless it's not specified, then nothing is outputted).

`variable` must be a correct variable expression (see Variables and context), and `fallback`, if specified, must a quoted constant string.

```
{# assumptions: foo and bar does not exist, baz exists #}
{# foo = "foo", bar = "bar", baz = "baz" #}

{% firstof foo bar baz %}    {# => baz #}
{% firstof foo bar "none" %} {# => none #}
```

4.1.9 {% for %} and {% empty %}

`{% for [<key>,] <value> in <iterable> %} ... [{% empty %} ...] {% endfor %}`

Iterates through variable using `foreach` statement. `key` and `value` must be simple variable names (no filters, no operators), `iterable` must be a variable expression evaluating to an iterable value (i.e. an array, or an object that implements `Traversable` interface).

`{% empty %}` can be used to specify alternate block, which will be used if `iterable` yields no results (i.e. an empty array).

You can also access {{ internal }} special variable from inside the for loop.

```
{# assumptions: entries is a non-empty array, tags is an empty array #}

{# this will output the entries array #}
{% for entry in entries %}
 {{ entry }}
{% empty %}
 No entries.
{% endfor %}

{# this will output "No tags." #}
{% for tag in tags %}
 {{ tag }}
{% empty %}
 No tags.
{% endfor %}
```

4.1.10 {% if %}, {% else %} and {% elseif %}

Non-Django behaviour: Django has simpler `{% if %}`, without rich comparison operators or grouping, and also have no `{% elseif %}`.

`{% if <condition> %} ... [{% elseif <condition> %} ... [{% elseif <condition> %} ... [...]]] [{% else %} ...] {% endif %}`

Conditional block. You can specify alternate condition blocks using one or more `{% elseif %}` tags, and an `{% else %}` tag (only one else is allowed).

`condition` is a conditional expression that supports statement grouping (like in PHP, using parentheses) and rich operators:

- `eq` - PHP's ==
- `neq` - PHP's !=
- `lt` - PHP's <
- `lte` - PHP's <=
- `gt` - PHP's >
- `gte` - PHP's >=
- `and` - PHP's &&
- `or` - PHP's ||
- `not` - PHP's !
- `id` - PHP's === (**added in 1.1a2**)
- `nid` - PHP's !== (**added in 1.1a2**)

```
{% if foo %}
 foo
{% elseif ((bar eq "foo") and baz) %}
 bar
{% elseif quux %}
 baz
{% else %}
 something else
{% endif %}
```

4.1.11 {% ifchanged %} and {% else %}

This tag may only appear inside the loop.

`{% ifchanged %} ... [{% else %} ...] {% endifchanged %}`

On every iteration, checks whether its contents has changed, and outputs correct block accordingly (or nothing, if it hasn't changed, and `{% else %}` block was not given).

`{% ifchanged <variable> %} ... [{% else %} ...] {% endifchanged %}`

Behaves like the first signature, but compares variable value instead of block content. `variable` must be a filtered variable expression.

4.1.12 {% ifequal %}, {% ifnotequal %} and {% else %}

`{% ifequal <variable> <variable> %} ... [{% else %} ...] {% endif %}`

`{% ifnotequal <variable> <variable> %} ... [{% else %} ...] {% endif %}`

Simplified versions of `{% if %}` provided for Django compatibility. Equivalent to `{% if <variable> eq <variable> %}` and `{% if <variable> neq <variable> %}`, accordingly.

4.1.13 {% include %}

`{% include <template> %}`

Includes another template's contents. `template` must be either quoted constant string, or a variable expression evaluating to correct template ID (see I/O system). Argument is tested against `restrictIncludeIO` setting (see Security settings in SithTemplate).

```
{% include "template.html" %}
{% include variable %} {# => contents of the variable will be used #}
```

4.1.14 {% load %}

This tag is a built-in (e.g. a part of the core itself, not standard library)

`{% load <plugin> %}`

Loads new plugin for current template, making its tags, filter and hooks immediately accessible. `plugin` must be a non-quoted constant string.

This tag respects security lists (see Security settings in SithTemplate).

```
{% load SomeLibrary %}
```

4.1.15 {% meta %}

Non-Django tag.

`{% meta <name> <value> %}`

Creates an entry in template's metadata. `name` must be a non-quoted constant string, `value` must be a quoted constant string.

Custom metadata will be prefixed `user:` internally, and can be retrieved using TemplateEnviron::getMeta API call.

```
{% meta foo "This template's foo is bar" %}
{% meta bar "This template's bar is foo" %}
```

4.1.16 {% now %}

`{% now <format> %}`

Outputs current timestamp, formatted using `format` argument, which must be either quoted constant string, or a variable expression that evaluates to string.

For format codes see PHP's `date` function documentation.

```
{% now "d-m-Y, H:i:s" %}
```

4.1.17 {% putblock %}

Non-Django tag.

`{% putblock <name> %}`

Outputs previously defined block (see {% block %}). You can use store blocks to maximize code reuse within single template (blocks are always evaluated with current context).

Keep in mind that blocks are internal to template, and can only be inherited. You cannot e.g. `{% putblock %}` a block from a included template (see {% include %}).

```
{% block foo store %}Hello{% endblock %} world {% putblock foo %} {# => world Hel
      lo #}
```

4.1.18 {% spaceless %}

`{% spaceless %} ... {% endspaceless %}`

Removes all whitespace between HTML tags (but not inside the tags).

```
{% spaceless %}
 <p>
            <b>  foo  </b>
 </p>
{% endspaceless %}

{# => <p><b>  foo  </b></p> #}
```

4.1.19 {% templatetag %}

`{% templatetag <tag> %}`

Inserts literal bit of template syntax, `tag` must be one of:

- `openblock` - inserts `{%`
- `closeblock` - inserts `%}`
- `openvariable` - inserts `{{`
- `closevariable` - inserts `}}`
- `opencomment` - inserts `{#`
- `closecomment` - inserts `#}`
- `openbrace` - inserts `{` (**added in 1.1a2**)
- `closebrace` - inserts `}` (**added in 1.1a2**)

Non-Django behaviour follows.

In addition to above, Django-compatible tags, SithTemplate also defines several aliases on its own (although you are encouraged to use full versions, these are kept mainly for backwards compatibility):

- ob, ot and opentag - alias to openblock
- cb, ct and closetag - alias to closeblock
- ov, and openvar - alias to openvariable
- cv, and closevar - alias to closevariable
- oc - alias to opencomment
- cc - alias to closecomment

4.1.20 {% widthratio %}

{% widthratio <value> <maxValue> <constant> %}

Calculates width ratio using formula: round((value/maxValue)*constant).

Both value and maxValue must be either numeric constants, or a variable expression evaluating to integer or float. constant must be an integer constant.

```
{# assumptions: current = 175, max = 200 #}

{% widthratio current max 100 %} {# => 88 #}
```

4.1.21 {% with %}

{% with <variable> as <name> %} ... {% endwith %}

Creates a new variable, using value of filtered variable expression, visible only within the block.

variable must be a filtered variable expression, and name must be a simple variable name.

```
{% with one.two->three.[four]|lower as simpler %}
 {{ simpler }}
{% endwith %}
```

4.2 Standard filters

4.2.1 add

add:<argument>

Adds the argument to the variable. argument must be either constant string (operator . (join strings) is used), or constant number/numeric variable (operator + (add numbers) is used).

```
{# assumptions: var = 5, var2 = "foo" #}

{{ var|add:5 }}       {# => 10 #}
{{ var2|add:" bar" }} {# => foo bar #}
```

4.2.2 addslashes

addslashes

See PHP's addslashes function.

```
{# var = foo\bar' #}

{{ var|addslashes }} {# => foo\\bar\' #}
```

4.2.3 capfirst, lower, upper, title

`capfirst`

`lower`

`upper`

`title`

Changes capitalization of the string variable - `capfirst` capitalizes first letter, `lower` converts entire string into lowercase, `upper` converts entire string into uppercase, and `title` converts entire string into titlecase (e.g. converts every first letter of a word into uppercase).

Also see PHP's `mb_convert_case` function.

```
{# var = hEllo woRld #}

{{ var|capfirst }} {# => HEllo woRld #}
{{ var|lower }}    {# => hello world #}
{{ var|upper }}    {# => HELLO WORLD #}
{{ var|title }}    {# => HEllo WoRld #}
```

4.2.4 cut

`cut:<argument>`

Removes the argument from the string variable. `argument` can be a variable or a constant. Uses PHP's `preg_replace`.

```
{# var = "foo bar baz" #}

{{ var|cut:"bar" }} {# => foo  baz #}
```

4.2.5 date

`date:<format>`

Formats the timestamp according to the `format`. See PHP's `date` function.

```
{# var = 123456789 #}

{{ var|date:"d-m-Y, H:i:s" }} {# => 29-11-1973, 22:33:09 #}
```

4.2.6 default, default_if_none

`default:<value>`

`default_if_none:<value>`

Uses default value if variable doesn't exist (or evaluates to false, `default`) or is NULL (`default_if_none`). `value` can be a constant, or a variable.

```
{# nonexistant does not exist, nullvar is NULL #}

{{ @nonexistant|default:"none" }}    {# => none; silenced, otherwise a warning wo
      uld be issued #}
{{ nullvar|default_if_none:"none" }} {# => none #}
```

4.2.7 divisibleby

`divisibleby:<value>`

Returns `true` if variable is evenly divisible by `value` (which can be either variable or a constant number different than zero).

```
{# var1 = 2, var2 = 3 #}

{% if var1|divisibleby:2 %} yes {% else %} no {% endif %} {# => yes #}
{% if var2|divisibleby:2 %} yes {% else %} no {% endif %} {# => no #}
```

4.2.8 escape and safe

`escape`

`safe`

`escape` applies `htmlspecialchars` function to the variable. `safe` is a pseudofilter (i.e. it's not actually defined and it's handled by hooks instead) that marks variable as safe (i.e. already escaped, or that it doesn't need escaping at all) causing autoescaping (see Security settings in SithTemplate) to skip it.

```
{# var = "<b>foo</b>" #}

{{ var|escape }} {# => &lt;b&gt;foo&lt;/b&gt; #}
```

4.2.9 filesizeformat

`filesizeformat`

Formats the integer variable as human-readable filesize (e.g. to bytes, kilobytes, megabytes, gigabytes).

```
{# var1 = 1000, var2 = 41211, var3 = 5230121, var4 = 5232338952 #}

{{ var1|filesizeformat }} {# => 1000 b #}
{{ var2|filesizeformat }} {# => 40.25 kB #}
{{ var3|filesizeformat }} {# => 4.99 MB #}
{{ var4|filesizeformat }} {# => 4.87 GB #}
```

4.2.10 fix_ampersands

`fix_ampersands`

Changes every `&` into `&` HTML entity.

```
{# var = "foo&bar" #}

{{ var|fix_ampersands }} {# => foo&amp;bar #}
```

4.2.11 join

`join:<separator>`

`implode's` an array variable using `separator` (a variable, or a constant).

```
{# var = array('a', 'b', 'c') #}

{{ var|join:", " }} {# => a, b, c #}
```

4.2.12 length, length_is

`length`

`length_is:<number>`

`length` returns length of a string or count of an array elements. `length_is` compares that length with a `number` (a variable, or a constant number), and returns boolean.

```
{# var1 = "foo", var2 = array('a', 'b') #}

{{ var1|length }}                                          {# => 3 #}
{% if var2|length_is:3 %} yes {% else %} no {% endif %} {# => no #}
```

4.2.13 linebreaks, linebreaksbr

`linebreaks`

`linebreaksbr`

`linebreaks` converts newlines in the variable into HTML paragraphs and linebreaks. `linebreaksbr` applies `nl2br`.

```
{% comment %}
 var = "foo

 bar
 baz"
{% endcomment %}

{{ var|linebreaks }}
{% comment %}
 Outputs (whitespace may vary):
 <p>foo</p>

 <p>bar<br />
 baz</p>
{% endcomment %}

{{ var|linebreaksbr }}
{% comment %}
 Outputs (whitespace may vary):
 foo<br />
 <br />
 bar<br />
 baz
{% endcomment %}
```

4.2.14 ljust, rjust

`ljust:<width>`

`rjust:<width>`

Aligns the text inside the field of given `width`.

```
{# var = "foo" #}

{{ var|ljust:5 }} {# => "foo  " #}
{{ var|rjust:5 }} {# => "  foo" #}
```

4.2.15 make_list

`make_list`

Splits the string/numeric variable into an array of characters/digits.

```
{# var = "foo" #}

{{ var|make_list|join:", " }} {# => f, o, o #}
```

4.2.16 pluralize

`pluralize`

Returns plural suffix `-s` if the filtered variable evaluates to integer bigger than 1.

`pluralize:<suffix>`

Returns user-specified plural suffix instead. `suffix` must be a quoted constant string.

`pluralize:<suffixes>`

Returns either singular or plural suffix, both user-specified in `suffixes`, and delimited by a comma. `suffixes` must be a quoted constant string.

```
{# var1 = 1, var2 = 7 #}

{{ var1|pluralize }} {# outputs nothing #}
{{ var2|pluralize }} {# outputs 's' #}

{{ var1|pluralize:"es" }} {# outputs nothing #}
{{ var2|pluralize:"es" }} {# outputs 'es' #}

{{ var1|pluralize:"e,es" }} {# outputs 'e' #}
{{ var2|pluralize:"e,es" }} {# outputs 'es' #}
```

4.2.17 random

`random`

Returns random element of the filtered array.

```
{# var = array('a', 'b') #}

{{ var|random }} {# will output either 'a' or 'b' #}
```

4.2.18 removetags

`removetags`

Applies PHP's `strip_tags` on filtered variable.

```
{# var = "<b>foo</b>" #}

{{ var|removetags }} {# => foo #}
```

4.2.19 slugify

`slugify`

Converts the string into a URL-friendly "slug", i.e. converts it to lowercase, strips HTML tags, converts all whitespace and underscores into dashes, and removes all remaining characters that are neither dash, nor alphanumeric.

```
{# UTF-8 ahead #}
{# var = "Hai aą   #1331" #}

{{ var|slugify }} {# => hai--1331 #}
```

4.2.20 urlencode, urldecode

`urlencode`

`urldecode`

PHP's `urlencode` and `urldecode`, respectively.

```
{# var = "http://example.com" #}

{{ var|urlencode }}           {# => http%3A%2F%2Fexample.com #}
{{ var|urlencode|urldecode }} {# => http://example.com #}
```

4.2.21 wordcount

`wordcount`

Counts the words in the string, using PHP's `str_word_count`.

```
{# var = "foo bar baz" #}

{{ var|wordcount }} {# => 3 #}
```

4.2.22 wordwrap

`wordwrap:<length>`

Applies PHP's `wordwrap`, using given `length` (a variable or a constant number).

```
{# var = "foobar foobar foobar" #}

{{ var|wordwrap:6|linebreaksbr }}
{% comment %}
```

```
 Will output:
 foobar<br />
 foobar<br />
 foobar
{% endcomment %}
```

4.3 Special variables

4.3.1 {{ block }}

This variable is accessible from within {% block %} tag.

It contains only one subkey - `{{ block.super }}` which evaluates to the contents of parent block.

```
{# parent template: #}
{% block something %}
 <p>something!</p>
{% endblock %}

{# child template: #}
{% block something %}
 <p>something else?</p> {{ block.super }}
{% endblock %}

{# output: <p>something else?</p> <p>something!</p> #}
{# whitespace may vary #}
```

4.3.2 {{ forloop }}

This variable is accessible from within the `for` loop (see {% for %} and {% empty %}).

It contains several subkeys:

- `{{ forloop.counter }}` - current iteration, starting from 1
- `{{ forloop.counter0 }}` - current iteration, starting from 0
- `{{ forloop.revcounter }}` - number of iterations left, ending on 1
- `{{ forloop.revcounter0 }}` - number of iterations left, ending on 0
- `{{ forloop.first }}` - true if first iteration
- `{{ forloop.last }}` - true if last iteration
- `{{ forloop.parentloop }}` - `forloop` variable of the parent loop, available in nested loops

```
{% for v in vs %}
 {% if forloop.first %}first!{% endif %}
 {% if forloop.last %}last!{% endif %}
 {{ forloop.counter }}/{{ forloop.counter0 }}
 {{ forloop.revcounter }}/{{ forloop.revcounter0 }}
 {% for x in xs %}
  {{ forloop.parentloop.counter }}
 {% endfor %}
{% endfor %}
```

4.3.3 {{ internal }}

This variable is accessible in the entire template.

