Smarty - le moteur et compilateur de template PHP

par

Monte Ohrt <monte at ohrt dot com> et Andrei Zmievski <andrei@php.net>

Smarty - le moteur et compilateur de template PHP

Publié 07-09-2007
Copyright © 2001-2005 New Digital Group, Inc.

Table des matières

Préface

"Comment rendre mes scripts PHP indépendants de la présentation ?". Voici sans doute la question la plus posée sur la mailing list PHP. Alors que PHP est étiqueté "langage de script pour HTML", on se rend vite compte, après quelques projets qui mélangent sans complexe HTML et PHP, que la séparation entre la forme et le contenu, c'est bien [TM]. De plus, dans de nombreuses entreprises les rôles du designer et du programmeur sont distincts. La solution template coule donc de source.

Dans notre entreprise par exemple, le développement d'une application se fait de la manière suivante : une fois le cahier des charges écrit, le designer réalise une maquette, et donne ses interfaces au programmeur. Le programmeur implémente les fonctionnalités applicatives et utilise les maquettes pour faire des squelettes de templates. Le projet est alors passé au designer HTML/responsable de la mise en page qui amène les templates jusqu'au faîte de leur gloire. Il est possible que le projet fasse une fois ou deux des allers/retours entre la programmation et la présentation. En conséquence, il est important de disposer d'un bon système de template. Les programmeurs ne veulent pas avoir à faire au HTML, et ne veulent pas non plus que les designers HTML bidouillent le code PHP. Les designers ont besoin d'outils comme des fichiers de configuration, des blocs dynamiques et d'autres solutions pour répondre à des problématiques d'interface, mais ne veulent pas nécessairement avoir à faire à toutes les subtilités de la programmation PHP.

Un rapide tour d'horizon des solutions type template aujourd'hui et l'on s'aperçoit que la plupart d'entre elles n'offrent que des moyens rudimentaires pour substituer des variables dans des templates, ainsi que des fonctionnalités limitées de blocs dynamiques. Cependant nous avons besoin d'un peu plus. Nous ne voulons pas que les programmeurs s'occupent de la présentation HTML du TOUT, mais celà est pratiquement inévitable. Par exemple, si un designer veut des couleurs d'arrière plan différentes pour alterner entre différents blocs dynamiques, il est nécessaire que ce dernier travaille avec le programmeur. Nous avons aussi besoin que les designers soient capables de travailler avec leurs propres fichiers de configuration pour y récupérer des variables, exploitables dans leurs templates. Et la liste est longue.

Fin 1999, nous avons commencé à écrire une spécification pour un moteur de template. Une fois la spécification terminée, nous avons commencé à travailler sur un moteur de template écrit en C qui pourrait, avec un peu de chance, être inclus à PHP. Non seulement nous avons rencontré des problèmes techniques complexes, mais nous avons participés à de nombreux débats sur ce que devait et ce que ne devait pas faire un moteur de template. De cette expérience nous avons décidé qu'un moteur de template se devait d'être écrit sous la forme d'une classe PHP, afin que quiconque puisse l'utiliser à sa convenance. Nous avons donc réalisé un moteur de template qui se contentait de faire celà, et SmartTemplate™ a vu le jour (note : cette classe n'a jamais été soumise au public). C'était une classe qui faisait pratiquement tout ce que nous voulions : substitution de variables, inclusion d'autres templates, intégration avec des fichiers de configuration, intégration de code PHP, instruction 'if' basique et une gestion plus robuste des blocks dynamiques imbriqués. Elle faisait tout celà avec des expressions rationnelles et le code se révéla, comment dire, impénétrable. De plus, elle était relativement lente pour les grosses applications à cause de l'analyse et du travail sur les expressions rationnelles qu'elle devait faire à chaque exécution. Le plus gros problème du point de vue du programmeur était tout le travail nécessaire en amont, dans le script PHP, pour configurer et exécuter les templates, et les blocs dynamiques. Comment rendre tout ceci plus simple ?

Puis vint la vision de ce que devait devenir Smarty. Nous savons combien le code PHP peut être rapide sans le coût d'analyse des templates. Nous savons aussi combien fastidieux et décourageant peut paraître le langage pour le designer moyen, et que celà peut être remplacé par une syntaxe spécifique, beaucoup plus simple. Et si nous combinions les deux forces ? Ainsi, Smarty était né...:-)

Partie I. Pour commencer

Table des matières

Chapitre 1. Qu'est-ce que Smarty ?

Smarty est un moteur de template pour PHP. Plus précisément, il facilite la séparation entre la logique applicative et la présentation. Celà s'explique plus facilement dans une situation où le programmeur et le designer de templates jouent des rôles différents, ou, comme la plupart du temps, sont deux personnes distinctes.

Supposons par exemple que vous concevez une page Web qui affiche un article de newsletter. Le titre, le sous-titre, l'auteur et le corps sont des éléments de contenu, ils ne contiennent aucune information concernant la présentation. Ils sont transmis à Smarty par l'application, puis le designer de templates éditent les templates et utilisent une combinaison de balises HTML et de balises de templates pour formater la présentation de ces éléments (tableaux HTML, couleurs d'arrière-plan, tailles des polices, feuilles de styles, etc.). Un beau jour le programmeur a besoin de changer la façon dont le contenu de l'article est récupéré (un changement dans la logique applicative). Ce changement n'affecte pas le designer de templates, le contenu arrivera toujours au template de la même façon. De même, si le le designer de templates veut changer complétement l'apparence du template, aucun changement dans la logique de l'application n'est nécessaire. Ainsi le programmeur peut changer la logique de l'application sans restructurer les templates, et le designer de templates peut changer les templates sans briser la logique applicative.

Un des objectifs de Smarty est la séparation de la logique métier de la logique de présentation. Celà signifie que les templates peuvent contenir des traitements, du moment qu'il soit relatif à de la présentation. Inclure d'autres templates, alterner les couleurs des lignes d'un tableau, mettre du texte en majuscule, parcourir un tableau de données pour l'afficher, etc. sont toutes des actions relatives à du traitement de présentation. Celà ne signifie pas que Smarty requiert une telle séparation de votre part. Smarty ne sais pas quoi est quoi, c'est donc à vous de placer la logique de présentation dans vos templates. Ainsi, si vous *ne désirez pas* disposer de logique métier dans vos templates, placez tous vos contenus dans des variables au format texte uniquement.

L'un des aspects unique de Smarty est la compilation des templates. Celà signifie que Smarty lit les templates et crée des scripts PHP à partir de ces derniers. Une fois créés, ils sont exécutés. Il n'y a donc pas d'analyse coûteuse de template à chaque requête, et les templates peuvent bénéficier des solutions de cache PHP comme Zend Accelerator (http:// www.zend.com/) ou PHP Accelerator.

Quelques caractéristiques de Smarty :

- Il est très rapide.
- Il est efficace, le parser PHP s'occupe du sale travail.
- Pas d'analyse de template coûteuse, une seule compilation.
- Il sait ne recompiler que les fichiers de templates qui ont été modifiés.
- Vous pouvez créer des fonctions utilisateurs et des modificateurs de variables personnalisés, le langage de template est donc extrémement extensible.
- Syntaxe des templates configurable, vous pouvez utiliser {}, {{}}, <!--{}-->, etc. comme délimiteurs tag.
- Les instructions if/elseif/else/endif sont passées au parser PHP, la syntaxe de l'expression {if...} peut être aussi simple ou aussi complexe que vous le désirez.
- Imbrication illimitée de sections, de 'if', etc. autorisée.
- Il est possible d'inclure du code PHP directement dans vos templates, bien que celà ne soit pas obligatoire (ni conseillé), vû que le moteur est extensible.
- Support de cache intégré.
- Sources de templates arbitraires.

- Fonctions de gestion de cache personnalisables.
- Architecture de plugins

Chapitre 2. Installation

Table des matières

Ce dont vous avez besoin

Smarty nécessite un serveur Web utilisant PHP 4.0.6 ou supérieur.

Installation de base

Copiez les fichiers bibliothèques de Smarty du sous-dossier `/libs/` de la distribution à un emplacement accessible à PHP. Ce sont des fichiers PHP que vous NE DEVEZ PAS modifier. Ils sont partagés par toutes les applications et ne seront mis à jour que lorsque vous installerez une nouvelle version de Smarty.

Exemple 2.1. fichiers nécessaires de la bibliothèque SMARTY

```
Smarty.class.php
Smarty_Compiler.class.php
Config_File.class.php
debug.tpl
/internals/*.php (tous)
/plugins/*.php (tous doivent être sûr, peut-être votre site n'a besoin seulement que d'un sous-ensemble
```

Smarty utilise une constante [http://php.net/define] PHP appelée `SMARTY_DIR` qui représente le **chemin complet** de la bibliothèque Smarty. En fait, si votre application trouve le fichier `Smarty.class.php`, vous n'aurez pas besoin de définir la variable `SMARTY_DIR`, Smarty s'en chargera pour vous. En revanche, si `Smarty.class.php` n'est pas dans votre répertoire d'inclusion ou que vous ne donnez pas un chemin absolu à votre application, vous devez définir `SMARTY_DIR` explicitement. `SMARTY_DIR` **doit** avoir être terminé par un slash.

Exemple 2.2. Créer une instance de Smarty

Voici comment créer une instance de Smarty dans vos scripts PHP :

```
<?php
// Note : Smarty a un 'S' majuscule
require_once('Smarty.class.php');
$smarty = new Smarty();
?>
```

Essayez de lancer le script ci-dessus. Si vous obtenez une erreur indiquant que le fichier `Smarty.class.php` n'est pas trouvé, tentez l'une des actions suivantes :

Exemple 2.3. Définition manuelle de la constante SMARTY_DIR

```
<?php
// Style *nix (notez le 'S' majuscule)
define('SMARTY_DIR', '/usr/local/lib/php/Smarty-v.e.r/libs/');

// Style Windows
define('SMARTY_DIR', 'c:/webroot/libs/Smarty-v.e.r/libs/');

require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();
?>
```

Exemple 2.4. Définir le chemin absolu au fichier de la bibliothèque

```
<?php
// Style *nix (notez le 'S' majuscule)
require_once('/usr/local/lib/php/Smarty-v.e.r/libs/Smarty.class.php');

// Style Windows
require_once('c:/webroot/libs/Smarty-v.e.r/libs/Smarty.class.php');

$smarty = new Smarty();
?>
```

Exemple 2.5. Ajout du dossier contenant la bibliothèque à l'include_path de PHP

```
<?php
// Editez votre fichier php.ini, ajoutez le dossier contenant la
// bibliothèque Smarty à l'include_path et redémarrez le serveur web.
// Puis, ce qui suit devrait fonctionner :
require_once('Smarty.class.php');
$smarty = new Smarty();
?>
```

Maintenant que les fichiers de la librairie sont en place, il est temps de définir les répertoires de Smarty, pour votre application.

Smarty a besoin de quatre répertoires qui sont, par défaut, `'templates/'`, `'templates_c/'`, `'configs/'` et `'cache/'`.

Chacun d'entre eux peut être défini via les attributs `$template_dir`, `$compile_dir`, `$config_dir` et `$cache_dir` respectivement. Il est vivement conseillé que vous régliez ces répertoires séparément pour chaque application qui utilise Smarty.

Assurez-vous de bien connaître chemin de la racine de votre arborescence Web. Dans notre exemple, la racine est `/web/www.example.com/docs/`. Seul Smarty accède aux répertoires en question, et jamais le serveur Web. Pour des raisons de sécurité, il est donc conseillé de sortir ces répertoires dans un répertoire *en dehors* de l'arborescence Web.

Dans notre exemple d'installation, nous allons régler l'environnement de Smarty pour une application de livre d'or. Nous avons ici choisi une application principalement pour mettre en évidence une convention de nommage des répertoires. Vous pouvez utiliser le même environnement pour n'importe quelle autre application, il suffit de remplacer «livredor» avec le nom de votre application. Nous allons mettre nos répertoires Smarty dans `/web/www.example.com/smarty/livredor/`.

Vous allez avoir besoin d'au moins un fichier à la racine de l'arborescence Web, il s'agit du script auquel l'internaute a accès. Nous allons l'appeler *'index.php'* et le placer dans un sous-répertoire appelé `/livredor/`.