It contains several subkeys:

- `{{ internal.request }}` - allows you to access PHP's superglobals (also see Security settings in SithTemplate)
- `{{ internal.const }}` - allows you to access PHP's constants (also see Security settings in SithTemplate)
- `{{ internal.version }}` - evaluates to the current engine's version (e.g. SITHTEMPLATE_VERSION)
  - keep in mind that this value is hardcoded into the template's code, so it won't change without recompilation

```
{{ internal.request.POST.foo }}
{{ internal.const.PHP_VERSION }}
{{ internal.version }}
```

Chapter 5

Todo List

Member TemplateCompilerEx::generateCode(TemplateNodeEx $root) Is it needed, or maybe TemplateCompilerEx::compile should do it?

Member TemplateCompilerEx::handleNode(TemplateNodeEx $node) Maybe it should be merged with TemplateCompilerEx::handleChildren?

Member TemplateUtils::splitEscaped($delimiter, $expression) Better way?

File CompilerEx.php Better variable parser?

Chapter 6

Class Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 7

Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 8

File Index

8.1 File List

Here is a list of all files with brief descriptions:

Chapter 9

Class Documentation

9.1 ITemplateIODriver Interface Reference

Interface required for all I/O drivers.

Public Member Functions

- upToDate (array &$settings, &$template, $mode)
  Should check whether given template is up-to-date.
- includeCode (array &$settings, &$template)
  Should include template's code into global namespace.
- className (array &$settings, &$template)
  Should return template's classname.
- loadTemplate (array &$settings, &$template)
  Should read template source code as whole, and return it.
- loadMetadata (array &$settings, &$template)
  Should read template's metadata, and return it.
- saveTemplate (array &$settings, &$template, &$code)
  Should save compiled template code.
- saveMetadata (array &$settings, &$template, array &$metadata)
  Should save template metadata.

9.1.1 Detailed Description

Interface required for all I/O drivers.

I/O machinery is separated from template plugins.

Definition at line 15 of file IIODriver.php.

9.1.2 Member Function Documentation

9.1.2.1 ITemplateIODriver::upToDate (array &$ *settings*, &$ *template*, $ *mode*)

Should check whether given template is up-to-date.

If driver uses `recompilationMode` setting, then it should use supplied `$mode` argument instead, to allow per-template mode override. Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

← ***$mode*** Recompilation mode

Return values:

true Template is up-to-date - no (re)compilation is needed

false Template must be (re)compiled

Implemented in TemplateFileIO, and TemplateStringIO.

9.1.2.2 ITemplateIODriver::includeCode (array &$ *settings*, &$ *template*)

Should include template's code into global namespace.

It must ensure that no code redefinition will happen. Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Included class name

Implemented in TemplateFileIO, and TemplateStringIO.

9.1.2.3 ITemplateIODriver::className (array &$ *settings*, &$ *template*)

Should return template's classname.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Class name

Implemented in TemplateFileIO, and TemplateStringIO.

Referenced by TemplateCompilerEx::compile().

9.1.2.4 ITemplateIODriver::loadTemplate (array &$ *settings*, &$ *template*)

Should read template source code as whole, and return it.

SithTemplate ensures that this will be called only when compilation is needed, so no additional checks are needed. Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Whole template source

Implemented in TemplateFileIO.

Referenced by TemplateCompilerEx::compile().

9.1.2.5 ITemplateIODriver::loadMetadata (array &$ *settings*, &$ *template*)

Should read template's metadata, and return it.

Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Template metadata or `false`.

Implemented in TemplateFileIO.

9.1.2.6 ITemplateIODriver::saveTemplate (array &$ *settings*, &$ *template*, &$ *code*)

Should save compiled template code.

Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

← ***$code*** Template code

Return values:

true Template has been saved

false An error occured

Implemented in TemplateFileIO.

Referenced by TemplateCompilerEx::compile().

9.1.2.7 ITemplateIODriver::saveMetadata (array &$ *settings*, &$ *template*, array &$ *metadata*)

Should save template metadata.

Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

← ***$metadata*** Metadata

Return values:

true Metadata has been saved

false An error occured

Implemented in TemplateFileIO.

Referenced by TemplateCompilerEx::compile().

The documentation for this interface was generated from the following file:

- IIODriver.php

9.2 ITemplatePlugin Interface Reference

Interface required for all plugins.

Public Member Functions

- providedHandlers ()

  Plugin's entry point, must return array of provided handlers (see Extending SithTemplate).

9.2.1 Detailed Description

Interface required for all plugins.

Since 1.1 plugins are used during compilation phase exclusively, there are no more 'runtime/compile-time libraries'.

Both tags and filters now use code inlining (e.g. they embed code directly in template's code).

Definition at line 19 of file IPlugin.php.

9.2.2 Member Function Documentation

9.2.2.1 ITemplatePlugin::providedHandlers ()

Plugin's entry point, must return array of provided handlers (see Extending SithTemplate).

Handler array structure:

- `tags` - array of tag handlers
  - `handler` (array/string, required) - whatever `call_user_func_array` can handle. If you want to use stand-alone function, you must also set `standalone` key to `true`, otherwise library will assume that you meant `array`($this,$handler).
  - `standalone` (boolean, optional) - set it, if you want to use non-member function as `handler`.
  - `type` (string, required) - tag type. Either `'block'` or `'inline'`.
  - `minArgs` (integer, optional) - minimum arguments this tag needs. Defaults to 0.
  - `parent` (string, optional) - enforcement of specific immediate parent. Used in e.g. `else/elseif`. Compiler will raise error if immediate parent of this tag isn't one specified here. Defaults to nothing. May contain wildcard * (e.g. `if*` matches `if`, `ifchanged`, `ifequals`, etc.).
- `filters` - array of filter handlers
  - `handler` (array/string, required) - see above.
  - `standalone` (boolean, optional) - see above.
  - `minArgs` (integer, optional) - see above.
- `hooks` - array of hook handlers (for available hookpoints see Extending: hooks)
  - `handler` (array/string, required) - see above.

- `standalone` (boolean, optional) - see above.

Returns:

Assoc. array of handlers

Implemented in TemplateStdLibExPlugin.

The documentation for this interface was generated from the following file:

- IPlugin.php

9.3 Template Class Reference

Abstract base class for templates.

Public Member Functions

- render (array $ctx, TemplateEnviron $environ)
  Render template using given context.
- _main ($environ)
  Contents of template's main block.

Protected Member Functions

- warnVar ($variable)
  Warn about non-existant variable.
- invalidVar ($variable, $message)
  Fail after encountering an invalid variable (e.g.

Protected Attributes

- $ctx = array()
  Current context.

9.3.1 Detailed Description

Abstract base class for templates.

Definition at line 13 of file Base.php.

9.3.2 Member Function Documentation

9.3.2.1 Template::render (array $ *ctx*, TemplateEnviron $ *environ*) [final]

Render template using given context.

Parameters:

← ***$ctx*** Context (assoc. array with template variables)
← ***$environ*** Environment to use (TemplateEnviron instance)

Returns:

Result as string

Definition at line 26 of file Base.php.

References _main().

9.3.2.2 Template::warnVar ($ *variable*) [final, protected]

Warn about non-existant variable.

Parameters:

← ***$variable*** Raw variable name, as encountered in template source

Definition at line 36 of file Base.php.

9.3.2.3 Template::invalidVar ($ *variable*, $ *message*) [final, protected]

Fail after encountering an invalid variable (e.g.

non-iterable used as loop source).

Parameters:

← ***$variable*** Raw variable name, as encountered in template source

← ***$message*** Additional error details

Definition at line 49 of file Base.php.

References TemplateError::E_INVALID_VAR.

9.3.2.4 Template::_main ($ *environ*) [abstract]

Contents of template's main block.

Parameters:

← ***$environ*** TemplateEnviron instance

Referenced by render().

9.3.3 Member Data Documentation

9.3.3.1 Template::$ctx = array() [protected]

Current context.

Definition at line 17 of file Base.php.

The documentation for this class was generated from the following file:

- Base.php

9.4 TemplateCompilerEx Class Reference

Primary compiler driver.

Public Member Functions

- __construct ()
  Constructor.
- reset ()
  Resets compiler to pristine state, and loads plugins specified in `'loadPlugins'` *setting.*
- compile (ITemplateIODriver $io, $template)
  Compiles given template into output package.
- handleChildren (array &$children)
  Generates code from given node's children.
- createBlock ($block, TemplateNodeEx $node)
  Creates code block from raw node.
- handleNode (TemplateNodeEx $node)
  Creates code from given node.
- parseVariableExpression (TemplateNodeEx $node, $variable)
  Parses variable expression, and creates runtime PHP access code.
- parseFilterChain (TemplateNodeEx $node, $filterExpr, $code)
  Handles filter chains.
- raise (TemplateNodeEx $node, $message, $code)
  Raises an error, appending "(in template <file> somewhere around line <line>)" using given node.
- raiseIf ($cond, TemplateNodeEx $node, $message, $code)
  Shorthand for conditional call to TemplateCompilerEx::raise.
- findAlternativeBranch (TemplateNodeEx $node, $tag)
  Used to find and isolate alternative branch of given node, starting with given inline tag.
- generateUniqueBlock ($idPrefix, $blockPrefix= 'custom:', $keyLength=5)
  Generates prefixed block name that is guaranteed to be unique in current template.

Public Attributes

- $settings = null
  Current settings (reference to TemplateEnviron::$settings).
- $plugins = null

Registry of available plugins.

- $loadedPlugins = array()

  Registry of loaded plugins (per-template).

- $blocks = array()

  Already constructed code blocks.

- $metadata = array()

  Template's metadata.

- $className = null

  Template's classname.

Private Member Functions

- createAST (&$tpl)

  Creates and returns an Abstract Syntax Tree of given template.

- parserGetNextToken (array &$tokens)

  Fetches next token from given stream, preprocesses it, and stores it in TemplateCompilerEx::$parserCurrentToken.

- parserEncounteredEndTag (TemplateNodeEx $node, $tag, $type)

  Checks whether the parser has encountered ending tag with given name.

- createNodeFromToken (TemplateNodeEx $node)

  Converts current token's array into the node, and adds it to given node's children.

- parseTokenStream (TemplateNodeEx $node, array &$tokens, $parseUntil=null, $blockTagType=null)

  Parses token stream until it runs out of tokens, or when it encounters given tag name.

- generateCode (TemplateNodeEx $root)

  Constructs blocks and generates code for the entire AST.

- handleTag (TemplateNodeEx $node)

  Handles `'blockTag'` and 'inlineTag' nodes.

- handleVariable (TemplateNodeEx $node)

  Handles `'var'` node.

- generateVariableAST ($variable)

  Generates syntax tree (using nested arrays ATM) from given variable expression.

- generateVariableCode (TemplateNodeEx $node, &$ast)

  Recursively parses variable AST, and generates PHP access code.

- commonVerifyElement (TemplateNodeEx $node, $element, &$name, array &$args)

Verifies correctness of given element.

- handleLoadBuiltin (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  Handler for `load` *built-in tag.*

- handleCommentBuiltin (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  Handler for `comment` *built-in tag.*

- runHooks ($hookPoint, array $args)

  Runs handlers associated with given hook-point.

Private Attributes

- $parserCurrentToken = null

  Currently processed token.

- $parserCurrentLine = 1

  Currently processed line (approx).

- $parserCurrentFile = null

  Currently processed template.

- $parserTokenRegexp = '~(\{\%.*?\%\})|(\{\{.*?\}\})|(\{\#.*?\#\})~u'

  Regular expression used to split template into tokens.

9.4.1 Detailed Description

Primary compiler driver.

It's responsible for creating the AST, and generating the code.

Warning, large strings and heavy recursion ahead.

Definition at line 29 of file CompilerEx.php.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 TemplateCompilerEx::__construct ()

Constructor.

Definition at line 82 of file CompilerEx.php.

9.4.3 Member Function Documentation

9.4.3.1 TemplateCompilerEx::reset ()

Resets compiler to pristine state, and loads plugins specified in `'loadPlugins'` setting.

Definition at line 102 of file CompilerEx.php.

Referenced by compile().

9.4.3.2 TemplateCompilerEx::compile (ITemplateIODriver $ *io*, $ *template*)

Compiles given template into output package.

Parameters:

← ***$io*** Used I/O driver

← ***$template*** Template name

Definition at line 117 of file CompilerEx.php.

References ITemplateIODriver::className(), createAST(), TemplateError::E_IO_LOAD_FAILURE, TemplateError::E_IO_SAVE_FAILURE, generateCode(), ITemplateIODriver::loadTemplate(), TemplateUtils::parseIODSN(), reset(), ITemplateIODriver::saveMetadata(), and ITemplateIODriver::saveTemplate().

9.4.3.3 TemplateCompilerEx::createAST (&$ *tpl*) `[private]`

Creates and returns an Abstract Syntax Tree of given template.

Parameters:

← ***$tpl*** Source template to parse

Returns:

Root node of the constructed AST

See also:

TemplateNodeEx

Definition at line 174 of file CompilerEx.php.

References parseTokenStream().

Referenced by compile().

9.4.3.4 TemplateCompilerEx::parserGetNextToken (array &$ *tokens*) `[private]`

Fetches next token from given stream, preprocesses it, and stores it in TemplateCompilerEx::$parserCurrentToken.

Parameters:

↔ ***$tokens*** Token stream (array of tokens)

Return values:

false When there is no more tokens in the stream

true Otherwise

Definition at line 191 of file CompilerEx.php.

References TemplateUtils::split(), and TemplateUtils::splitEscaped().

Referenced by parseTokenStream().

9.4.3.5 TemplateCompilerEx::parserEncounteredEndTag (TemplateNodeEx $ *node*, $ *tag*, $ *type*) [private]

Checks whether the parser has encountered ending tag with given name.

Parameters:

← ***$node*** Currently processed node

← ***$tag*** Tag name to look for ('end' is automatically prepended)

← ***$type*** Tag type - if `ignore`, then invalid ending tags will be ignored

Return values:

false When `$tag` is `null`, current token's type is not `'tag'` or parser has not encountered the tag yet

true When the tag has been found

Definition at line 235 of file CompilerEx.php.

References TemplateError::E_INVALID_SYNTAX.

Referenced by parseTokenStream().

9.4.3.6 TemplateCompilerEx::createNodeFromToken (TemplateNodeEx $ *node*) [private]

Converts current token's array into the node, and adds it to given node's children.

Parameters:

↔ ***$node*** Node to append child to

Definition at line 261 of file CompilerEx.php.

References TemplateNodeEx::addChild().

Referenced by parseTokenStream().

9.4.3.7 TemplateCompilerEx::parseTokenStream (TemplateNodeEx $ *node*, array &$ *tokens*, $ *parseUntil* = `null`, $ *blockTagType* = `null`) [private]

Parses token stream until it runs out of tokens, or when it encounters given tag name.

Parameters:

↔ ***$node*** Currently processed node, newly constructed children will be appended to it

↔ ***$tokens*** Token stream

← ***$parseUntil*** If not null, then parser will halt when it encounters tag with that name

← ***$blockTagType*** Type of the block tag (either `block` or `ignore`)

Definition at line 274 of file CompilerEx.php.

References createNodeFromToken(), parserEncounteredEndTag(), and parserGetNextToken().

Referenced by createAST().

9.4.3.8 TemplateCompilerEx::generateCode (TemplateNodeEx $ *root*) [private]

Constructs blocks and generates code for the entire AST.

Parameters:

← ***$root*** Root node of the AST

Returns:

Array of the constructed code blocks

Todo

Is it needed, or maybe TemplateCompilerEx::compile should do it?

Definition at line 317 of file CompilerEx.php.

References createBlock().

Referenced by compile().

9.4.3.9 TemplateCompilerEx::handleChildren (array &$ *children*)

Generates code from given node's children.

Accepts array instead of TemplateNodeEx for greater flexibility.

Parameters:

← ***$children*** Array to process

Returns:

Code as string

Definition at line 330 of file CompilerEx.php.

References handleNode().

Referenced by TemplateStdLibExPlugin::commonIfEqual(), TemplateStdLibExPlugin::handleTAutoEscape(), TemplateStdLibExPlugin::handleTFor(), TemplateStdLibExPlugin::handleTIf(), TemplateStdLibExPlugin::handleTIfChanged(), and TemplateStdLibExPlugin::handleTSpaceless().

9.4.3.10 TemplateCompilerEx::createBlock ($ *block*, TemplateNodeEx $ *node*)

Creates code block from raw node.

Part of exposed compiler API.

Parameters:

← ***$block*** Name of the block - in form of loop:XXX or block:XXX.

← ***$node*** Node to process

See also:

TemplateCompilerEx::handleChildren

Definition at line 345 of file CompilerEx.php.