> **Technical Note:** Il est pratique de configurer le serveur Web de sorte que `index.php` soit identifié comme fichier par défaut de ce répertoire. Aicnsi, si l'on tape `http://www.example.com/livredor/`, le script `index.php` soit exécuté sans que `index.php` ne soit spécifié dans l'URL. Avec Apache, vous pouvez régler cela en ajoutant `index.php` à la ligne où se trouve `DirectoryIndex` (séparez chaque entrée par un espace) dans le `httpd.conf`.
>
> ```
> DirectoryIndex index.htm index.html index.cgi index.php
> ```

Jetons un coup d'oeil à la structure de fichier obtenue :

Exemple 2.6. Structure de fichiers

```
/usr/local/lib/php/Smarty-v.e.r/libs/Smarty.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/Smarty_Compiler.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/Config_File.class.php
/usr/local/lib/php/Smarty-v.e.r/libs/debug.tpl
/usr/local/lib/php/Smarty-v.e.r/libs/internals/*.php
/usr/local/lib/php/Smarty-v.e.r/libs/plugins/*.php

/web/www.example.com/smarty/guestbook/templates/
/web/www.example.com/smarty/guestbook/templates_c/
/web/www.example.com/smarty/guestbook/configs/
/web/www.example.com/smarty/guestbook/cache/

/web/www.example.com/docs/guestbook/index.php
```

Smarty a besoin d'**accéder en écriture** aux répertoires *$compile_dir* et *$cache_dir*, assurez-vous donc que le serveur Web dispose de ces droits d'accès. Il s'agit généralement de l'utilisateur "nobody" et du group "nobody". Pour les utilisateurs de OS X, l'utilisateur par défaut est "web" et le group "web". Si vous utilisez Apache, vous pouvez parcourir le fichier `httpd.conf` (en général dans "/usr/local/apache/conf/") pour déterminer quel est l'utilisateur et le groupe auquel il appartient.

Exemple 2.7. régler les permissions d'accès

```
chown nobody:nobody /web/www.example.com/smarty/livredor/templates_c/
chmod 770 /web/www.example.com/smarty/livredor/templates_c/

chown nobody:nobody /web/www.example.com/smarty/livredor/cache/
chmod 770 /web/www.example.com/smarty/livredor/cache/
```

> **Note:** La commande chmod 770 est relativement bien sécurisée, elle donne à l'utilisateur "nobody" et au groupe "nobody" les accès en lecture/écriture aux répertoires. Si vous voulez donner le droit d'accès en lecture à tout le monde (principalement pour pouvoir accéder vous-même à ces fichiers), vous pouvez lui préférer chmod 775.

Nous devons créer le fichier `index.tpl` que Smarty va charger. Il va se trouver dans le dossier *$template_dir*.

Exemple 2.8. Notre /web/www.example.com/smarty/templates/index.tpl

```
{* Smarty *}

Bonjour, {$name}, Bienvenue dans Smarty !
```

> **Note technique:** `{* Smarty *}` est un commentaire de template. Il n'est pas obligatoire mais il est bon de commencer tous vos templates avec ce commentaire. Celà rend le fichier facilement reconnaissable en plus de son extension. Les éditeurs de texte peuvent par exemple reconnaître le fichier et adapter la coloration syntaxique.

Maintenant passons à l'édition du fichier `index.php`. Nous allons créer une instance de Smarty, `assigner` une valeur à une variable de template et afficher le résultat avec `index.tpl`.

Exemple 2.9. Édition de /web/www.example.com/docs/livredor/index.php

```
<?php
// charge la bibliothèque Smarty
require_once(SMARTY_DIR . 'Smarty.class.php');

$smarty = new Smarty();

$smarty->template_dir = '/web/www.example.com/smarty/livredor/templates/';
$smarty->compile_dir = '/web/www.example.com/smarty/livredor/templates_c/';
$smarty->config_dir = '/web/www.example.com/smarty/livredor/configs/';
$smarty->cache_dir = '/web/www.example.com/smarty/livredor/cache/';

$smarty->assign('name','Ned');

$smarty->display('index.tpl');
?>
```

> **Note techique:** Dans notre exemple, nous avons configuré les chemins absolus pour chacun des répertoires Smarty. Si `/web/www.example.com/smarty/livredor/` est dans votre include_path PHP alors ces réglages ne sont pas nécessaires. Quoi qu'il en soit, il est plus efficace et (par expérience) moins générateur d'erreurs de les définir avec des chemins absolus. Celà nous garantit que Smarty récupèrera les bons fichiers.

Et maintenant appelez le fichier `index.php` avec navigateur Web. Vous devriez voir "Bonjour, Ned, Bienvenue dans Smarty !".

Vous venez de terminer l'installation de base de Smarty !

Configuration avancée

Ceci est la suite de l'installation de base, veuillez lire cette dernière avant de poursuivre.

Une manière un peu plus commode de configurer Smarty est de faire votre propre classe fille et de l'initialiser selon votre environnement. De la sorte, nous n'aurons plus besoin de configurer à chaques fois les chemins de notre environnement. Créons un nouveau répertoire `/php/includes/livredor/` et un nouveau fichier appelé `setup.php`. Dans notre exemple d'environnement, `/php/includes` est notre include_path PHP. Assurez-vous de faire la même chose ou alors d'utiliser des chemins absolus.

Exemple 2.10. Édition de /php/includes/livredor/setup.php

```
<?php
// charge la librairie Smarty
require('Smarty.class.php');

// le fichier setup.php est un bon
// endroit pour charger les fichiers
// de librairies de l'application et vous pouvez
// faire celà juste ici. Par exemple :
// require('livredor/livredor.lib.php');
```

```
class Smarty_livredor extends Smarty {

 function Smarty_livredor() {

 // Constructeur de la classe.
 // Appelé automatiquement à l'instanciation de la classe.

 $this->Smarty();

 $this->template_dir = '/web/www.example.com/smarty/livredor/templates/';
 $this->compile_dir = '/web/www.example.com/smarty/livredor/templates_c/';
 $this->config_dir = '/web/www.example.com/smarty/livredor/configs/';
 $this->cache_dir = '/web/www.example.com/smarty/livredor/cache/';

 $this->caching = true;
 $this->assign('app_name', 'Guest Book');
 }

}
?>
```

Modifions maintenant le fichier `index.php` pour qu'il utilise `setup.php`

Exemple 2.11. Édition de /web/www.example.com/docs/livredor/index.php

```
<?php
require('livredor/setup.php');

$smarty = new Smarty_livredor();

$smarty->assign('name','Ned');

$smarty->display('index.tpl');

?>
```

Vous savez maintenant qu'il est facile de créer une instance de Smarty, correctement configurée, en utilisant `Smarty_livredor()` qui initialise automatiquement tout ce qu'il faut pour votre application.

Partie II. Smarty pour les graphistes

Table des matières

Chapitre 3. Bases syntaxiques

Table des matières

Toutes les balises Smarty sont entourées de délimiteurs. Par défaut, ils sont { et }, mais ils peuvent être modifiés.

Pour les exemples de ce manuel, nous supposons que vous utiliserez leur valeur par défaut. Dans Smarty, le contenu qui est situé en dehors des délimiteurs est affiché comme contenu statique, inchangé. Lorsque Smarty rencontre des balises de template, il tente de les comprendre et en affiche la sortie appropriée, en lieu et place.

Commentaires

Les commentaires dans Smarty sont entourés d'asterisques, et entourés par le délimiteurs de cette façon :

```
{* ceci est un comentaire *}
```

Les commentaires Smarty ne sont PAS affichés dans la sortie finale du template, différemment des `<!-- commentaires HTML -->`. Ils sont utilisés pour des notes internes, dans le template que personne ne verra ;)

Exemple 3.1. Commentaires dans un template

```
{* Je suis un commentaire Smarty, je n'existe pas dans la sortie compilée *}
<html>
 <head>
  <title>{$title}</title>
 </head>
 <body>

{* un autre commentaire Smarty sur une seule ligne *}
<!-- Un commentaire Html qui sera envoyé au navigateur -->

{* ces multi-lignes sont des commentaires
qui ne sont pas envoyées au navigateur
*}

{************************************************************
Un bloc de commentaires multilignes contenant les crédits
@ author:         bg@example.com
@ maintainer:     support@example.com
@ para:           var that sets block style
@ css:            the style output
************************************************************}

{* Inclusion du fichier d'en-tête contenant le logo principal *}
{include file='header.tpl'}

{* Note aux développeurs :  $includeFile est assigné au script foo.php  *}
```

```
<!-- Affichage du bloc principal -->
{include file=$includeFile}

{* Ce block <select> est redondant *}
{*
<select name="company">
 {html_options options=$vals selected=$selected_id}
</select>
*}

<!-- L'affichage de l'en-tête est désactivé -->
{* $affiliate|upper *}

{* Vous ne pouvez pas imbriquer des commentaires *}
{*
<select name="company">
  {* <option value="0">-- none -- </option> *}
  {html_options options=$vals selected=$selected_id}
</select>
*}

{* Balise cvs pour un template, ci-dessous, le 36 DOIT ÊTRE une devise américaine sinon,
il sera converti en cvs.. *}
{* &#36;Id: Exp &#36; *}
{* $Id: *}
</body>
</html>
```

Variables

Les variables de template commence par un signe dollar ($). Elles peuvent contenir des nombres, des lettres et des underscores, tout comme une variable PHP [http://php.net/language.variables]. Vous pouvez référencer des tableaux indexés numériquement ou non. Vous pouvez aussi référencer des propriétés d'objet ainsi que des méthodes.

Les variables des fichiers de configuration sont une exception à la synthaxe utilisant un signe dollar. Elles peuvent être référencées en les entourant du signe dièse (#) ou via la variable spéciale *$smarty.config*.

Exemple 3.2. Variables

```
{$foo}        <-- affiche une variable simple (qui n'estpas un tableau ou un objet)
{$foo[4]}     <-- affiche le 5ème élément d'un tableau indexé
{$foo.bar}    <-- affiche la clé "bar" d'un tableau, identique à $foo['bar'] en PHP
{$foo.$bar}   <-- affiche la valeur de la clé d'un tableau, identique à $foo[$bar] en PHP
{$foo->bar}   <-- affiche la propriété "bar" de l'objet
{$foo->bar()} <-- affiche la valeur retournée de la méthode "bar" de l'objet
{#foo#}       <-- affiche la variable du fichier de configuration "foo"
{$smarty.config.foo} <-- synonyme pour {#foo#}
{$foo[bar]}   <-- synthaxe uniquement valide dans une section de boucle, voir {section}
{assign var=foo value='baa'}{$foo} <--  affiche "baa", voir {assign}

Plusieurs autres combinaisons sont autorisées

{$foo.bar.baz}
{$foo.$bar.$baz}
{$foo[4].baz}
{$foo[4].$baz}
{$foo.bar.baz[4]}
{$foo->bar($baz,2,$bar)} <-- passage de paramètres
{"foo"}       <-- les valeurs statiques sont autorisées

{* affiche la variable serveur "SERVER_NAME" ($_SERVER['SERVER_NAME'])*}
{$smarty.server.SERVER_NAME}
```

Les variables spéciales comme `$_GET`, `$_SESSION`, etc. sont également disponibles, lisez le chapitre sur les variables réservées *$smarty* pour plus de détails.

Voir aussi *$smarty*, les variables de configuration, `{assign}` et `assign()`.

Fonctions

Les balises Smarty affichent une variable ou invoquent une fonction. Elles sont appelées lorsqu'elles sont entourées, ainsi que leurs paramètres, des délimiteurs Smarty. Par exemple : `{nomfonction attr1='val' attr2='val'}`.