Referenced by generateCode(), TemplateStdLibExPlugin::handleTBlock(), and TemplateStdLibExPlugin::handleTFilter().

9.4.3.11 TemplateCompilerEx::handleNode (TemplateNodeEx $ *node*)

Creates code from given node.

Parameters:

← ***$node*** Node to process

Returns:

Code as string

Todo

Maybe it should be merged with TemplateCompilerEx::handleChildren?

Definition at line 356 of file CompilerEx.php.

Referenced by handleChildren().

9.4.3.12 TemplateCompilerEx::handleTag (TemplateNodeEx $ *node*) [private]

Handles `'blockTag'` and 'inlineTag' nodes.

Calls proper tag handler (see Extending: handlers).

Parameters:

← ***$node*** Tag node to process

Returns:

Code as string

Definition at line 372 of file CompilerEx.php.

9.4.3.13 TemplateCompilerEx::handleVariable (TemplateNodeEx $ *node*) [private]

Handles `'var'` node.

Parameters:

← ***$node*** Variable node to process

Returns:

Code as string

Definition at line 401 of file CompilerEx.php.

9.4.3.14 TemplateCompilerEx::parseVariableExpression (TemplateNodeEx $ *node*, $ *variable*)

Parses variable expression, and creates runtime PHP access code.

Parameters:

← ***$node*** Source node

← ***$variable*** Variable expression to parse

Returns:

Array(access code, existence checking code)

Definition at line 423 of file CompilerEx.php.

References TemplateUtils::escape().

Referenced by TemplateStdLibExPlugin::commonIfEqual(), TemplateStdLibExPlugin::handleTCycle(), TemplateStdLibExPlugin::handleTFirstOf(), TemplateStdLibExPlugin::handleTFor(), TemplateStdLibExPlugin::handleTIfChanged(), TemplateStdLibExPlugin::handleTInclude(), TemplateStdLibExPlugin::handleTNow(), and TemplateStdLibExPlugin::parseIfExpression().

9.4.3.15 TemplateCompilerEx::generateVariableAST ($ *variable*) [private]

Generates syntax tree (using nested arrays ATM) from given variable expression.

Parameters:

← ***$variable*** Variable expression to parse

Returns:

Arrayized AST

Definition at line 442 of file CompilerEx.php.

9.4.3.16 TemplateCompilerEx::generateVariableCode (TemplateNodeEx $ *node*, &$ *ast*) [private]

Recursively parses variable AST, and generates PHP access code.

Parameters:

← ***$node*** Source node

← ***$ast*** AST to process

Returns:

Access code

Definition at line 464 of file CompilerEx.php.

9.4.3.17 TemplateCompilerEx::parseFilterChain (TemplateNodeEx $ *node*, $ *filterExpr*, $ *code*)

Handles filter chains.

Wraps given code, and returns new one.

Parameters:

← ***$node*** Filter chain source node

← ***$filterExpr*** Filter chain expression (e.g. a|b|c:d)

← ***$code*** Code to wrap in filters

Returns:

New code that uses filters

Definition at line 528 of file CompilerEx.php.

References TemplateUtils::escape(), TemplateUtils::split(), and TemplateUtils::splitEscaped().

Referenced by TemplateStdLibExPlugin::commonIfEqual(), TemplateStdLibExPlugin::handleTFilter(), TemplateStdLibExPlugin::handleTNow(), and TemplateStdLibExPlugin::parseIfExpression().

9.4.3.18 TemplateCompilerEx::commonVerifyElement (TemplateNodeEx $ *node*, $ *element*, &$ *name*, array &$ *args*) [private]

Verifies correctness of given element.

Checks whether:

- Element exists
- Element's handler is callable
- Enough arguments have been provided

Parameters:

← ***$node*** Template node representing element

← ***$element*** Element type - `'tag'` or `'filter'`

← ***$name*** Element's name

← ***$args*** Element's arguments

Returns:

Element's info array

Definition at line 580 of file CompilerEx.php.

References TemplateUtils::checkIfAllowed(), TemplateError::E_INVALID_HANDLER, TemplateError::E_INVALID_SYNTAX, TemplateError::E_UNKNOWN_FILTER, TemplateError::E_-UNKNOWN_TAG, TemplateUtils::panic(), and TemplateUtils::strip().

9.4.3.19 TemplateCompilerEx::handleLoadBuiltin (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*) [private]

Handler for `load` built-in tag.

For handlers reference, see Extending: handlers. For standard and built-in tags reference, see Standard tags.

Parameters:

← ***$compiler*** Compiler handle (TemplateCompilerEx instance)

← ***$node*** Node handle (TemplateNodeEx instance, see Extending: AST nodes)

← ***$tag*** Tag name (as string)

← ***$args*** Tag arguments (as array)

Definition at line 628 of file CompilerEx.php.

References TemplateError::E_UNKNOWN_PLUGIN.

9.4.3.20 TemplateCompilerEx::handleCommentBuiltin (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*) [private]

Handler for `comment` built-in tag.

For handlers reference, see Extending: handlers. For standard and built-in tags reference, see Standard tags.

Parameters:

← ***$compiler*** Compiler handle (TemplateCompilerEx instance)

← ***$node*** Node handle (TemplateNodeEx instance, see Extending: AST nodes)

← ***$tag*** Tag name (as string)

← ***$args*** Tag arguments (as array)

Definition at line 649 of file CompilerEx.php.

9.4.3.21 TemplateCompilerEx::raise (TemplateNodeEx $ *node*, $ *message*, $ *code*)

Raises an error, appending "(in template <file> somewhere around line <line>)" using given node.

Parameters:

← ***$node*** Node producing an error

← ***$message*** Error message

← ***$code*** Error code

See also:

TemplateError

Definition at line 665 of file CompilerEx.php.

Referenced by TemplateStdLibExPlugin::handleTFor().

9.4.3.22 TemplateCompilerEx::raiseIf ($ *cond*, TemplateNodeEx $ *node*, $ *message*, $ *code*)

Shorthand for conditional call to TemplateCompilerEx::raise.

Won't raise if `$cond` is `false`.

Parameters:

← ***$cond*** Condition value as boolean

← ***$node*** Node producing an error

← ***$message*** Error message

← ***$code*** Error code

See also:

TemplateError

Definition at line 682 of file CompilerEx.php.

Referenced by TemplateStdLibExPlugin::commonIfEqual(), TemplateStdLibExPlugin::handleTAutoEscape(), TemplateStdLibExPlugin::handleTBlock(), TemplateStdLibExPlugin::handleTCycle(), TemplateStdLibExPlugin::handleTExtends(), TemplateStdLibExPlugin::handleTFirstOf(), TemplateStdLibExPlugin::handleTFor(), TemplateStdLibExPlugin::handleTIfChanged(), TemplateStdLibExPlugin::handleTTemplateTag(), TemplateStdLibExPlugin::parseIfExpression(), and TemplateStdLibExPlugin::parseIfExpressionCheckParens().

9.4.3.23 TemplateCompilerEx::findAlternativeBranch (TemplateNodeEx $ *node*, $ *tag*)

Used to find and isolate alternative branch of given node, starting with given inline tag.

Can be used wherever alternative (e.g. `{% else %}`) branch cannot be implemented by simply inlining some code (e.g. `} else {`).

Parameters:

← ***$node*** Current node

← ***$tag*** Tag starting alternative branch

Returns:

Array(main nodes, alternative branch nodes)

Definition at line 695 of file CompilerEx.php.

Referenced by TemplateStdLibExPlugin::handleTFor(), and TemplateStdLibExPlugin::handleTIfChanged().

9.4.3.24 TemplateCompilerEx::generateUniqueBlock ($ *idPrefix*, $ *blockPrefix* = `'custom:'`, $ *keyLength* = `5`)

Generates prefixed block name that is guaranteed to be unique in current template.

By default generates 5-character unique key using combination of `uniqid`, `mt_rand` and `md5`. Beware: potential infinite loop - if key length is too small, then key space might be exhausted, which will lead to infinite loop in this function. Increase `$keyLength` if your usage could lead to this condition.

Parameters:

← ***$idPrefix*** `uniqid` prefix

← ***$blockPrefix*** Unique key will be prefixed with this. Default: `custom:`.

← ***$keyLength*** Length of generated key. Must be lower than 32. Default: 5.

Definition at line 729 of file CompilerEx.php.

References TemplateUtils::panic().

Referenced by TemplateStdLibExPlugin::handleTFor(), and TemplateStdLibExPlugin::handleTIfChanged().

9.4.3.25 TemplateCompilerEx::runHooks ($ *hookPoint*, array $ *args*) `[private]`

Runs handlers associated with given hook-point.

Every handler might return `true` boolean value, to break the chain and finish hook execution.

Parameters:

← ***$hookPoint*** Hook-point to execute

← ***$args*** Array of hook arguments

Return values:

true Some handler has broken the chain

false All handlers have been executed, and none has broken the chain

Definition at line 756 of file CompilerEx.php.

References TemplateError::E_INVALID_HANDLER, and TemplateUtils::strip().

9.4.4 Member Data Documentation

9.4.4.1 TemplateCompilerEx::$settings = null

Current settings (reference to TemplateEnviron::$settings).

Definition at line 33 of file CompilerEx.php.

9.4.4.2 TemplateCompilerEx::$plugins = null

Registry of available plugins.

TemplatePlugins instance.

Definition at line 39 of file CompilerEx.php.

9.4.4.3 TemplateCompilerEx::$loadedPlugins = array()

Registry of loaded plugins (per-template).

Definition at line 43 of file CompilerEx.php.

9.4.4.4 TemplateCompilerEx::$parserCurrentToken = null [private]

Currently processed token.

Assoc. array containing two keys - 'type' and 'content'.

Definition at line 50 of file CompilerEx.php.

9.4.4.5 TemplateCompilerEx::$parserCurrentLine = 1 [private]

Currently processed line (approx).

Definition at line 54 of file CompilerEx.php.

9.4.4.6 TemplateCompilerEx::$parserCurrentFile = null [private]

Currently processed template.

Definition at line 58 of file CompilerEx.php.

9.4.4.7 TemplateCompilerEx::$parserTokenRegexp = '~(\{\%.*?\%\})|(\{\{.*?\}\})|(\{\#.*?\#\})~u' [private]

Regular expression used to split template into tokens.

Definition at line 62 of file CompilerEx.php.

9.4.4.8 TemplateCompilerEx::$blocks = array()

Already constructed code blocks.

Definition at line 68 of file CompilerEx.php.

9.4.4.9 TemplateCompilerEx::$metadata = array()

Template's metadata.

Definition at line 72 of file CompilerEx.php.

9.4.4.10 TemplateCompilerEx::$className = null

Template's classname.

Definition at line 76 of file CompilerEx.php.

The documentation for this class was generated from the following file:

- CompilerEx.php

9.5 TemplateEnviron Class Reference

Template environment - library's end-user API.

Public Member Functions

- __construct (array $settings=array())

  Constructor.

- compile (ITemplateIODriver $io, $template)

  Compile given template.

- include_ ($template, $mode=null, $returnMeta=false)

  Includes template's code into global namespace via I/O driver given in DSN.

- get ($template, $mode=null)

  Returns template instance.

- getMeta ($template, $mode=null)

  Returns user-defined template metadata.

- cachedGet ($template, $mode=null)

  Cached version of TemplateEnviron::get.

- render ($template, array $context, $mode=null)

  Render the template directly.

Static Public Member Functions

- static createFromINI ($settingsINI)

  Named constructor.

Public Attributes

- const RECOMPILE_ALWAYS = 1

  One of recompilation modes - always recompile.

- const RECOMPILE_IF_CHANGED = 0

  One of recompilation modes - recompile only when necessary (default).

- const RECOMPILE_NEVER = -1

  One of recompilation modes - never recompile (a.k.a.

- const SECURITY_DISABLE = 0

  One of security modes - do not test against the lists.

- const SECURITY_ALLOW_ALL = 1

  One of security modes - first allow all, then check 'disallowed' list.

- const SECURITY_ALLOW_DENY = 2

  One of security modes - first check 'allowed' list, then 'disallowed'.

- const SECURITY_DENY_ALLOW = 3

  One of security modes - first check 'disallowed' list, then 'allowed'.

- const SECURITY_DENY_ALL = 4

  One of security modes - first disallow all, then check 'allowed' list.

- const SECURITY_MATCH_EVERYTHING = true

  May be used instead of `allowed` *or* `disallowed` *list, as a wildcard matching everything.*

- const LOAD_ALL_PLUGINS = true

  May be used as `loadPlugins` *setting to always load all available plugins on all search paths.*

- $settings

  Default environment settings.

- $templateCache = array()

  Internal template objects cache.

- $compiler = null

  Compiler instance.

9.5.1 Detailed Description

Template environment - library's end-user API.

Examples:

00_hello.php, 01_io.php, 02_settings.php, 03_context.php, 08_security.php, and 09_errors.php.

Definition at line 198 of file Environment.php.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 TemplateEnviron::__construct (array $ *settings* = `array()`)

Constructor.

Optionally sets up initial settings.

Parameters:

← ***$settings*** Settings array

See also:

TemplateEnviron::$settings

Definition at line 412 of file Environment.php.

References $settings.

9.5.3 Member Function Documentation

9.5.3.1 static TemplateEnviron::createFromINI ($ *settingsINI*) `[static]`

Named constructor.

Shorthand for INI parsing.

Parameters:

← ***$settingsINI*** Settings INI filename

Returns:

TemplateEnviron instance

See also:

TemplateEnviron::__construct

Examples:

02_settings.php.

Definition at line 426 of file Environment.php.

9.5.3.2 TemplateEnviron::compile (ITemplateIODriver $ *io*, $ *template*)

Compile given template.

This doesn't check `recompilationMode` setting, so it may be used to forcibly recompile template. It doesn't use DSN - you must provide correct I/O driver object.

Parameters:

← ***$io*** I/O driver

← ***$template*** Template name

Definition at line 438 of file Environment.php.

Referenced by include_().

9.5.3.3 TemplateEnviron::include_ ($ *template*, $ *mode* = `null`, $ *returnMeta* = `false`)

Includes template's code into global namespace via I/O driver given in DSN.

I/O system checks `recompilationMode` (which you may override per-template using `$mode` parameter) and acts accordingly, recompiling only when it's required either by this setting or template change. Also recursively handles inclusion of template parent.

Parameters:

← ***$template*** Template name
← ***$mode*** Per-template recompilation mode override (optional)
← ***$returnMeta*** If `true`, then returns metadata instead of including the code

Returns:

Template's class name, as string; or metadata, as array

Definition at line 458 of file Environment.php.

References compile(), and TemplateUtils::parseIODSN().

Referenced by get(), and getMeta().

9.5.3.4 TemplateEnviron::get ($ *template*, $ *mode* = `null`)

Returns template instance.

Keep in mind that it doesn't cache template objects - every call will result in object construction, which may lead to performance loss. If you want to use internal object cache, use TemplateEnviron::cachedGet.

Parameters:

← ***$template*** Template ID
← ***$mode*** Per-template recompilation mode override (optional)

Returns:

Template subclass instance - an template object

See also:

TemplateEnviron::include_
TemplateEnviron::cachedGet

Definition at line 487 of file Environment.php.

References include_().

9.5.3.5 TemplateEnviron::getMeta ($ *template*, $ *mode* = `null`)

Returns user-defined template metadata.

It will trigger the compilation, if neccessary. Keep in mind, that the core doesn't cache the metadata. Every call will result in I/O, array unserialization and array filtering.

Parameters:

← ***$template*** Template ID
← ***$mode*** Per-template recompilation mode override (optional)

Returns:

Metadata array

Definition at line 501 of file Environment.php.

References include_().

9.5.3.6 TemplateEnviron::cachedGet ($ *template*, $ *mode* = `null`)

Cached version of TemplateEnviron::get.

Parameters:

← ***$template*** Template ID

← ***$mode*** Per-template recompilation mode override (optional)

Returns:

Template subclass instance - an template object

See also:

TemplateEnviron::$templateCache

Definition at line 522 of file Environment.php.

Referenced by render().

9.5.3.7 TemplateEnviron::render ($ *template*, array $ *context*, $ *mode* = `null`)

Render the template directly.

Uses internal cache.

Parameters:

← ***$template*** Template ID

← ***$context*** Context array

← ***$mode*** Per-template recompilation mode override (optional)

See also:

Template::render
TemplateEnviron::cachedGet

Definition at line 538 of file Environment.php.

References cachedGet().

9.5.4 Member Data Documentation

9.5.4.1 const TemplateEnviron::RECOMPILE_ALWAYS = 1

One of recompilation modes - always recompile.

Examples:

02_settings.php.

Definition at line 202 of file Environment.php.

Referenced by TemplateStringIO::upToDate(), and TemplateFileIO::upToDate().

9.5.4.2 const TemplateEnviron::RECOMPILE_IF_CHANGED = 0

One of recompilation modes - recompile only when necessary (default).

Definition at line 206 of file Environment.php.

Referenced by TemplateFileIO::upToDate().

9.5.4.3 const TemplateEnviron::RECOMPILE_NEVER = -1

One of recompilation modes - never recompile (a.k.a.

performance mode).

Definition at line 210 of file Environment.php.

Referenced by TemplateFileIO::upToDate().

9.5.4.4 const TemplateEnviron::SECURITY_DISABLE = 0

One of security modes - do not test against the lists.

Definition at line 215 of file Environment.php.

Referenced by TemplateUtils::checkIfAllowed().

9.5.4.5 const TemplateEnviron::SECURITY_ALLOW_ALL = 1

One of security modes - first allow all, then check 'disallowed' list.

Definition at line 219 of file Environment.php.

Referenced by TemplateUtils::checkIfAllowed().

9.5.4.6 const TemplateEnviron::SECURITY_ALLOW_DENY = 2

One of security modes - first check 'allowed' list, then 'disallowed'.

Definition at line 223 of file Environment.php.

Referenced by TemplateUtils::checkIfAllowed().

9.5.4.7 const TemplateEnviron::SECURITY_DENY_ALLOW = 3

One of security modes - first check 'disallowed' list, then 'allowed'.

Definition at line 227 of file Environment.php.

Referenced by TemplateUtils::checkIfAllowed().

9.5.4.8 const TemplateEnviron::SECURITY_DENY_ALL = 4

One of security modes - first disallow all, then check 'allowed' list.

Examples:

08_security.php.

Definition at line 231 of file Environment.php.

Referenced by TemplateUtils::checkIfAllowed().

9.5.4.9 const TemplateEnviron::SECURITY_MATCH_EVERYTHING = true

May be used instead of `allowed` or `disallowed` list, as a wildcard matching everything.

Implemented for greater flexibility than only hardcoded modes specified above.

Definition at line 238 of file Environment.php.

9.5.4.10 const TemplateEnviron::LOAD_ALL_PLUGINS = true

May be used as `loadPlugins` setting to always load all available plugins on all search paths.

Definition at line 244 of file Environment.php.

9.5.4.11 TemplateEnviron::$settings

Default environment settings.

Available settings are:

- `inputPrefix` (string) - will be prefixed to all input filenames. Interpretation is up to the I/O driver. In bundled `'file'` I/O: source directory name. In bundled `'string'` I/O: not used. By default it's `'./templates/'`.
- `outputPrefix` (string) - will be prefixed to all output filenames. Interpretation is up to the I/O driver. In both bundled I/O drivers (`'file'` and `'string'`): output directory name. By default it's `'./templates_c/'`.
- `loadPlugins` (array) - if it's an array: list of plugins to load when compilation starts. Plugins are not loaded until compilation is required.
- `loadPlugins` (bool) - if it's a boolean (value is not checked, but you should use TemplateEnviron::LOAD_ALL_PLUGINS for better self-documentation): library will gather and load all plugins, on every path given in `pluginsPaths`. It's also a default behaviour.
- `pluginsPaths` (array) - plugins' search paths. When plugin is loaded, all paths given in this array are searched for plugin's file. See Extending SithTemplate for more information about plugins.
- `useDefaultPluginsPath` (bool) - determines whether default plugins' search path (i.e. `SITHTEMPLATE_DIR/plugins/`) should be used. Note that it works only on construction (by appending to `pluginsPaths`) - if you override `pluginsPaths` later, this setting won't have any effect. By default it's `true`.
- `recompilationMode` (int) - controls how recompilation is handled. One of TemplateEnviron::RECOMPILE_ALWAYS (templates are recompiled on each request), TemplateEnviron::RECOMPILE_IF_CHANGED (templates are recompiled when modified), or TemplateEnviron::RECOMPILE_NEVER (templates are compiled once and never recompiled). By default it's `RECOMPILE_IF_CHANGED`.
- `defaultIODriver` (string) - default I/O driver to use. Note that you must register it using TemplateIO::register before you request any template.

- `autoEscape` (bool) - should variables not marked with pseudofilter `safe` be automatically escaped, using `StdLibEx` filter `escape`? Do not enable, if you do not use `StdLibEx`. Disabled by default.
- `allowInternalRequest` (bool) - should access to super-globals be allowed through `{{ internal.request }}`? Enabled by default.
- `allowInternalConstants` (bool) - should access to global constants be allowed through `{{ internal.const }}`? Enabled by default.
- `restrictIncludeIO` (bool) - should all {% include %} calls be restricted to the same I/O driver used in TemplateEnviron::get or TemplateEnviron::cachedGet? Disabled by default. Note that it is only enforced at runtime.
- `restrictExtendIO` (bool) - should all {% extend %} calls be restricted to the same I/O driver used in TemplateEnviron::get or TemplateEnviron::cachedGet? Disabled by default. Note that it is only enforced at compile time.
- `securityEvalMode` (int) - specifies whether and how all plugins, tags, filters and function calls should be tested against security (allow/disallow) lists. One of TemplateEnviron::SECURITY_DISABLE, TemplateEnviron::SECURITY_ALLOW_ALL, TemplateEnviron::SECURITY_ALLOW_DENY, TemplateEnviron::SECURITY_DENY_ALLOW, TemplateEnviron::SECURITY_DENY_ALL. Default is `SECURITY_DISABLE`.
- `allowedPlugins` (array) - whitelist of entire plugins. You can also use TemplateEnviron::SECURITY_MATCH_EVERYTHING.
- `disallowedPlugins` (array) - blacklist of entire plugins. You can also use TemplateEnviron::SECURITY_MATCH_EVERYTHING.
- `allowedTags` (array) - whitelist of single tags. You can also use TemplateEnviron::SECURITY_MATCH_EVERYTHING.
- `disallowedTags` (array) - blacklist of single tags. You can also use TemplateEnviron::SECURITY_MATCH_EVERYTHING.
- `allowedFilters` (array) - whitelist of single filters. You can also use TemplateEnviron::SECURITY_MATCH_EVERYTHING.
- `disallowedFilters` (array) - blacklist of single filters. You can also use TemplateEnviron::SECURITY_MATCH_EVERYTHING.
- `allowedFunctions` (array) - whitelist of single functions (used in `{% call %}`). You can also use TemplateEnviron::SECURITY_MATCH_EVERYTHING. Note that these are the only lists that have effect not only during compilation, but also on runtime.
- `disallowedFunctions` (array) - blacklist of single functions (used in `{% call %}`). You can also use TemplateEnviron::SECURITY_MATCH_EVERYTHING. Note that these are the only lists that have effect not only during compilation, but also on runtime.

Definition at line 363 of file Environment.php.

Referenced by __construct().

9.5.4.12 TemplateEnviron::$templateCache = array()

Internal template objects cache.

Definition at line 399 of file Environment.php.

9.5.4.13 TemplateEnviron::$compiler = null

Compiler instance.

Created when compilation is required.

Definition at line 404 of file Environment.php.

The documentation for this class was generated from the following file:

- Environment.php

9.6 TemplateError Class Reference

Main and currently the only exception type thrown by SithTemplate internals.

Public Attributes

- const E_UNKNOWN_ERROR = 0x00000000

  An unknown error.

- const E_INVALID_VAR = 0x00000001

  An invalid variable error.

- const E_IO_LOAD_FAILURE = 0x00000002

  An I/O read error.

- const E_IO_SAVE_FAILURE = 0x00000003

  An I/O save error.

- const E_UNKNOWN_TAG = 0x00000004

  An unknown tag error.

- const E_UNKNOWN_FILTER = 0x00000005

  An unknown filter error.

- const E_INVALID_HANDLER = 0x00000006

  An invalid handler error.

- const E_INVALID_SYNTAX = 0x00000007

  An invalid syntax error.

- const E_UNKNOWN_PLUGIN = 0x00000008

  An unknown plugin error.

- const E_INVALID_PLUGIN = 0x00000009

  An invalid plugin error.

- const E_INVALID_ARGUMENT = 0x0000000A

  An invalid argument error.

- const E_SECURITY_VIOLATION = 0x0000000B

  A security violation.

- const E_INTERNAL_CORE_FAILURE = 0xFFFFFFFF

  A core panic.

9.6.1 Detailed Description

Main and currently the only exception type thrown by SithTemplate internals.

Since 1.1, preformatted messages are no longer used, full message is constructed wherever exception is thrown instead.

Examples:

09_errors.php.

Definition at line 15 of file Error.php.

9.6.2 Member Data Documentation

9.6.2.1 const TemplateError::E_UNKNOWN_ERROR = 0x00000000

An unknown error.

Exception with this code indicates a mistake in code, and should be reported as bug.

Definition at line 20 of file Error.php.

9.6.2.2 const TemplateError::E_INVALID_VAR = 0x00000001

An invalid variable error.

Thrown when variable fails constraint test (e.g. non-iterable used as argument in `{% for %}`). Used in runtime only.

Definition at line 26 of file Error.php.

Referenced by Template::invalidVar().

9.6.2.3 const TemplateError::E_IO_LOAD_FAILURE = 0x00000002

An I/O read error.

Thrown if template DSN cannot be resolved (e.g. template doesn't exist, or cannot be read; or its metadata; or compiled code). Used in both compile time and runtime.

Definition at line 32 of file Error.php.

Referenced by TemplateCompilerEx::compile(), TemplateStringIO::includeCode(), TemplateFileIO::includeCode(), and TemplateFileIO::upToDate().

9.6.2.4 const TemplateError::E_IO_SAVE_FAILURE = 0x00000003

An I/O save error.

Thrown if template code or metadata cannot be saved. Used in compile time only.

Definition at line 37 of file Error.php.

Referenced by TemplateCompilerEx::compile().

9.6.2.5 const TemplateError::E_UNKNOWN_TAG = 0x00000004

An unknown tag error.

Thrown if unknown tag is encountered in template source. Used in compile time only.

Definition at line 42 of file Error.php.

Referenced by TemplateCompilerEx::commonVerifyElement().

9.6.2.6 const TemplateError::E_UNKNOWN_FILTER = 0x00000005

An unknown filter error.

Thrown if unknown filter is encountered in template source. Used in compile time only.

Definition at line 47 of file Error.php.

Referenced by TemplateCompilerEx::commonVerifyElement().

9.6.2.7 const TemplateError::E_INVALID_HANDLER = 0x00000006

An invalid handler error.

This indicates a bug in the plugin you use. Don't report it, unless this plugin is `StdLibEx`, which implements SithTemplate's standard library. Used in compile time only.

Definition at line 53 of file Error.php.

Referenced by TemplateCompilerEx::commonVerifyElement(), TemplatePlugins::register(), and TemplateCompilerEx::runHooks().

9.6.2.8 const TemplateError::E_INVALID_SYNTAX = 0x00000007

An invalid syntax error.

Thrown when compiler or tag/filter detects an syntax error, which doesn't have it's own error code. Used in compile time only.

Definition at line 58 of file Error.php.

Referenced by TemplateCompilerEx::commonVerifyElement(), TemplateStdLibExPlugin::handleTCycle(), TemplateStdLibExPlugin::handleTExtends(), and TemplateCompilerEx::parserEncounteredEndTag().

9.6.2.9 const TemplateError::E_UNKNOWN_PLUGIN = 0x00000008

An unknown plugin error.

In compile time: thrown when library tries to load non-existant plugin. In runtime: thrown when library tries to use non-existant I/O driver.

Definition at line 63 of file Error.php.

Referenced by TemplatePlugins::findPlugins(), TemplateIO::get(), and TemplateCompilerEx::handleLoadBuiltin().

9.6.2.10 const TemplateError::E_INVALID_PLUGIN = 0x00000009

An invalid plugin error.

It indicates a bug in the plugin or I/O driver you use. Don't report it, unless it's related to `StdLibEx` plugin, or `file` or `string` I/O drivers. Used in both runtime and compile time.

Definition at line 69 of file Error.php.

Referenced by TemplateIO::get(), and TemplatePlugins::load().

9.6.2.11 const TemplateError::E_INVALID_ARGUMENT = 0x0000000A

An invalid argument error.

Thrown when a function, or tag/filter gets called with invalid arguments. Used in both runtime and compile time.

Definition at line 74 of file Error.php.

Referenced by TemplateStdLibExPlugin::commonIfEqual(), TemplateStdLibExPlugin::handleTAutoEscape(), TemplateStdLibExPlugin::handleTBlock(), TemplateStdLibExPlugin::handleTCycle(), TemplateStdLibExPlugin::handleTFirstOf(), TemplateStdLibExPlugin::handleTFor(), TemplateStdLibExPlugin::handleTIfChanged(), TemplateStdLibExPlugin::handleTTemplateTag(), TemplateStdLibExPlugin::parseIfExpression(), TemplateStdLibExPlugin::parseIfExpressionCheckParens(), TemplateIO::register(), and TemplateFileIO::upToDate().

9.6.2.12 const TemplateError::E_SECURITY_VIOLATION = 0x0000000B

A security violation.

Thrown when library encounters situation forbidden by the security settings.

Definition at line 79 of file Error.php.

Referenced by TemplateUtils::checkIfAllowed(), and TemplateUtils::checkIORestriction().

9.6.2.13 const TemplateError::E_INTERNAL_CORE_FAILURE = 0xFFFFFFFF

A core panic.

If you get exception with this code, report it - it's a bug.

Definition at line 84 of file Error.php.

Referenced by TemplateUtils::panic().

The documentation for this class was generated from the following file:

- Error.php

9.7 TemplateFileIO Class Reference

File I/O implementation.

Public Member Functions

- upToDate (array &$settings, &$template, $mode)
- includeCode (array &$settings, &$template)
- className (array &$settings, &$template)
- loadTemplate (array &$settings, &$template)
- loadMetadata (array &$settings, &$template)
- saveTemplate (array &$settings, &$template, &$code)
- saveMetadata (array &$settings, &$template, array &$metadata)

Protected Member Functions

- pfn (array &$settings, &$template)

9.7.1 Detailed Description

File I/O implementation.

Definition at line 78 of file IO.php.

9.7.2 Member Function Documentation

9.7.2.1 TemplateFileIO::pfn (array &$ *settings*, &$ *template*) [protected]

Reimplemented in TemplateStringIO.

Definition at line 84 of file IO.php.

Referenced by includeCode(), loadMetadata(), loadTemplate(), saveMetadata(), saveTemplate(), and upToDate().

9.7.2.2 TemplateFileIO::upToDate (array &$ *settings*, &$ *template*, $ *mode*)

Should check whether given template is up-to-date.

If driver uses `recompilationMode` setting, then it should use supplied `$mode` argument instead, to allow per-template mode override. Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

← ***$mode*** Recompilation mode

Return values:

true Template is up-to-date - no (re)compilation is needed

false Template must be (re)compiled

Implements ITemplateIODriver.

Reimplemented in TemplateStringIO.

Definition at line 93 of file IO.php.

References TemplateError::E_INVALID_ARGUMENT, TemplateError::E_IO_LOAD_FAILURE, pfn(), TemplateEnviron::RECOMPILE_ALWAYS, TemplateEnviron::RECOMPILE_IF_CHANGED, and TemplateEnviron::RECOMPILE_NEVER.

9.7.2.3 TemplateFileIO::includeCode (array &$ *settings*, &$ *template*)

Should include template's code into global namespace.

It must ensure that no code redefinition will happen. Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Included class name

Implements ITemplateIODriver.

Reimplemented in TemplateStringIO.

Definition at line 122 of file IO.php.

References className(), TemplateError::E_IO_LOAD_FAILURE, and pfn().

9.7.2.4 TemplateFileIO::className (array &$ *settings*, &$ *template*)

Should return template's classname.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Class name

Implements ITemplateIODriver.

Reimplemented in TemplateStringIO.

Definition at line 140 of file IO.php.

Referenced by includeCode().

9.7.2.5 TemplateFileIO::loadTemplate (array &$ *settings*, &$ *template*)

Should read template source code as whole, and return it.

SithTemplate ensures that this will be called only when compilation is needed, so no additional checks are needed. Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Whole template source

Implements ITemplateIODriver.

Definition at line 145 of file IO.php.

References pfn().

9.7.2.6 TemplateFileIO::loadMetadata (array &$ *settings*, &$ *template*)

Should read template's metadata, and return it.

Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Template metadata or `false`.

Implements ITemplateIODriver.

Definition at line 151 of file IO.php.