Exemple 3.3. syntaxe des fonctions

```
{config_load file='colors.conf'}
{include file='header.tpl'}
{insert file='banner_ads.tpl' title='Smarty est cool !'}

{if $logged_in}
   Bonjour, <font color="{#fontColor#}">{$name}!</font>
{else}
   Bonjour, {$name}!
{/if}

{include file='footer.tpl' ad=$random_id}
```

- Les fonctions natives et les fonctions utilisateurs ont toutes deux la même syntaxe, dans les templates.
- Les fonctions natives sont relatives au traitement **interne** de Smarty, comme `{if}`, `{section}` et `{strip}`. Il n'y a aucune raison à ce qu'elles soient modifiées ou changées.
- Les fonctions utilisateurs sont des fonctions **additionnelles**, implémentées par l'intermédiaire de plugins. Elles peuvent être modifiées pour correspondre à vos besoins, et vous pouvez en créer de nouvelles. `{html_options}` et `{popup}` sont deux exemples de fonctions utilisateurs.

Voir aussi `register_function()`.

Paramètres

La plupart des fonctions attendent des paramètres qui régissent leur comportement. Les paramètres des fonctions Smarty sont très proches des attributs des balises HTML. Les valeurs numériques n'ont pas besoin d'être entourées par des guillemets, par contre, ces guillemets sont recommandées lors de l'utilisation de chaînes de caractères. Des variables peuvent aussi être utilisées en tant que paramètres, et ne doivent pas être entourées de guillemets.

Certains paramètres requièrent des valeurs booléennes (`TRUE` ou `FALSE`). Elles peuvent être spécifiées par l'une des valeures suivantes, sans guillemet: `true`, `on`, et `yes`, ou `false`, `off`, et `no`.

Exemple 3.4. Paramètres de fonction, syntaxe

```
{include file='header.tpl'}

{include file='header.tpl' attrib_name='attrib value'}

{include file=$includeFile}
```

```
{include file=#includeFile# title='Smarty est cool'}

{html_select_date display_days=yes}

{mailto address='smarty@example.com'}

<select name='company'>
 {html_options options=$choices selected=$selected}
</select>
```

Variables insérées dans des chaînes de caractères

- Smarty est capable d'interpréter les variables assignées à l'intérieur de chaînes entre guillemets, du moment que leur nom est exclusivement composé de chiffres, lettres, underscores et crochets Voir le nommage [http:/ / php.net/ language.variables] pour plus de détails.
- Si le nom de la variable contient tout autre caractère (point, référence à un objet, etc.) la variable doit être entourée d'apostrophes inverses (`).
- Vous ne pouvez jamais insérer de modificateurs, ils doivent toujours être appliquer à l'extérieur des guillemets.

Exemple 3.5. Exemples de synthaxes

```
{func var="test $foo test"}       <-- comprends $foo
{func var="test $foo_bar test"}   <-- comprends $foo_bar
{func var="test $foo[0] test"}    <-- comprends $foo[0]
{func var="test $foo[bar] test"}  <-- comprends $foo[bar]
{func var="test $foo.bar test"}   <-- comprends $foo (not $foo.bar)
{func var="test `$foo.bar` test"} <-- comprends $foo.bar
{func var="test `$foo.bar` test"|escape} <-- modifieurs à l'extérieur des guillemets !
```

Exemple 3.6. Exemples pratiques

```
{* remplacera $tpl_name par la valeur *}
{include file="subdir/$tpl_name.tpl"}

{* ne remplacera pas $tpl_name *}
{include file='subdir/$tpl_name.tpl'} <--

{* doit contenir des apostophes inverses car il contient un . *}
{cycle values="one,two,`$smarty.config.myval`"}

{* identique à $module['contact'].'.tpl' dans un script PHP
{include file="`$module.contact`.tpl"}

{* identique à $module[$view].'.tpl' dans un script PHP
{include file="$module.$view.tpl"}
```

Voir aussi `escape`.

Opérations mathématiques

Les opérations mathématiques peuvent être directement appliquées aux variables.

Exemple 3.7. Exemples d'opérations mathématiques

```
{$foo+1}

{$foo*$bar}

{* quelques exemples plus compliqués *}

{$foo->bar-$bar[1]*$baz->foo->bar()-3*7}

{if ($foo+$bar.test%$baz*134232+10+$b+10)}

{$foo|truncate:"`$fooTruncCount/$barTruncFactor-1`"}

{assign var="foo" value="`$foo+$bar`"}
```

Voir aussi la fonction `{math}` pour les équations complexes et `{eval}`.

Désactiver l'analyse de Smarty

Il est quelques fois utile, voir nécessaire, de demander à Smarty d'ignorer certaines sections que seraient analysées sinon. Un exemple classique est l'incorporation de code Javascript ou CSS dans les templates. Le problème est que ces langages utilisent les caractères { et }, qui sont aussi les délimiteurs Smarty par défaut.

Le plus simple pour éviter une telle situation est de placer vos codes Javascript et CSS dans des fichiers séparés, puis d'utiliser les méthodes standards HTML pour y accéder.

Inclure du contenu tel quel est possible en utilisant les blocs `{literal} .. {/literal}`. Similairement à l'utilisation d'entités HTML, vous pouvez utiliser `{ldelim}` et `{rdelim}`, ou `{$smarty.ldelim}` pour afficher les délimiteurs.

Il est souvent plus simple de modifier les délimiteurs de Smarty : *$left_delimiter* et *$right_delimiter*.

Exemple 3.8. Exemple de changement de délimiteur

```
<?php

$smarty->left_delimiter = '<!--{';
$smarty->right_delimiter = '}-->';

$smarty->assign('foo', 'bar');
$smarty->assign('name', 'Albert');
$smarty->display('example.tpl');

?>
```

Où le template est:

```
Bienvenue <!--{$name}--> sur Smarty
<script language="javascript">
var foo = <!--{$foo}-->;
function dosomething() {
    alert("foo = " + foo);
}
dosomething();
</script>
```

Chapitre 4. Variables

Table des matières

Smarty possède différents types de variables. Le type de ces variables dépend du symbole qui les préfixe, ou des symboles qui les entourent.

Les variables de Smarty peuvent être soit affichées directement, soit utilisées comme arguments pour les fonctions et modificateurs, à l'intérieur d'expressions conditionnelles, etc. Pour afficher une variable, il suffit de l'entourer par des délimiteurs de telle sorte qu'elle soit la seule chose qu'ils contiennent.