References pfn().

9.7.2.7 TemplateFileIO::saveTemplate (array &$ *settings*, &$ *template*, &$ *code*)

Should save compiled template code.

Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

← ***$code*** Template code

Return values:

true Template has been saved

false An error occured

Implements ITemplateIODriver.

Definition at line 157 of file IO.php.

References pfn().

9.7.2.8 TemplateFileIO::saveMetadata (array &$ *settings*, &$ *template*, array &$ *metadata*)

Should save template metadata.

Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

← ***$metadata*** Metadata

Return values:

true Metadata has been saved

false An error occured

Implements ITemplateIODriver.

Definition at line 163 of file IO.php.

References pfn().

The documentation for this class was generated from the following file:

- IO.php

9.8 TemplateIO Class Reference

Global I/O driver storage.

Static Public Member Functions

- static get ($driver)

  Returns (creates if necessary) an I/O driver instance.

- static register ($driver, $className)

  Registers new I/O driver.

Static Private Attributes

- static $ioDrivers

  I/O driver registry.

9.8.1 Detailed Description

Global I/O driver storage.

Definition at line 13 of file IO.php.

9.8.2 Member Function Documentation

9.8.2.1 static TemplateIO::get ($ *driver*) [static]

Returns (creates if necessary) an I/O driver instance.

Parameters:

← ***$driver*** I/O driver name

Returns:

`ITemplateIODriver` implementation

Definition at line 29 of file IO.php.

References TemplateError::E_INVALID_PLUGIN, and TemplateError::E_UNKNOWN_PLUGIN.

Referenced by TemplateUtils::parseIODSN().

9.8.2.2 static TemplateIO::register ($ *driver*, $ *className*) [static]

Registers new I/O driver.

New driver may override existing one.

Parameters:

← ***$driver*** I/O driver name

← ***$className*** I/O driver class

Definition at line 67 of file IO.php.

References TemplateError::E_INVALID_ARGUMENT.

9.8.3 Member Data Documentation

9.8.3.1 TemplateIO::$ioDrivers [static, private]

Initial value:

```
array(

 'file'     => array('TemplateFileIO',     null),
 'string'   => array('TemplateStringIO',   null),
)
```

I/O driver registry.

Definition at line 17 of file IO.php.

The documentation for this class was generated from the following file:

- IO.php

9.9 TemplateNodeEx Class Reference

Class-container for AST nodes.

Public Member Functions

- __construct ($id, $parent=null, $content=null, $file=null, $line=0)

  Constructor.

- addChild ($id, $content=null, $file=null, $line=0)

  Creates a new `TemplateNodeEx` *instance and adds it to this node children.*

- dump ($level=0)

  Debugging aid.

Public Attributes

- $nodeID = ''

  Type of this node.

- $nodeParent = null

  Parent of this node.

- $nodeChildren = array()

  Children of this node.

- $nodeContent = null

  Content of this node.

- $nodeLine = 0

  Aid for template debugging, source line where the parser constructed this node.

- $nodeFile = null

  Aid for template debugging, source file where the parser constructed this node.

9.9.1 Detailed Description

Class-container for AST nodes.

Contains node ID (it's type), references to its parent and children, preprocessed content, and source template line where it has been found (although it may not be very accurate).

Definition at line 784 of file CompilerEx.php.

9.9.2 Constructor & Destructor Documentation

9.9.2.1 TemplateNodeEx::__construct ($ *id*, $ *parent* = `null`, $ *content* = `null`, $ *file* = `null`, $ *line* = `0`)

Constructor.

Parameters:

$id Type of this node

$parent Parent of this node

$content Content of this node

$file Source template file

$line Source template line

Definition at line 822 of file CompilerEx.php.

9.9.3 Member Function Documentation

9.9.3.1 TemplateNodeEx::addChild ($ *id*, $ *content* = `null`, $ *file* = `null`, $ *line* = `0`)

Creates a new `TemplateNodeEx` instance and adds it to this node children.

Parameters:

$id Type of new node

$content Content of new node

$file Source template file

$line Source template line

Definition at line 838 of file CompilerEx.php.

Referenced by TemplateCompilerEx::createNodeFromToken().

9.9.3.2 TemplateNodeEx::dump ($ *level* = `0`)

Debugging aid.

Dumps the AST and its children.

Parameters:

$level Current indentation level

Returns:

The plaintext dump of current level and levels below

Definition at line 848 of file CompilerEx.php.

9.9.4 Member Data Documentation

9.9.4.1 TemplateNodeEx::$nodeID = ”

Type of this node.

It may be `'text'` (for plaintext nodes), `'var'` (for variable nodes), `'inlineTag'` or `'blockTag'` (for inline and block tag nodes).

Definition at line 790 of file CompilerEx.php.

9.9.4.2 TemplateNodeEx::$nodeParent = null

Parent of this node.

Definition at line 794 of file CompilerEx.php.

9.9.4.3 TemplateNodeEx::$nodeChildren = array()

Children of this node.

Definition at line 798 of file CompilerEx.php.

9.9.4.4 TemplateNodeEx::$nodeContent = null

Content of this node.

It may be plain text (for text and var nodes), or an array (for tag nodes - [0] will be the tag’s name and [1] the array of its arguments).

Definition at line 803 of file CompilerEx.php.

9.9.4.5 TemplateNodeEx::$nodeLine = 0

Aid for template debugging, source line where the parser constructed this node.

Definition at line 807 of file CompilerEx.php.

9.9.4.6 TemplateNodeEx::$nodeFile = null

Aid for template debugging, source file where the parser constructed this node.

Definition at line 811 of file CompilerEx.php.

The documentation for this class was generated from the following file:

- CompilerEx.php

9.10 TemplatePlugins Class Reference

Handles discovery, registration and utilization of plugins.

Public Member Functions

- __construct (array $builtins)
  Constructor.
- load (TemplateCompilerEx $compiler, $node, $plugin, $pluginFile=null, $noThrow=false)
  Load single plugin.
- loadMultiple (TemplateCompilerEx $compiler, $node, $plugins)
  Load multiple plugins.
- known ($type, $name)
  Check whether given element is registered.
- & get ($type, $name)
  Returns handler(s) information for given element.

Public Attributes

- $searchPaths = array()
  Plugins' search paths.

Private Member Functions

- findPlugin ($plugin)
  Looks for plugin file on all search paths.
- findPlugins ($plugins)
  Looks for multiple plugins' files on all search paths.
- register ($plugin, ITemplatePlugin $pluginObj, $type, array &$handlers)
  Registers given elements.
- registerHooks ($plugin, ITemplatePlugin $pluginObj, array &$hooks)
  Register given hooks.

Private Attributes

- $plugins = array()
  Already loaded plugins.

- $elements = array()

  Elements registry.

9.10.1 Detailed Description

Handles discovery, registration and utilization of plugins.

Definition at line 13 of file Plugins.php.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 TemplatePlugins::__construct (array $ *builtins*)

Constructor.

Also handles registration of built-ins.

Parameters:

← ***$builtins*** Built-in elements to register

Definition at line 32 of file Plugins.php.

9.10.3 Member Function Documentation

9.10.3.1 TemplatePlugins::load (TemplateCompilerEx $ *compiler*, $ *node*, $ *plugin*, $ *pluginFile* = `null`, $ *noThrow* = `false`)

Load single plugin.

Parameters:

← ***$compiler*** TemplateCompilerEx instance

← ***$node*** Optional TemplateNodeEx instance

← ***$plugin*** Plugin name

← ***$pluginFile*** Plugin's full filename (optional)

← ***$noThrow*** If `true`, no exception will be thrown on error (boolean `false` will be returned)

Return values:

true Plugin has been loaded

false Plugin is invalid/non-existant

Definition at line 47 of file Plugins.php.

References TemplateUtils::checkIfAllowed(), TemplateUtils::doesImplement(), TemplateError::E_-INVALID_PLUGIN, findPlugin(), and registerHooks().

Referenced by loadMultiple().

9.10.3.2 TemplatePlugins::loadMultiple (TemplateCompilerEx $ *compiler*, $ *node*, $ *plugins*)

Load multiple plugins.

Parameters:

← ***$compiler*** TemplateCompilerEx instance

← ***$node*** Optional TemplateNodeEx instance

← ***$plugins*** Array (plugins to look for) or boolean ('load all plugins' mode)

Definition at line 90 of file Plugins.php.

References $plugins, findPlugins(), and load().

9.10.3.3 TemplatePlugins::known ($ *type*, $ *name*)

Check whether given element is registered.

Parameters:

← ***$type*** Element type (`'tag'`, `'filter'`, `'hook'` or `'plugin'`)

← ***$name*** Element name

Return values:

true Element is registered

false Element is not registered

Definition at line 104 of file Plugins.php.

9.10.3.4 & TemplatePlugins::get ($ *type*, $ *name*)

Returns handler(s) information for given element.

Doesn't check for element's existence.

Parameters:

← ***$type*** Element type (`'tag'`, `'filter'` or `'hook'`)

← ***$name*** Element name

Returns:

Assoc. array

Definition at line 117 of file Plugins.php.

9.10.3.5 TemplatePlugins::findPlugin ($ *plugin*) `[private]`

Looks for plugin file on all search paths.

Parameters:

← ***$plugin*** Plugin name (as string)

Returns:

Filename or `false`

See also:

TemplatePlugins::load

Definition at line 128 of file Plugins.php.

Referenced by load().

9.10.3.6 TemplatePlugins::findPlugins ($ *plugins*) `[private]`

Looks for multiple plugins' files on all search paths.

Parameters:

← ***$plugins*** Plugins (array or boolean)

Returns:

Assoc. array (plugin => file)

See also:

TemplatePlugins::loadMultiple

Definition at line 148 of file Plugins.php.

References $plugins, and TemplateError::E_UNKNOWN_PLUGIN.

Referenced by loadMultiple().

9.10.3.7 TemplatePlugins::register ($ *plugin*, ITemplatePlugin $ *pluginObj*, $ *type*, array &$ *handlers*) `[private]`

Registers given elements.

Parameters:

← ***$plugin*** Plugin name

← ***$pluginObj*** Plugin instance

← ***$type*** Element type (`'tag'` or `'filter'`)

↔ ***$handlers*** Array of handlers

Definition at line 186 of file Plugins.php.

References TemplateError::E_INVALID_HANDLER.

9.10.3.8 TemplatePlugins::registerHooks ($ *plugin*, ITemplatePlugin $ *pluginObj*, array &$ *hooks*) `[private]`

Register given hooks.

Parameters:

← ***$plugin*** Plugin name

← ***$pluginObj*** Plugin instance

↔ ***$hooks*** Array of hooks

Definition at line 211 of file Plugins.php.

Referenced by load().

9.10.4 Member Data Documentation

9.10.4.1 TemplatePlugins::$plugins = array() [private]

Already loaded plugins.

Definition at line 17 of file Plugins.php.

Referenced by findPlugins(), and loadMultiple().

9.10.4.2 TemplatePlugins::$elements = array() [private]

Elements registry.

Definition at line 21 of file Plugins.php.

9.10.4.3 TemplatePlugins::$searchPaths = array()

Plugins' search paths.

Reference to `'pluginsPaths'` key of TemplateEnviron::$settings.

Definition at line 25 of file Plugins.php.

The documentation for this class was generated from the following file:

- Plugins.php

9.11 TemplateStdLibExPlugin Class Reference

New `StdLibEx` plugin, which combines old `CoreTags`, `CoreFilters` and `CoreHooks`.

Public Member Functions

- providedTags ()

  Provided tags.

- providedFilters ()

  Provided filters.

- providedHooks ()

  Provided hooks.

- providedHandlers ()

  Provided handlers.

- handleTAutoEscape (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% autoescape %}` *tag.*

- handleTBlock (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% block %}` *tag.*

- handleTCycle (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% cycle %}` *tag.*

- handleTDebug (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% debug %}` *tag.*

- handleTExtends (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% extends %}` *tag.*

- handleTFilter (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% filter %}` *tag.*

- handleTFirstOf (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% firstof %}` *tag.*

- handleTFor (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% for %}` *tag.*

- handleTEmpty (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% empty %}` *tag.*

- handleTIfChanged (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% ifchanged %}` *tag.*

- handleTIf (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% if %}` *tag.*

- handleTElse (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% else %}` *tag.*

- handleTElseIf (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% elseif %}` *tag.*

- handleTIfEqual (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% ifequal %}` *tag.*

- handleTIfNotEqual (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% ifnotequal %}` *tag.*

- handleTInclude (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% include %}` *tag.*

- handleTNow (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% now %}` *tag.*

- handleTSpaceless (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% spaceless %}` *tag.*

- handleTTemplateTag (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% templatetag %}` *tag.*

- handleTWidthRatio (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% widthratio %}` *tag.*

- handleTWith (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% with %}` *tag.*

- handleTPutBlock (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% putblock %}` *tag.*

- handleTCall (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% call %}` *tag.*

- handleTMeta (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$tag, array &$args)

  `{% meta %}` *tag.*

- handleFAdd (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `add` *filter.*

- handleFAddSlashes (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `addslashes` *filter.*

- handleFCapFirst (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `capfirst` *filter.*

- handleFCut (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `cut` *filter.*

- handleFDate (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `date` *filter.*

- handleFDefault (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `default` *filter.*

- handleFDefaultIfNone (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `default_if_none` *filter.*

- handleFDivisibleBy (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `divisibleby` *filter.*

- handleFEscape (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `escape` *filter.*

- handleFFileSizeFormat (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `filesizeformat` *filter.*

- handleFFixAmpersands (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `fix_ampersands` *filter.*

- handleFJoin (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `join` *filter.*

- handleFLength (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `length` *filter.*

- handleFLengthIs (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `length_is` *filter.*

- handleFLineBreaks (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `linebreaks` *filter.*

- handleFLineBreaksBR (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `linebreaksbr` *filter.*

- handleFLJust (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `ljust` *filter.*

- handleFLower (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `lower` *filter.*

- handleFMakeList (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `make_list` *filter.*

- handleFPluralize (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `pluralize` *filter.*

- handleFRandom (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `random` *filter.*

- handleFRemoveTags (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `removetags` *filter.*

- handleFRJust (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `rjust` *filter.*

- handleFSlugify (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `slugify` *filter.*

- handleFTitle (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `title` *filter.*

- handleFUpper (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `upper` *filter.*

- handleFURLEncode (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `urlencode` *filter.*

- handleFURLDecode (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `urldecode` *filter.*

- handleFWordCount (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `wordcount` *filter.*

- handleFWordWrap (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, array &$args)

  `wordwrap` *filter.*

- handleHAutoEscape (TemplateCompilerEx $compiler, array &$filterChain)

  Auto-escaping hook.

- handleHInternalVariable (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$variableCode, &$variableCheck)

  `{{ internal }}` *variable handler.*

- handleHForLoopVariable (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$variableCode, &$variableCheck)

  `{{ forloop }}` *variable handler.*

- handleHBlockVariable (TemplateCompilerEx $compiler, TemplateNodeEx $node, &$variableCode, &$variableCheck)

  `{{ block }}` *variable handler.*

Private Member Functions

- parseIfExpressionNonEmpty ($x)
- parseIfExpressionCheckParens (TemplateCompilerEx $compiler, TemplateNodeEx $node, $level, $final=false)
- parseIfExpression (TemplateCompilerEx $compiler, TemplateNodeEx $node, $expression)
- commonIfEqual (TemplateCompilerEx $compiler, TemplateNodeEx $node, array $variables, $operator)
- commonFJust ($sign, TemplateCompilerEx $compiler, TemplateNodeEx $node, &$filter, &$width)

9.11.1 Detailed Description

New `StdLibEx` plugin, which combines old `CoreTags`, `CoreFilters` and `CoreHooks`.

Since there is no more distinction between compile-time and run-time plugins, everything is provided within single class, to save I/O and object allocations.

Definition at line 685 of file StdLibEx.plugin.php.

9.11.2 Member Function Documentation

9.11.2.1 TemplateStdLibExPlugin::providedTags ()

Provided tags.

See Standard tags.

Returns:

Array of handlers

Definition at line 693 of file StdLibEx.plugin.php.

Referenced by providedHandlers().

9.11.2.2 TemplateStdLibExPlugin::providedFilters ()

Provided filters.

See Standard filters.

Returns:

Array of handlers

Definition at line 728 of file StdLibEx.plugin.php.

Referenced by providedHandlers().

9.11.2.3 TemplateStdLibExPlugin::providedHooks ()

Provided hooks.

Returns:

Array of handlers

Definition at line 768 of file StdLibEx.plugin.php.

Referenced by providedHandlers().

9.11.2.4 TemplateStdLibExPlugin::providedHandlers ()

Provided handlers.

See Standard library.

Returns:

Array of handlers

See also:

ITemplatePlugin::providedHandlers

Implements ITemplatePlugin.

Definition at line 786 of file StdLibEx.plugin.php.

References providedFilters(), providedHooks(), and providedTags().

9.11.2.5 TemplateStdLibExPlugin::handleTAutoEscape (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% autoescape %}` tag.

Definition at line 799 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, TemplateCompilerEx::handleChildren(), and TemplateCompilerEx::raiseIf().

9.11.2.6 TemplateStdLibExPlugin::handleTBlock (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% block %}` tag.

Definition at line 816 of file StdLibEx.plugin.php.

References TemplateCompilerEx::createBlock(), TemplateError::E_INVALID_ARGUMENT, and TemplateCompilerEx::raiseIf().

9.11.2.7 TemplateStdLibExPlugin::handleTCycle (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% cycle %}` tag.

Definition at line 843 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, TemplateError::E_INVALID_SYNTAX, TemplateUtils::panic(), TemplateCompilerEx::parseVariableExpression(), and TemplateCompilerEx::raiseIf().

9.11.2.8 TemplateStdLibExPlugin::handleTDebug (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% debug %}` tag.

Definition at line 906 of file StdLibEx.plugin.php.

9.11.2.9 TemplateStdLibExPlugin::handleTExtends (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% extends %}` tag.

Definition at line 911 of file StdLibEx.plugin.php.

References TemplateUtils::checkIORestriction(), TemplateError::E_INVALID_SYNTAX, and TemplateCompilerEx::raiseIf().

9.11.2.10 TemplateStdLibExPlugin::handleTFilter (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% filter %}` tag.

Definition at line 931 of file StdLibEx.plugin.php.

References TemplateCompilerEx::createBlock(), and TemplateCompilerEx::parseFilterChain().

9.11.2.11 TemplateStdLibExPlugin::handleTFirstOf (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% firstof %}` tag.

Definition at line 943 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, TemplateCompilerEx::parseVariableExpression(), and TemplateCompilerEx::raiseIf().

9.11.2.12 TemplateStdLibExPlugin::handleTFor (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% for %}` tag.

Definition at line 975 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, TemplateCompilerEx::findAlternativeBranch(), TemplateCompilerEx::generateUniqueBlock(), TemplateCompilerEx::handleChildren(), TemplateCompilerEx::parseVariableExpression(), TemplateCompilerEx::raise(), TemplateCompilerEx::raiseIf(), TemplateUtils::sanitize(), and TemplateUtils::split().

9.11.2.13 TemplateStdLibExPlugin::handleTEmpty (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% empty %}` tag.

Definition at line 1112 of file StdLibEx.plugin.php.

References TemplateUtils::panic().

9.11.2.14 TemplateStdLibExPlugin::handleTIfChanged (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% ifchanged %}` tag.

Definition at line 1120 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, TemplateCompilerEx::findAlternativeBranch(), TemplateCompilerEx::generateUniqueBlock(), TemplateCompilerEx::handleChildren(), TemplateCompilerEx::parseVariableExpression(), TemplateCompilerEx::raiseIf(), and TemplateUtils::sanitize().

9.11.2.15 TemplateStdLibExPlugin::parseIfExpressionNonEmpty ($ *x*) [private]

Definition at line 1175 of file StdLibEx.plugin.php.

9.11.2.16 TemplateStdLibExPlugin::parseIfExpressionCheckParens (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, $ *level*, $ *final* = `false`) [private]

Definition at line 1178 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, and TemplateCompilerEx::raiseIf().

Referenced by parseIfExpression().

9.11.2.17 TemplateStdLibExPlugin::parseIfExpression (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, $ *expression*) [private]

Definition at line 1192 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, TemplateCompilerEx::parseFilterChain(), parseIfExpressionCheckParens(), TemplateCompilerEx::parseVariableExpression(), TemplateCompilerEx::raiseIf(), and TemplateUtils::split().

9.11.2.18 TemplateStdLibExPlugin::handleTIf (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% if %}` tag.

Definition at line 1300 of file StdLibEx.plugin.php.

References TemplateCompilerEx::handleChildren().

9.11.2.19 TemplateStdLibExPlugin::handleTElse (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% else %}` tag.

Definition at line 1308 of file StdLibEx.plugin.php.

References TemplateUtils::panic().

9.11.2.20 TemplateStdLibExPlugin::handleTElseIf (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% elseif %}` tag.

Definition at line 1316 of file StdLibEx.plugin.php.

9.11.2.21 TemplateStdLibExPlugin::commonIfEqual (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, array $ *variables*, $ *operator*) [private]

Definition at line 1321 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, TemplateCompilerEx::handleChildren(), TemplateCompilerEx::parseFilterChain(), TemplateCompilerEx::parseVariableExpression(), TemplateCompilerEx::raiseIf(), and TemplateUtils::split().

Referenced by handleTIfEqual(), and handleTIfNotEqual().

9.11.2.22 TemplateStdLibExPlugin::handleTIfEqual (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% ifequal %}` tag.

Definition at line 1342 of file StdLibEx.plugin.php.

References commonIfEqual().

9.11.2.23 TemplateStdLibExPlugin::handleTIfNotEqual (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% ifnotequal %}` tag.

Definition at line 1347 of file StdLibEx.plugin.php.

References commonIfEqual().

9.11.2.24 TemplateStdLibExPlugin::handleTInclude (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% include %}` tag.

Definition at line 1352 of file StdLibEx.plugin.php.

References TemplateUtils::escape(), and TemplateCompilerEx::parseVariableExpression().

9.11.2.25 TemplateStdLibExPlugin::handleTNow (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% now %}` tag.

Definition at line 1369 of file StdLibEx.plugin.php.

References TemplateCompilerEx::parseFilterChain(), TemplateCompilerEx::parseVariableExpression(), and TemplateUtils::split().

9.11.2.26 TemplateStdLibExPlugin::handleTSpaceless (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% spaceless %}` tag.

Definition at line 1383 of file StdLibEx.plugin.php.

References TemplateCompilerEx::handleChildren().

9.11.2.27 TemplateStdLibExPlugin::handleTTemplateTag (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% templatetag %}` tag.

Definition at line 1394 of file StdLibEx.plugin.php.

References TemplateError::E_INVALID_ARGUMENT, and TemplateCompilerEx::raiseIf().

9.11.2.28 TemplateStdLibExPlugin::handleTWidthRatio (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% widthratio %}` tag.

Definition at line 1431 of file StdLibEx.plugin.php.

9.11.2.29 TemplateStdLibExPlugin::handleTWith (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% with %}` tag.

Definition at line 1458 of file StdLibEx.plugin.php.

9.11.2.30 TemplateStdLibExPlugin::handleTPutBlock (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% putblock %}` tag.

Definition at line 1481 of file StdLibEx.plugin.php.

9.11.2.31 TemplateStdLibExPlugin::handleTCall (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% call %}` tag.

Definition at line 1504 of file StdLibEx.plugin.php.

9.11.2.32 TemplateStdLibExPlugin::handleTMeta (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *tag*, array &$ *args*)

`{% meta %}` tag.

Definition at line 1543 of file StdLibEx.plugin.php.

9.11.2.33 TemplateStdLibExPlugin::handleFAdd (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`add` filter.

Definition at line 1557 of file StdLibEx.plugin.php.

9.11.2.34 TemplateStdLibExPlugin::handleFAddSlashes (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`addslashes` filter.

Definition at line 1565 of file StdLibEx.plugin.php.

9.11.2.35 TemplateStdLibExPlugin::handleFCapFirst (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`capfirst` filter.

Definition at line 1570 of file StdLibEx.plugin.php.

9.11.2.36 TemplateStdLibExPlugin::handleFCut (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`cut` filter.

Definition at line 1575 of file StdLibEx.plugin.php.

9.11.2.37 TemplateStdLibExPlugin::handleFDate (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`date` filter.

Definition at line 1587 of file StdLibEx.plugin.php.

9.11.2.38 TemplateStdLibExPlugin::handleFDefault (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`default` filter.

Definition at line 1592 of file StdLibEx.plugin.php.

9.11.2.39 TemplateStdLibExPlugin::handleFDefaultIfNone (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`default_if_none` filter.

Definition at line 1597 of file StdLibEx.plugin.php.

9.11.2.40 TemplateStdLibExPlugin::handleFDivisibleBy (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`divisibleby` filter.

Definition at line 1602 of file StdLibEx.plugin.php.

9.11.2.41 TemplateStdLibExPlugin::handleFEscape (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`escape` filter.

Definition at line 1623 of file StdLibEx.plugin.php.

9.11.2.42 TemplateStdLibExPlugin::handleFFileSizeFormat (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`filesizeformat` filter.

Definition at line 1628 of file StdLibEx.plugin.php.

9.11.2.43 TemplateStdLibExPlugin::handleFFixAmpersands (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`fix_ampersands` filter.

Definition at line 1645 of file StdLibEx.plugin.php.

9.11.2.44 TemplateStdLibExPlugin::handleFJoin (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`join` filter.

Definition at line 1650 of file StdLibEx.plugin.php.

9.11.2.45 TemplateStdLibExPlugin::handleFLength (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`length` filter.

Definition at line 1655 of file StdLibEx.plugin.php.

9.11.2.46 TemplateStdLibExPlugin::handleFLengthIs (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`length_is` filter.

Definition at line 1660 of file StdLibEx.plugin.php.

9.11.2.47 TemplateStdLibExPlugin::handleFLineBreaks (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`linebreaks` filter.

Definition at line 1665 of file StdLibEx.plugin.php.

9.11.2.48 TemplateStdLibExPlugin::handleFLineBreaksBR (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`linebreaksbr` filter.

Definition at line 1678 of file StdLibEx.plugin.php.

9.11.2.49 TemplateStdLibExPlugin::commonFJust ($ *sign*, TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, &$ *width*) `[private]`

Definition at line 1683 of file StdLibEx.plugin.php.

9.11.2.50 TemplateStdLibExPlugin::handleFLJust (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`ljust` filter.

Definition at line 1700 of file StdLibEx.plugin.php.

9.11.2.51 TemplateStdLibExPlugin::handleFLower (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`lower` filter.

Definition at line 1705 of file StdLibEx.plugin.php.

9.11.2.52 TemplateStdLibExPlugin::handleFMakeList (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`make_list` filter.

Definition at line 1710 of file StdLibEx.plugin.php.

9.11.2.53 TemplateStdLibExPlugin::handleFPluralize (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`pluralize` filter.

Definition at line 1715 of file StdLibEx.plugin.php.

9.11.2.54 TemplateStdLibExPlugin::handleFRandom (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`random` filter.

Definition at line 1742 of file StdLibEx.plugin.php.

9.11.2.55 TemplateStdLibExPlugin::handleFRemoveTags (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`removetags` filter.

Definition at line 1751 of file StdLibEx.plugin.php.

9.11.2.56 TemplateStdLibExPlugin::handleFRJust (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`rjust` filter.

Definition at line 1756 of file StdLibEx.plugin.php.

9.11.2.57 TemplateStdLibExPlugin::handleFSlugify (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`slugify` filter.

Definition at line 1761 of file StdLibEx.plugin.php.

9.11.2.58 TemplateStdLibExPlugin::handleFTitle (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`title` filter.

Definition at line 1775 of file StdLibEx.plugin.php.

9.11.2.59 TemplateStdLibExPlugin::handleFUpper (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`upper` filter.

Definition at line 1780 of file StdLibEx.plugin.php.

9.11.2.60 TemplateStdLibExPlugin::handleFURLEncode (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`urlencode` filter.

Definition at line 1785 of file StdLibEx.plugin.php.

9.11.2.61 TemplateStdLibExPlugin::handleFURLDecode (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`urldecode` filter.

Definition at line 1790 of file StdLibEx.plugin.php.

9.11.2.62 TemplateStdLibExPlugin::handleFWordCount (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`wordcount` filter.

Definition at line 1795 of file StdLibEx.plugin.php.

9.11.2.63 TemplateStdLibExPlugin::handleFWordWrap (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *filter*, array &$ *args*)

`wordwrap` filter.

Definition at line 1800 of file StdLibEx.plugin.php.

9.11.2.64 TemplateStdLibExPlugin::handleHAutoEscape (TemplateCompilerEx $ *compiler*, array &$ *filterChain*)

Auto-escaping hook.

Hooked into `parseFilterChain:entry`.

Definition at line 1812 of file StdLibEx.plugin.php.

9.11.2.65 TemplateStdLibExPlugin::handleHInternalVariable (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *variableCode*, &$ *variableCheck*)

`{{ internal }}` variable handler.

Hooked into `parseVariableExpression:postCodeGen`.

Definition at line 1844 of file StdLibEx.plugin.php.

9.11.2.66 TemplateStdLibExPlugin::handleHForLoopVariable (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *variableCode*, &$ *variableCheck*)

`{{ forloop }}` variable handler.

Hooked into `parseVariableExpression:postCodeGen`.

Definition at line 1896 of file StdLibEx.plugin.php.

9.11.2.67 TemplateStdLibExPlugin::handleHBlockVariable (TemplateCompilerEx $ *compiler*, TemplateNodeEx $ *node*, &$ *variableCode*, &$ *variableCheck*)

`{{ block }}` variable handler.

Hooked into `parseVariableExpression:postCodeGen`.

Definition at line 1909 of file StdLibEx.plugin.php.

The documentation for this class was generated from the following file:

- StdLibEx.plugin.php

9.12 TemplateStringIO Class Reference

String I/O implementation.

Public Member Functions

- upToDate (array &$settings, &$template, $mode)
- includeCode (array &$settings, &$template)
- className (array &$settings, &$template)

Protected Member Functions

- pfn (array &$settings, &$template)

9.12.1 Detailed Description

String I/O implementation.

Definition at line 172 of file IO.php.

9.12.2 Member Function Documentation

9.12.2.1 TemplateStringIO::pfn (array &$ *settings*, &$ *template*) [protected]

Reimplemented from TemplateFileIO.

Definition at line 173 of file IO.php.

Referenced by includeCode(), and upToDate().

9.12.2.2 TemplateStringIO::upToDate (array &$ *settings*, &$ *template*, $ *mode*)

Should check whether given template is up-to-date.

If driver uses `recompilationMode` setting, then it should use supplied `$mode` argument instead, to allow per-template mode override. Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

← ***$mode*** Recompilation mode

Return values:

true Template is up-to-date - no (re)compilation is needed

false Template must be (re)compiled

Reimplemented from TemplateFileIO.

Definition at line 185 of file IO.php.

References pfn(), and TemplateEnviron::RECOMPILE_ALWAYS.

9.12.2.3 TemplateStringIO::includeCode (array &$ *settings*, &$ *template*)

Should include template's code into global namespace.

It must ensure that no code redefinition will happen. Although parameters are supplied via reference, they should not be modified in any way.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Included class name

Reimplemented from TemplateFileIO.

Definition at line 193 of file IO.php.

References className(), TemplateError::E_IO_LOAD_FAILURE, and pfn().

9.12.2.4 TemplateStringIO::className (array &$ *settings*, &$ *template*)

Should return template's classname.

Parameters:

← ***$settings*** Settings array, see TemplateEnviron::$settings

← ***$template*** Template name

Returns:

Class name

Reimplemented from TemplateFileIO.

Definition at line 210 of file IO.php.

Referenced by includeCode().

The documentation for this class was generated from the following file:

- IO.php

9.13 TemplateUtils Class Reference

Namespace-acting all-static class.

Static Public Member Functions

- static escape ($str)

  Escape string to use in template class.

- static sanitize ($str)

  Sanitize string, for use as function name.

- static strip ($str)

  Strip newlines and spaces from string.

- static split ($separator, $str, $reverse=false)

  Split string into two.

- static splitEscaped ($delimiter, $expression)

  Properly splits given expression using given delimiter.

- static filterEmpty ($x)
- static doesImplement ($classOrObject, $interface)

  Check whether class implements given interface.

- static splitIODSN (array &$settings, $dsn)

  Splits I/O DSN into driver name and template name.

- static parseIODSN (array &$settings, $dsn)

  Splits I/O DSN, and creates correct driver object.

- static className ($template)

  Returns class name for given template or DSN.

- static panic ($file, $line)

  Panics.

- static checkIfAllowed ($obj, $type, $name, $node=null)

  Checks whether element is allowed.

- static checkIORestriction ($obj, $setting, $dsn, $expectedDriver, $node=null)

  Checks whether I/O restriction is in effect.

9.13.1 Detailed Description

Namespace-acting all-static class.