Exemple 4.1. Exemple de variables

```
{$Nom}

{$Contacts[enreg].Telephone}

<body bgcolor="{#bgcolor#}">
```

Astuce: La façon de la simple d'analyser les variables Smarty est d'utiliser la console de débogage.

Variables assignées depuis PHP

Pour utiliser une variables assignées depuis PHP, il faut la préfixer par le symbole dollar `$`. Les variables asignées depuis un template grâce à la fonction `{assign}` sont manipulées de la même façon.

Exemple 4.2. Variables assignées

Script PHP

```
<?php

$smarty = new Smarty;

$smarty->assign('firstname', 'Doug');
$smarty->assign('lastname', 'Evans');
$smarty->assign('meetingPlace', 'New York');

$smarty->display('index.tpl');

?>
```

Où `index.tpl` est :

```
Bonjour {$firstname} {$lastname}, heureux de voir que tu es arrivé ici.
<br />
```

```
{* ceci ne fonctionnera pas car $vars est sensible à la casse *}
Cette semaine, le meeting est à {$meetingplace}.
{* ceci fonctionnera *}
Cette semaine, le meeting est à {$meetingPlace}.
```

Affichera :

```
Bienvenue Doug, heureux de voir que tu es arrivé ici.
<br />
Cette semaine, le meeting est à .
Cette semaine, le meeting est à New York.
```

Tableaux associatifs

Vous pouvez également utiliser des variables sous forme de tableaux associatifs assignées depuis PHP en en spécifiant la clef, après le symbole '.' (point).

Exemple 4.3. Accéder aux variables de tableaux associatifs

```
<?php

$smarty->assign('Contacts',
    array('fax' => '555-222-9876',
          'email' => 'zaphod@slartibartfast.example.com',
          'phone' => array('home' => '555-444-3333',
                           'cell' => '555-111-1234')
                          )
           );
$smarty->display('index.tpl');
?>
```

Où `index.tpl` est :

```
{$Contacts.fax}<br />
{$Contacts.email}<br />
{* vous pouvez afficher des tableaux de tableaux *}
{$Contacts.phone.home}<br />
{$Contacts.phone.cell}<br />
```

Affichera :

```
555-222-9876<br />
zaphod@slartibartfast.example.com<br />
555-444-3333<br />
555-111-1234<br />
```

Tableaux indexés

Vous pouvez utiliser des tableaux indexés de la même façon que vous le faites en PHP.

Exemple 4.4. Accès aux tableaux grâce à l'index

```
<?php
$smarty->assign('Contacts', array(
```

```
                    '555-222-9876',
                    'zaphod@slartibartfast.example.com',
                      array('555-444-3333',
                            '555-111-1234')
                      ));
$smarty->display('index.tpl');
?>
```

Où `index.tpl` est :

```
{$Contacts[0]}<br />
{$Contacts[1]}<br />
{* Vous pouvez également afficher des tableaux *}
{$Contacts[2][0]}<br />
{$Contacts[2][1]}<br />
```

Affichera :

```
555-222-9876<br />
zaphod@slartibartfast.example.com<br />
555-444-3333<br />
555-111-1234<br />
```

Objets

Les attributs des objets assignés depuis PHP peuvent être utilisées en en spécifiant le nom après le symbole `->`.

Exemple 4.5. Accéder aux attributs des objets

```
nom: {$person->name}<br />
email: {$person->email}<br />
```

Affichera :

```
nom: Zaphod Beeblebrox<br />
email: zaphod@slartibartfast.example.com<br />
```

Variables chargées depuis des fichiers de configuration

Les variables récupérées depuis un fichier de configuration sont utilisées entourées du symbole dièse (#), ou via la variable spéciale smarty *$smarty.config*. La dernière synthaxe est utile pour mettre entre guillemets les valeurs des attributs.

Exemple 4.6. variables de fichiers de configuration

Exemple de fichier de configuration - `foo.conf` :

```
pageTitle = "C'est le mien"
bodyBgColor = '#eeeeee'
tableBorderSize = 3
tableBgColor = '#bbbbbb'
rowBgColor = '#cccccc'
```

Exemple de template :

```
{config_load file='foo.conf'}
<html>
<title>{#pageTitle#}</title>
<body bgcolor="{#bodyBgColor#}">
<table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
<tr bgcolor="{#rowBgColor#}">
<td>First</td>
<td>Last</td>
<td>Address</td>
</tr>
</table>
</body>
</html>
```

Un template démontrant la méthode *$smarty.config* :

```
{config_load file="foo.conf"}
<html>
<title>{$smarty.config.pageTitle}</title>
<body bgcolor="{$smarty.config.bodyBgColor}">
<table border="{$smarty.config.tableBorderSize}" bgcolor="{$smarty.config.tableBgColor}">
<tr bgcolor="{$smarty.config.rowBgColor}">
<td>First</td>
<td>Last</td>
<td>Address</td>
</tr>
</table>
</body>
</html>
```

Les deux exemples ci-dessus afficheront :

```
<html>
<title>C'est le mien</title>
<body bgcolor="#eeeeee">
<table border="3" bgcolor="#bbbbbb">
<tr bgcolor="#cccccc">
<td>First</td>
<td>Last</td>
<td>Address</td>
</tr>
</table>
</body>
</html>
```

Les variables de fichier de configuration ne peuvent être utilisés tant qu'elles n'ont pas été chargées. Cette procédure est expliquée plus loin dans le document, voir `{config_load}`.

Voir aussi les variables et les variables réservées $smarty.

Variable réservée {$smarty}

La variable PHP réservée *{$smarty}* peut être utilisée pour accéder à plusieurs variables d'environnements. En voici la liste complète.