Definition at line 12 of file Utils.php.

9.13.2 Member Function Documentation

9.13.2.1 static TemplateUtils::escape ($ *str*) [static]

Escape string to use in template class.

Parameters:

← ***$str*** String

Returns:

Escaped string

Definition at line 19 of file Utils.php.

Referenced by TemplateStdLibExPlugin::handleTInclude(), TemplateCompilerEx::parseFilterChain(), and TemplateCompilerEx::parseVariableExpression().

9.13.2.2 static TemplateUtils::sanitize ($ *str*) [static]

Sanitize string, for use as function name.

Parameters:

← ***$str*** String

Returns:

Sanitized string

Definition at line 35 of file Utils.php.

Referenced by TemplateStdLibExPlugin::handleTFor(), and TemplateStdLibExPlugin::handleTIfChanged().

9.13.2.3 static TemplateUtils::strip ($ *str*) [static]

Strip newlines and spaces from string.

Parameters:

← ***$str*** String

Returns:

Stripped string

Definition at line 45 of file Utils.php.

Referenced by TemplateCompilerEx::commonVerifyElement(), and TemplateCompilerEx::runHooks().

9.13.2.4 static TemplateUtils::split ($ *separator*, $ *str*, $ *reverse* = `false`) [static]

Split string into two.

Parameters:

← ***$separator*** Separator

← ***$str*** String to split

← ***$reverse*** Use reversed search

Returns:

Array

Definition at line 57 of file Utils.php.

Referenced by TemplateStdLibExPlugin::commonIfEqual(), TemplateStdLibExPlugin::handleTFor(), TemplateStdLibExPlugin::handleTNow(), TemplateCompilerEx::parseFilterChain(), TemplateStdLibExPlugin::parseIfExpression(), TemplateCompilerEx::parserGetNextToken(), and splitIODSN().

9.13.2.5 static TemplateUtils::splitEscaped ($ *delimiter*, $ *expression*) [static]

Properly splits given expression using given delimiter.

Supports string delimiter escaping (\").

Parameters:

← ***$delimiter*** Delimiter to use

← ***$expression*** Expression to split

Returns:

Split expression

Todo

Better way?

Definition at line 79 of file Utils.php.

References panic().

Referenced by TemplateCompilerEx::parseFilterChain(), and TemplateCompilerEx::parserGetNextToken().

9.13.2.6 static TemplateUtils::filterEmpty ($ *x*) [static]

Definition at line 118 of file Utils.php.

9.13.2.7 static TemplateUtils::doesImplement ($ *classOrObject*, $ *interface*) [static]

Check whether class implements given interface.

Parameters:

← ***$classOrObject*** Mixed

← ***$interface*** String

Returns:

Boolean

Definition at line 127 of file Utils.php.

Referenced by TemplatePlugins::load().

9.13.2.8 static TemplateUtils::splitIODSN (array &$ *settings*, $ *dsn*) [static]

Splits I/O DSN into driver name and template name.

Parameters:

← ***$settings*** Settings array

← ***$dsn*** DSN to split

Returns:

Array(driver name, template name)

Definition at line 138 of file Utils.php.

References split().

Referenced by checkIORestriction(), and parseIODSN().

9.13.2.9 static TemplateUtils::parseIODSN (array &$ *settings*, $ *dsn*) [static]

Splits I/O DSN, and creates correct driver object.

Parameters:

← ***$settings*** Settings array

← ***$dsn*** DSN to parse

Returns:

Array(I/O driver, template name)

Definition at line 153 of file Utils.php.

References TemplateIO::get(), and splitIODSN().

Referenced by TemplateCompilerEx::compile(), and TemplateEnviron::include_().

9.13.2.10 static TemplateUtils::className ($ *template*) [static]

Returns class name for given template or DSN.

Note that only real template name should be used in class name.

Parameters:

← ***$template*** Template name

Returns:

Class name

Definition at line 165 of file Utils.php.

References panic().

9.13.2.11 static TemplateUtils::panic ($*file*, $*line*) [static]

Panics.

Used internally when sanity checks are failing.

Parameters:

← ***$file*** Source filename

← ***$line*** Source line

Definition at line 176 of file Utils.php.

References TemplateError::E_INTERNAL_CORE_FAILURE.

Referenced by checkIfAllowed(), checkIORestriction(), className(), TemplateCompilerEx::commonVerifyElement(), TemplateCompilerEx::generateUniqueBlock(), TemplateStdLibExPlugin::handleTCycle(), TemplateStdLibExPlugin::handleTElse(), TemplateStdLibExPlugin::handleTEmpty(), and splitEscaped().

9.13.2.12 static TemplateUtils::checkIfAllowed ($*obj*, $*type*, $*name*, $*node* = null) [static]

Checks whether element is allowed.

Raises TemplateError if it's not.

Parameters:

← ***$obj*** Instance of TemplateEnviron or TemplateCompilerEx

← ***$type*** Element type (`'plugin'`, `'tag'`, `'filter'`, `'function'`)

← ***$name*** Element name

← ***$node*** Optional instance of TemplateNodeEx

Definition at line 191 of file Utils.php.

References TemplateError::E_SECURITY_VIOLATION, panic(), TemplateEnviron::SECURITY_ALLOW_ALL, TemplateEnviron::SECURITY_ALLOW_DENY, TemplateEnviron::SECURITY_DENY_ALL, TemplateEnviron::SECURITY_DENY_ALLOW, and TemplateEnviron::SECURITY_DISABLE.

Referenced by TemplateCompilerEx::commonVerifyElement(), and TemplatePlugins::load().

9.13.2.13 static TemplateUtils::checkIORestriction ($ *obj*, $ *setting*, $ *dsn*, $ *expectedDriver*, $ *node* = `null`) `[static]`

Checks whether I/O restriction is in effect.

Raises TemplateError if setting is active and driver names mismatch.

Parameters:

← ***$obj*** Instance of TemplateEnviron or TemplateCompilerEx

← ***$setting*** Setting to check (either `restrictExtendIO` or `restrictIncludeIO`)

← ***$dsn*** DSN to check

← ***$expectedDriver*** Expected driver name

← ***$node*** Optional instance of TemplateNodeEx

Definition at line 274 of file Utils.php.

References TemplateError::E_SECURITY_VIOLATION, panic(), and splitIODSN().

Referenced by TemplateStdLibExPlugin::handleTExtends().

The documentation for this class was generated from the following file:

- Utils.php

Chapter 10

File Documentation

10.1 Base.php File Reference

File containing common abstract base class, used by compiled templates.

Classes

- class Template

  Abstract base class for templates.

10.1.1 Detailed Description

File containing common abstract base class, used by compiled templates.

Since:

1.1a0

License:

New BSD License

Author:

PiotrLegnica

Definition in file Base.php.

10.2 CompilerEx.php File Reference

New and shiny AST-based template compiler.

Classes

- class TemplateCompilerEx
  Primary compiler driver.
- class TemplateNodeEx
  Class-container for AST nodes.

10.2.1 Detailed Description

New and shiny AST-based template compiler.

Since:

1.1a0

Author:

PiotrLegnica

License:

New BSD License

Todo

Better variable parser?

Definition in file CompilerEx.php.

10.3 Environment.php File Reference

Client API of the library.

Classes

- class TemplateEnviron

  Template environment - library's end-user API.

10.3.1 Detailed Description

Client API of the library.

Since:

1.1a0

Author:

PiotrLegnica

License:

New BSD License

Definition in file Environment.php.

10.4 Error.php File Reference

Exceptions used in the library.

Classes

- class TemplateError

  Main and currently the only exception type thrown by SithTemplate internals.

10.4.1 Detailed Description

Exceptions used in the library.

Since:

1.1a0

Author:

PiotrLegnica

License:

New BSD License

Definition in file Error.php.

10.5 IIODriver.php File Reference

Common interface for I/O drivers.

Classes

- interface ITemplateIODriver

  Interface required for all I/O drivers.

10.5.1 Detailed Description

Common interface for I/O drivers.

Since:

1.1a0

Author:

PiotrLegnica

License:

New BSD License

Definition in file IIODriver.php.

10.6 IO.php File Reference

I/O management, and default I/O drivers.

Classes

- class TemplateIO
  Global I/O driver storage.
- class TemplateFileIO
  File I/O implementation.
- class TemplateStringIO
  String I/O implementation.

10.6.1 Detailed Description

I/O management, and default I/O drivers.

Since:

1.1a0

Author:

PiotrLegnica

License:

New BSD License

Definition in file IO.php.

10.7 IPlugin.php File Reference

Common interface for plugins.

Classes

- interface ITemplatePlugin

  Interface required for all plugins.

10.7.1 Detailed Description

Common interface for plugins.

Since:

1.1a0

Author:

PiotrLegnica

License:

New BSD License

Definition in file IPlugin.php.

10.8 Plugins.php File Reference

Contains plugin machinery.

Classes

- class TemplatePlugins

  Handles discovery, registration and utilization of plugins.

10.8.1 Detailed Description

Contains plugin machinery.

Since:

1.1a0

Author:

PiotrLegnica

License:

New BSD License

Definition in file Plugins.php.

10.9 SithTemplate.php File Reference

Entry point of the SithTemplate library, containing global constants and SPL autoloader.

Enumerations

- enum SITHTEMPLATE_VERSION

  Current version of the library.

Functions

- sithtemplate_spl_autoload ($cls)

  SPL autoloader for SithTemplate.

10.9.1 Detailed Description

Entry point of the SithTemplate library, containing global constants and SPL autoloader.

Available constants:

- SITHTEMPLATE_VERSION - non-overridable, contains current version of the library
- SITHTEMPLATE_DIR - overridable, contains path to library's files
- SITHTEMPLATE_NO_AUTOLOADER - if defined, autoloader won't be registered with SPL
- SITHTEMPLATE_MBSTRING_UTF8 - if defined, mbstring internal encoding won't be changed to UTF-8 (note that no other setting is tested, and therefore library may fail to work properly)
-

Author:

PiotrLegnica

License:

New BSD License

Definition in file SithTemplate.php.

10.9.2 Enumeration Type Documentation

10.9.2.1 enum SITHTEMPLATE_VERSION

Current version of the library.

Definition at line 201 of file SithTemplate.php.

10.9.3 Function Documentation

10.9.3.1 siththemplate_spl_autoload ($ *cls*)

SPL autoloader for SithTemplate.

Parameters:

$cls Class to load

Since:

0.4.0

Definition at line 171 of file SithTemplate.php.

10.10 StdLibEx.plugin.php File Reference

Contains all of the standard tags, filters and hooks.

Classes

- class TemplateStdLibExPlugin

  New `StdLibEx` *plugin, which combines old* `CoreTags`, `CoreFilters` *and* `CoreHooks`.

10.10.1 Detailed Description

Contains all of the standard tags, filters and hooks.

Since:

1.1a0

Author:

PiotrLegnica

License:

New BSD License

Definition in file StdLibEx.plugin.php.

10.11 Utils.php File Reference

Utilities used throughout SithTemplate.

Classes

- class TemplateUtils

  Namespace-acting all-static class.

10.11.1 Detailed Description

Utilities used throughout SithTemplate.

License:

New BSD License

Author:

PiotrLegnica

Definition in file Utils.php.

Chapter 11

Example Documentation

11.1 00_hello.php

An "hello world" example, showing how to create template environment with default settings, template object using string I/O, and render it.

```
<?php
require_once 'SithTemplate.php';

// 1. We create environment
$environ  = new TemplateEnviron;
// 2. Next, we create template object
// Library will take care of the (re)compilation.
// SithTemplate 1.1 introduced unified I/O system,
// which allows you to easily inline small templates in your PHP code.
$template = $environ->get('string://Hello world');
// 3. Finally, we render and display previously created template
// You may notice that display/fetch APIs are gone, replaced by
// generic ones - you need to display template output by yourself.
//
// You can also see that environment object is passed back to the template -
// it is used in several places, like {% include %}-generated code, but passing
// it here, and not during construction, keeps template object more lightweight
// and independent, as it doesn't carry reference to original environment.
// It also eliminates possibility of circular reference, when template object
// is stored in environ's internal cache.
echo $template->render(array(), $environ);

// If you don't want to cache the template object on your own, you can use
// chained calls to cachedGet and render:
$environ->cachedGet('string://Other')->render(array(), $environ);

// If you don't need the object at all, you can call TemplateEnviron::render inst
      ead.
// This call is the same as the chained call above, just shorter and less explici
      t.
$environ->render('string://Other', array());
```

11.2 01_io.php

An example showing different default I/O drivers.

```
<?php
require_once 'SithTemplate.php';

$environ = new TemplateEnviron;

// SithTemplate 1.1 comes with two I/O drivers bundled:
//
// - "file" I/O - a traditional template-from-file driver.
//   This driver uses "inputPrefix" as source directory with templates,
//   and "outputPrefix" as cache directory, to store metadata and
//   compiled templates' code.
echo $environ->get('template.html')->render(array(), $environ);
// - "string" I/O, which allows you to inline templates in code.
//   This driver uses only "outputPrefix" setting.
echo $environ->get('string://Hai')->render(array(), $environ);
//
// inputPrefix defaults to ./templates/
// outputPrefix defaults to ./templates_c/
```

11.3 02_settings.php

An example showing how to change default library settings.

```
<?php
require_once 'SithTemplate.php';

// You can change default settings during TemplateEnviron construction,
// by passing associative array to the constructor.
$environ = new TemplateEnviron(array(
 'inputPrefix'  => './templates/',
 'outputPrefix' => './templates_c/',
));

// You can also load settings from INI file, using static named constructor
// See sample-configuration.ini for syntax.
$environ = TemplateEnviron::createFromINI('settings.ini');

// Finally, you can change settings in runtime, by modifying settings
// array directly. Note that some settings won't take effect if changed
// in that way. Refer to documentation for more information.
$environ->settings['recompilationMode'] = TemplateEnviron::RECOMPILE_ALWAYS;
```

11.4 03_context.php

An example showing how to create and use template context.

```
<?php
require_once 'SithTemplate.php';

$environ = new TemplateEnviron;

// Context array is passed as first argument to Template::render, or as second
// argument to TemplateEnviron::render.
$tpl = $environ->get('string://{{ foo }} ');

echo $tpl->render(array('foo' => 'first'),  $environ);
echo $tpl->render(array('foo' => 'second'), $environ);
// Will produce: "first second "

// Above is the simplest variable expression. To access nested elements, slightly
       more
// complex syntax is required, presented below, with equivalent PHP code:
//
// - accessing a named array element
//   {{ foo.bar }} is equivalent to $context['foo']['bar']
// - accessing a numeric array index
//   {{ foo.42 }} is equivalent to $context['foo'][42]
// - accessing a named or numeric array index, using value of another variable as
       key
//   {{ foo.[bar] }} is equivalent to $context['foo'][$context['bar']]
//
// Same syntax rules applies to object properties - you just use -> operator inst
      ead of ., e.g.
// {{ foo->bar }}.
//
// This syntax allows you to create very complex constructs, like:
//  {{ [one->[two]].three->four }} which is equivalent to
//  $context[ $context['one']->{$context['two']} ]['three']->four
//
// SithTemplate by default generates code to check whether variable really exists
       in the context
// before it is used, which triggers E_USER_WARNING if it doesn't. This can inter
      fere with "optional"
// variables (e.g. ones used with 'default' filter). You can tell compiler to omi
      t this code, by prefixing
// entire expression with @ sign:
//  {{ @non-existant-variable }}

// Filter chains are built with pipe operator. Filter arguments are comma-separat
      ed, passed after colon.
//  {{ variable|filter1|filter2:variable2,"foo" }}
// is roughly equivalent (if filters were simply functions) to
//  filter2(filter1($context['variable']), $context['variable2'], 'foo')
```

11.5 04_syntax.html

Example template showing syntax rules.

```
{{ this.is.variable }}

There are two kinds of tags:
{% inline %} and {% block %} which require an ending tag {% endblock %}

{# this is single-line comment - they are ignored by the compiler #}
{% comment %}
 and this is
 multi-line comment
 which is implemented as built-in block tag
{% endcomment %}
```

11.6 05_inheritance_parent.html

Inheritance example - parent template.

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <link rel="stylesheet" href="style.css">
  <title>{% block title %}Awesome HTML5 site{% endblock %}</title>
 </head>
 <body>
  <div id="main">
  {% block contents %}
   <p>
    This is default content of the block.
   </p>
   <p>
    It will be used in case no children template will override it,
    or when parent template will be rendered directly.
   </p>
  {% endblock %}
  </div>

  <div id="somethingelse">
  {% block other %}
   <p>
    Foo.
   </p>
  {% endblock %}
  </div>
</body>
</html>
```

11.7 06_inheritance_child.html

Inheritance example - child (inheriting) template.

```
{# This is the most important line: #}
{% extends "05_inheritance_parent.html" %}

{% block contents %}
 <p>
  And here is overriden contents of this block!
 </p>
{% endblock %}

{% block other %}
 <p>
  This one shows how to use parent block's contents.
 </p>
 {{ block.super }}
{% endblock %}
```

11.8 07_inheritance_result.html

Inheritance example - rendered child template.

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <link rel="stylesheet" href="style.css">
  <title>Awesome HTML5 site</title>
 </head>
 <body>
  <div id="main">
   <p>
    And here is overriden contents of this block!
   </p>
  </div>

  <div id="somethingelse">
   <p>
    This one shows how to use parent block's contents.
   </p>
   <p>
    Foo.
   </p>
  </div>
</body>
</html>
```

11.9 08_security.php

An example showing various security-related settings in SithTemplate.

```
<?php
require_once 'SithTemplate.php';

$environ = new TemplateEnviron;

// All security settings are set using environment's setting array.
// Some of them may be enforced at runtime, and some at compile time,
// see TemplateEnviron::$settings documentation for reference.

// The most common is variable autoescaping, which applies "escape" filter
// to all stand-alone variables (i.e. {{ vars }}), unless they are marked
// with "safe" pseudofilter.
// Autoescaping is turned on with "autoEscape" boolean setting.
$environ->settings['autoEscape'] = true;
$environ->render('string://{{ var }}', array('var' => '<b>'));      // will retur
      n "&lt;b&gt;"
$environ->render('string://{{ var|safe }}', array('var' => '<b>')); // will retur
      n "<b>"

// Next, there are I/O restriction settings. They allow you to enforce specific I
      /O driver,
// e.g. when you load template using your own db:// driver, and you don't want lo
      aded template
// to use any other I/O driver, like file:// or string://.
// Note that this is a bit primitive, and may be replaced sometime in the future.

// I/O restrictions are turned on by "restrictIncludeIO" and "restrictExtendIO" b
      oolean settings.
$environ->settings['restrictIncludeIO'] = true;
$environ->render('string://{% include "string://test" %}', array());    // will r
      eturn "test"
$environ->render('string://{% include "file://test.html" %}', array()); // will r
      aise TemplateError

// Next, there are {{ internal }} access restrictions (again, a bit primitive and
       boolean only).
// Since {{ internal }} allows template to access global constants and supergloba
      l arrays
// (like $_SERVER or $_ENV), it may introduce security risk in sandboxed environm
      ent
// (e.g. when templates are loaded from DB, and users can edit them).
// {{ internal }} restrictions can be set by turning off "allowInternalRequest"
// and/or "allowInternalConstants" boolean settings.
// Since this is boolean-only and a bit inconsistent, it may get replaced.
$environ->render('string://{{ internal.request.ENV.PATH.0 }}', array()); // will
      return $_ENV['PATH'][0]
$environ->settings['allowInternalRequest'] = false;
$environ->render('string://{{ internal.request.ENV.PATH.0 }}', array()); // will
      raise TemplateError

// Finally, there are security lists, that allows you to handpick plugins, tags,
      filters and
// plain PHP functions that templates are allowed to use. Lists are the most comp
      lex of security
// settings, as they support multiple modes of evaluation (allow all, deny; allow
      , deny; deny, allow; deny all, allow),
// and wildcards (TemplateEnviron::SECURITY_MATCH_EVERYTHING).
// Evaluation mode is controlled by "securityEvalMode" enumerative setting, and l
      ists themselves
// are stored in several array settings: "allowedPlugins", "allowedTags", "allowe
      dFilters", "allowedFunctions"
// and their "disallowed*" counterparts.
```

```
$environ->settings['securityEvalMode'] = TemplateEnviron::SECURITY_DENY_ALL; // m
      ost restrictive setting
$environ->settings['allowedTags']      = array('block'); // you don't have to spe
      cify ending tags
$environ->render('string://{% block foo %}foo{% endblock %}', array()); // will r
      eturn "foo"
$environ->render('string://{% comment %}foo{% endcomment %}', array()); // will r
      aise TemplateError
```

11.10 09_errors.php

An example showing error handling.

```
<?php
require_once 'SithTemplate.php';

$environ = new TemplateEnviron;

// You should always remember about error handling
// If error occurs during template compilation, exception message
// may contain template file and approx. line of the mistake.

// Errors are grouped - every group has it's own errorcode, specified
// as class constants in TemplateError.
try {
 $environ->render('string://{% bkock foo %}Typos are evil.{% endblock %}', array(
      ));
} catch (TemplateError $e) {
 echo $e->getMessage(); // Unknown tag ...
 echo $e->getCode();    // TemplateError::E_UNKNOWN_TAG
}
```

11.11 stdlib/00_tag_autoescape.html

Example on how to use {% autoescape %} tag.

```
{# assumptions: variable = "<b>foo</b>", global autoescape is off #}

{{ variable }} {# => <b>foo</b> #}

{% autoescape on %}
 {{ variable }}      {# => &lt;b&gt;foo&lt;/b&gt; #}
 {{ variable|safe }} {# => <b>foo</b> #}
{% endautoescape %}
```

11.12 stdlib/00_tag_block.html

Example on how to use {% block %} tag.

```
{% block foo %}
 <p>Hello</p> {# will be outputted #}
{% endblock %}

{% block bar store %}
 <p>world</p> {# will not be outputted #}
{% endblock %}
```

11.13 stdlib/00_tag_call.html

Example on how to use {% call %} tag.

```
{% call "sha1" "foo" %}            {# => 0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33
       #}
{% call "sha1" "foo" as fooHash %} {# will create fooHash variable instead of out
      putting #}

<p>{{ fooHash }}</p> {# => 0beec7b5ea3f0fdbc95d0dd47f3c5bc275da8a33 #}
```

11.14 stdlib/00_tag_cycle.html

Example on how to use {% cycle %} tag.

```
{% for entry in entries %}
 {% cycle "red" "blue" "green" %}
{% endfor %}

{% cycle "red" "blue" "green" as rgbCycle %} {# => red #}
{% cycle rgbCycle %}                         {# => blue #}
{% cycle rgbCycle %}                         {# => green #}
{% cycle rgbCycle %}                         {# => red #}
```

11.15 stdlib/00_tag_extends.html

Example on how to use {% extends %} tag.

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

{# extending from string is allowed, but very limited ATM #}
{# extending from a different I/O driver is perfectly fine, though #}
{% extends "string://another" %}

{# note that in one template, there may be only one {% extends %} #}
```

11.16 stdlib/00_tag_filter.html

Example on how to use {% filter %} tag.

```
{% filter lower|cut:"foo" %}
 FoO bAr FoO {# => bar #}
{% endfilter %}
```

11.17 stdlib/00_tag_firstof.html

Example on how to use {% firstof %} tag.

```
{# assumptions: foo and bar does not exist, baz exists #}
{# foo = "foo", bar = "bar", baz = "baz" #}

{% firstof foo bar baz %}    {# => baz #}
{% firstof foo bar "none" %} {# => none #}
```

11.18 stdlib/00_tag_for_empty.html

Example on how to use {% for %} and {% empty %} tags.

```
{# assumptions: entries is a non-empty array, tags is an empty array #}

{# this will output the entries array #}
{% for entry in entries %}
 {{ entry }}
{% empty %}
 No entries.
{% endfor %}

{# this will output "No tags." #}
{% for tag in tags %}
 {{ tag }}
{% empty %}
 No tags.
{% endfor %}
```

11.19 stdlib/00_tag_if_else_elseif.html

Example on how to use {% if %}, {% else %} and {% elseif %} tags.

```
{% if foo %}
 foo
{% elseif ((bar eq "foo") and baz) %}
 bar
{% elseif quux %}
 baz
{% else %}
 something else
{% endif %}
```

11.20 stdlib/00_tag_ifchanged.html

Example on how to use {% ifchanged %} and {% else %} tag.

11.21 stdlib/00_tag_include.html

Example on how to use {% include %} tag.

```
{% include "template.html" %}
{% include variable %} {# => contents of the variable will be used #}
```

11.22 stdlib/00_tag_load.html

Example on how to use {% load %} tag.

```
{% load SomeLibrary %}
```

11.23 stdlib/00_tag_meta.html

Example on how to use {% meta %} tag.

```
{% meta foo "This template's foo is bar" %}
{% meta bar "This template's bar is foo" %}
```

11.24 stdlib/00_tag_now.html

Example on how to use {% now %} tag.

```
{% now "d-m-Y, H:i:s" %}
```

11.25 stdlib/00_tag_putblock.html

Example on how to use {% putblock %} tag.

```
{% block foo store %}Hello{% endblock %} world {% putblock foo %} {# => world Hel
      lo #}
```

11.26 stdlib/00_tag_spaceless.html

Example on how to use {% spaceless %} tag.

```
{% spaceless %}
 <p>
            <b>  foo  </b>
 </p>
{% endspaceless %}

{# => <p><b>  foo  </b></p> #}
```

11.27 stdlib/00_tag_widthratio.html

Example on how to use {% widthratio %} tag.

```
{# assumptions: current = 175, max = 200 #}

{% widthratio current max 100 %} {# => 88 #}
```

11.28 stdlib/00_tag_with.html

Example on how to use {% with %} tag.

```
{% with one.two->three.[four]|lower as simpler %}
 {{ simpler }}
{% endwith %}
```

11.29 stdlib/01_filter_add.html

Example on how to use add filter.

```
{# assumptions: var = 5, var2 = "foo" #}

{{ var|add:5 }}       {# => 10 #}
{{ var2|add:" bar" }} {# => foo bar #}
```

11.30 stdlib/01_filter_addslashes.html

Example on how to use addslashes filter.

```
{# var = foo\bar' #}

{{ var|addslashes }} {# => foo\\bar\' #}
```

11.31 stdlib/01_filter_capfirst_lower_upper_title.html

Example on how to use capfirst, lower, upper, title filters.

```
{# var = hEllo woRld #}

{{ var|capfirst }} {# => HEllo woRld #}
{{ var|lower }}    {# => hello world #}
{{ var|upper }}    {# => HELLO WORLD #}
{{ var|title }}    {# => HEllo WoRld #}
```

11.32 stdlib/01_filter_cut.html

Example on how to use cut filter.

```
{# var = "foo bar baz" #}

{{ var|cut:"bar" }} {# => foo  baz #}
```

11.33 stdlib/01_filter_date.html

Example on how to use date filter.

```
{# var = 123456789 #}

{{ var|date:"d-m-Y, H:i:s" }} {# => 29-11-1973, 22:33:09 #}
```

11.34 stdlib/01_filter_default_default_if_none.html

Example on how to use default, default_if_none filters.

```
{# nonexistant does not exist, nullvar is NULL #}

{{ @nonexistant|default:"none" }}    {# => none; silenced, otherwise a warning wo
      uld be issued #}
{{ nullvar|default_if_none:"none" }} {# => none #}
```

11.35 stdlib/01_filter_divisibleby.html

Example on how to use divisibleby filter.

```
{# var1 = 2, var2 = 3 #}

{% if var1|divisibleby:2 %} yes {% else %} no {% endif %} {# => yes #}
{% if var2|divisibleby:2 %} yes {% else %} no {% endif %} {# => no #}
```

11.36 stdlib/01_filter_escape.html

Example on how to use escape and safe filters.

```
{# var = "<b>foo</b>" #}

{{ var|escape }} {# => &lt;b&gt;foo&lt;/b&gt; #}
```

11.37 stdlib/01_filter_filesizeformat.html

Example on how to use filesizeformat filter.

```
{# var1 = 1000, var2 = 41211, var3 = 5230121, var4 = 5232338952 #}

{{ var1|filesizeformat }} {# => 1000 b #}
{{ var2|filesizeformat }} {# => 40.25 kB #}
{{ var3|filesizeformat }} {# => 4.99 MB #}
{{ var4|filesizeformat }} {# => 4.87 GB #}
```

11.38 stdlib/01_filter_fix_ampersands.html

Example on how to use fix_ampersands filter.

```
{# var = "foo&bar" #}

{{ var|fix_ampersands }} {# => foo&amp;bar #}
```

11.39 stdlib/01_filter_join.html

Example on how to use join filter.

```
{# var = array('a', 'b', 'c') #}

{{ var|join:", " }} {# => a, b, c #}
```

11.40 stdlib/01_filter_length_length_is.html

Example on how to use length, length_is filters.

```
{# var1 = "foo", var2 = array('a', 'b') #}

{{ var1|length }}                                        {# => 3 #}
{% if var2|length_is:3 %} yes {% else %} no {% endif %} {# => no #}
```

11.41 stdlib/01_filter_linebreaks_linebreaksbr.html

Example on how to use linebreaks, linebreaksbr filters.

```
{% comment %}
 var = "foo

 bar
 baz"
{% endcomment %}

{{ var|linebreaks }}
{% comment %}
 Outputs (whitespace may vary):
 <p>foo</p>

 <p>bar<br />
 baz</p>
{% endcomment %}

{{ var|linebreaksbr }}
{% comment %}
 Outputs (whitespace may vary):
 foo<br />
 <br />
 bar<br />
 baz
{% endcomment %}
```

11.42 stdlib/01_filter_ljust_rjust.html

Example on how to use ljust, rjust filters.

```
{# var = "foo" #}

{{ var|ljust:5 }} {# => "foo  " #}
{{ var|rjust:5 }} {# => "  foo" #}
```

11.43 stdlib/01_filter_make_list.html

Example on how to use make_list filter.

```
{# var = "foo" #}

{{ var|make_list|join:", " }} {# => f, o, o #}
```

11.44 stdlib/01_filter_pluralize.html

Example on how to use pluralize filter.

```
{# var1 = 1, var2 = 7 #}

{{ var1|pluralize }} {# outputs nothing #}
{{ var2|pluralize }} {# outputs 's' #}

{{ var1|pluralize:"es" }} {# outputs nothing #}
{{ var2|pluralize:"es" }} {# outputs 'es' #}

{{ var1|pluralize:"e,es" }} {# outputs 'e' #}
{{ var2|pluralize:"e,es" }} {# outputs 'es' #}
```

11.45 stdlib/01_filter_random.html

Example on how to use random filter.

```
{# var = array('a', 'b') #}

{{ var|random }} {# will output either 'a' or 'b' #}
```

11.46 stdlib/01_filter_removetags.html

Example on how to use removetags filter.

```
{# var = "<b>foo</b>" #}

{{ var|removetags }} {# => foo #}
```

11.47 stdlib/01_filter_slugify.html

Example on how to use slugify filter.

```
{# UTF-8 ahead #}
{# var = "Hai ąą   #1331" #}

{{ var|slugify }} {# => hai--1331 #}
```

11.48 stdlib/01_filter_urlencode_urldecode.html

Example on how to use urlencode, urldecode filters.

```
{# var = "http://example.com" #}

{{ var|urlencode }}           {# => http%3A%2F%2Fexample.com #}
{{ var|urlencode|urldecode }} {# => http://example.com #}
```

11.49 stdlib/01_filter_wordcount.html

Example on how to use wordcount filter.

```
{# var = "foo bar baz" #}

{{ var|wordcount }} {# => 3 #}
```

11.50 stdlib/01_filter_wordwrap.html

Example on how to use wordwrap filter.

```
{# var = "foobar foobar foobar" #}

{{ var|wordwrap:6|linebreaksbr }}
{% comment %}
 Will output:
 foobar<br />
 foobar<br />
 foobar
{% endcomment %}
```

11.51 stdlib/02_var_block.html

Example on how to use {{ block }} special variable.

```
{# parent template: #}
{% block something %}
 <p>something!</p>
{% endblock %}

{# child template: #}
{% block something %}
 <p>something else?</p> {{ block.super }}
{% endblock %}

{# output: <p>something else?</p> <p>something!</p> #}
{# whitespace may vary #}
```

11.52 stdlib/02_var_forloop.html

Example on how to use {{ forloop }} special variable.

```
{% for v in vs %}
 {% if forloop.first %}first!{% endif %}
 {% if forloop.last %}last!{% endif %}
 {{ forloop.counter }}/{{ forloop.counter0 }}
 {{ forloop.revcounter }}/{{ forloop.revcounter0 }}
 {% for x in xs %}
  {{ forloop.parentloop.counter }}
 {% endfor %}
{% endfor %}
```

11.53 stdlib/02_var_internal.html

Example on how to use {{ internal }} special variable.

```
{{ internal.request.POST.foo }}
{{ internal.const.PHP_VERSION }}
{{ internal.version }}
```