Variables de requête

Les variables de requête [http:// / php.net/ reserved.variables] comme `$_GET`, `$_POST`, `$_COOKIE`, `$_SERVER`, `$_ENV` et `$_SESSION` (voir `$request_vars_order` et `$request_use_auto_globals`) peuvent être utilisées comme dans

l'exemple suivant :

Exemple 4.7. Afficher des variables de requête

```
{* Affiche la valeur de page dans l'url ($_GET) http://www.example.com/index.php?page=foo *}
{$smarty.get.page}

{* affiche la variable "page" récupérée depuis un formulaire ($_POST['page']) *}
{$smarty.post.page}

{* affiche la valeur du cookie "utilisateur" ($_COOKIE['username']) *}
{$smarty.cookies.utilisateur}

{* affiche la variable serveur "SERVER_NAME" ($_SERVER['SERVER_NAME']) *}
{$smarty.server.SERVER_NAME}

{* affiche la variable d'environnement "PATH" *}
{$smarty.env.PATH}

{* affiche la variable de session PHP "id" ($_SESSION['id']) *}
{$smarty.session.id}

{* affiche la variable "utilisateur" du regroupement de get/post/cookies/server/env *}
{$smarty.request.utilisateur}
```

> **NOTE:** Pour des raisons historiques, *{$SCRIPT_NAME}* peut être accédé directement, cependant, *{$smarty.server.SCRIPT_NAME}* est la solution proposée pour accéder à cette valeur.
>
> ```
> <a href="{$SCRIPT_NAME}?page=smarty">click me</a>
> <a href="{$smarty.server.SCRIPT_NAME}?page=smarty">click me</a>
> ```

{$smarty.now}

Le timestamp [http://php.net/function.time] courant peut être récupéré grâce à *{$smarty.now}*. La valeur correspond au nombre de secondes écoulées depuis Epoch (1 Janvier 1970) et peut être passé directement au modificateur de variable date `date_format` à des fins d'affichage. Notez que `time()` [http://php.net/function.time] est appelé à chaque invocation, i.e. un script qui prend 3 secondes à s'exécuter avec *$smarty.now* au début et à la fin montrera les 3 secondes de différence.

Exemple 4.8. Utilisation de {$smarty.now}

```
{* utilise le modificateur de variable date_format pour afficher la date et heure *}
{$smarty.now|date_format:'%d-%m-%Y %H:%M:%S'}
```

{$smarty.const}

Vous pouvez directement accéder aux constantes PHP. Voir aussi les constantes smarty.

```
// la constante définie dans PHP
define('_MY_CONST_VAL','CHERRIES');
```

Affiche la constante dans un template comme :

```
{* la sortie de la constante PHP dans le template *}
{$smarty.const._MA_CONSTANTE_}
```

{$smarty.capture}

La sortie du template réalisée via `{capture}..{/capture}` peut être récupérée par l'intermédiaire de la variable *{$smarty.capture}*. Voir la section sur `{capture}` pour un exemple à ce sujet.

{$smarty.config}

La variable *{$smarty.config}* peut être utilisée pour désigner une variable d'un fichier de configuration. *{$smarty.config.foo}* est un synonyme de *{#foo#}*. Voir la section {config_load} pour un exemple à ce sujet.

{$smarty.section}, {$smarty.foreach}

La variable `{section}` peut être utilisée pour accéder aux propriétés des boucles *{$smarty.section}* et *{$smarty.foreach}*. Voir la documentation de `{section}` et `{foreach}`. Ils ont des valeurs vraiment utiles comme `.first`, `.index`, etc.

{$smarty.template}

Retourne le nom du template courant. Cet exemple montre le `container.tpl` ainsi que le `banner.tpl` inclu avec *{$smarty.template}*.

```
<b>Le conteneur principal est {$smarty.template}</b>
{include file='banner.tpl'}
```

Affichera :

```
<b>Le conteneur principal est container.tpl</b>
banner.tpl
```

{$smarty.version}

Retourne la version de Smarty ayant servie à compiler le template.

```
<div id="footer">Généré par Smarty {$smarty.version}</div>
```

{$smarty.ldelim}, {$smarty.rdelim}

Ces variables sont utilisées pour afficher le délémiteur gauche et le délimiteur droit. Lisez aussi la partie `{ldelim},{rdelim}`.

Voir aussi les variables et les variables de configuration.

Chapitre 5. Modificateurs de variables

Table des matières

Les modificateurs de variables peuvent être appliqués aux variables, fonctions utilisateurs ou chaînes de caractères. Pour appliquer un modificateur de variable, tappez une valeure suivie de | (pipe) et du nom du modificateur. Un modificateur de variable est succeptible d'accepter des paramètres additionnels, qui en affectent le comportement. Ces paramètres suivent le nom du modificateur et sont séparés par un : (deux points). *Toutes les fonctions PHP peuvent être utilisées en tant que modifieurs implicitement* (plus d'informations ci-dessous) et les modificateurs peuvent être combinés.

Exemple 5.1. Exemple de modificateur

```
{* applique un modificateur à une variable *}
{$titre|upper}

{* modificateur avec paramètres *}
{$titre|truncate:40:'...'}

{* applique un modificateur à un paramètre de fonction *}
{html_table loop=$mavariable|upper}
{* avec paramètres *}
{html_table loop=$mavariable|truncate:40:'...'}

{* applique un modificateur à une chaine de caractères *}
{'foobar'|upper}

{* utilise date_format pour mettre en forme la date *}
{$smarty.now|date_format:"%d/%m/%Y"}

{* applique un modificateur à une fonction utilisateur *}
{mailto|upper address='smarty@example.com'}

{* utilisation de la fonction PHP str_repeat *}
```

```
{'='|str_repeat:80}

{* Compteur PHP *}
{$myArray|@count}

{* mélange aléatoire des IP du serveur grâce à PHP *}
{$smarty.server.SERVER_ADDR|shuffle}

(* ceci va mettre en majuscule et tronque le tableau *}
<select name="name_id">
 {html_options output=$myArray|upper|truncate:20}
</select>
```

- Si vous appliquez un modificateur de variable à un tableau plutôt qu'à une variable simple, le modificateur sera appliqué à chaque valeur du tableau. Si vous souhaitez que le modificateur travaille réellement avec le tableau en tant que tel, vous devez préfixer le nom du modificateur avec un symbole @

  Exemple: `{$articleTitle|@count}` - affichera le nombre d'éléments dans le tableau *$articleTitle* en utilisant la fonction PHP `count()` [http://php.net/count] comme modificateur.

- Les modificateurs sont chargés automatiquement depuis votre répertoire de plugin *$plugins_dir* ou peuvent être enregistrés explicitement avec `register_modifier()` ; ceci est utile pour partager une fonction dans un scirpt PHP et les templates Smarty.

- Toutes les fonction PHP peuvent être utilisées comme modificateur, sans autre déclaration, tel que dans l'exemple ci-dessus. Cepdendant, l'utilisation de fonctions PHP comme modificateurs contient deux petits pièges à éviter :

  - Le premier - quelques fois, l'ordre des paramètres de la fonction n'est pas celui attendu. Le formattage de `$foo` avec `{"%2.f"|sprintf:$foo}` fonctionne actuellement, mais n'est pas aussi intuitif que `{$foo|string_format:"%2.f"}`, ce qui est fournit par Smarty.

  - Le deuxième - Si *$security* est activé, toutes les fonctions PHP qui devront être utilisées comme modificateurs, doivent être déclarées dans l'élément `MODIFIER_FUNCS` du tableau *$security_settings*.

Voir aussi `register_modifier()`, les modificateurs combinés. et étendre Smarty avec des plugins.

capitalize

Met la première lettre de chaque mot d'une variable en majuscule. C'est l'équivalent de la fonction PHP `ucfirst()` [http://php.net/ucfirst].

Position du paramètre	Type	Requis	Défaut	Description
1	booléen	No	`FALSE`	Détermine si oui ou non les mots contenant des chiffres doivent être mis en majuscule

Exemple 5.2. Mise en majuscule

```
<?php
```

```
$smarty->assign('titreArticle', 'Le nouveau php5 est vraiment performant !');

?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|capitalize}
{$titreArticle|capitalize:true}
```

Affichera :

```
Le nouveau php5 est vraiment performant !
Le Nouveau php5 Est Vraiment Performant !
Le Nouveau Php5 Est Vraiment Performant !
```

Voir aussi `lower` et `upper`.

cat

Cette valeur est concaténée à la variable donnée.

Position du paramètre	Type	Requis	cat	Description
1	chaîne de caractères	Non	*empty*	Valeur à concaténer à la variable donnée.

Exemple 5.3. cat

```
<?php

$smarty->assign('articleTitle', "'Les devins ont prévus que le monde existera toujours");

?>
```

Où le template est :

```
{$articleTitle|cat:' demain.'}
```

Affichera :

```
Les devins ont prévus que le monde existera toujours demain.
```

count_characters

Compte le nombre de caractères dans une variable.

Position du paramètre	Type	Requis	Defaut	Description
1	boolean	Non	FALSE	Si l'on doit inclure les espaces dans le compte.

Exemple 5.4. count_characters

```
<?php

$smarty->assign('titreArticle', 'Vagues de froid liées à la température.');

?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|count_characters}
{$titreArticle|count_characters:true}
```

Ce qui donne en sortie :

```
Vagues de froid liées à la température.
33
39
```

Voir aussi count_words, count_sentences et count_paragraphs.

count_paragraphs

Compte le nombre de paragraphes dans une variable.

Exemple 5.5. count_paragraphs

```
<?php

$smarty->assign('articleTitle',
               "War Dims Hope for Peace. Child's Death Ruins Couple's Holiday.\n\n
               Man is Fatally Slain. Death Causes Loneliness, Feeling of Isolation."
             );
?>
```

Où le template est :

```
{$TitreArticle}
{$TitreArticle|count_paragraphs}
```

Affichera :

```
La guerre apporte la paix, au prix de la vie des innocents.
1
```

Voir aussi `count_characters`, `count_sentences` et `count_words`.

count_sentences

Compte le nombre de phrases dans une variable.

Exemple 5.6. count_sentences

```
<?php

$smarty->assign('articleTitle',
                'Two Soviet Ships Collide - One Dies.
               Enraged Cow Injures Farmer with Axe.'
             );

?>
```

Où le template est :

```
{$TitreArticle}
{$TitreArticle|count_sentences}
```

Affichera :

```
Deux navires rentrent en collision
- Un des deux coule. Des vaches enragées blessent un fermier
à coups de haches.
2
```

Voir aussi `count_characters`, `count_paragraphs` et `count_words`.

count_words

Compte le nombre de mots dans une variable.

Exemple 5.7. count_words

```
<?php

$smarty->assign('TitreArticle', 'Un anneau pour les gouverner tous.');

?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|count_words}
```

Affichera :

```
Un anneau pour les gouverner tous.
6
```

Voir aussi `count_characters`, `count_paragraphs` et `count_sentences`.

date_format

Formate une date / heure au format `strftime()` [http://php.net/strftime] donné. Les dates peuvent être passées à smarty en tant que timestamp [http://php.net/function.time] unix, timestamp mysql ou comme chaîne quelconque contenant mois jour année (interprétable par `strtotime()` [http:/ / php.net/ strtotime]). Les concepteurs de templates peuvent utiliser `date_format` pour contrôler parfaitement le format de sortie de la date. Si la date passée à `date_format` est vide, et qu'un second paramètre est donné, ce dernier sera utilisé comme étant la date à formater.

Position du paramètre	Type	Requis	Défaut	Description
1	chaîne de caractères	Non	%b %e, %Y	Format de sortie de la date.
2	chaîne de caractères	Non	n/a	Date par défaut si aucune n'est spécifiée en entrée.

NOTE: Depuis Smarty 2.6.10, les valeurs numériques passées à `date_format` sont *toujours* (excepté pour les timestamps mysql, voir ci-dessous) interprétées comme un timestamp Unix.

Avant la version 2.6.10 de Smarty, les chaînes numériques qui étaient également analysables par `strtotime()` en PHP (comme `YYYYMMDD`), étaient, parfois, dépendamment de l'implémentation de `strtotime()`, interprétées en tant que des chaînes date et NON des timestamps.

La seule exception est les timestamps MySQL : Ils sont uniquement numériques et d'une longueur de 14 caractères (`YYYYMMDDHHMMSS`). Les timestamps MySQL ont la priorité sur les timestamps Unix.

Note pour les développeurs: `date_format` est essentiellement un gestionnaire pour la fonction PHP `strftime()` [http://php.net/strftime]. Vous pourriez avoir plus ou moins d'options disponibles suivant le système sur lequel la fonction PHP `strftime()` [http://php.net/strftime] a été compilé. Vérifiez la documentation pour votre système pour avoir une liste complète des options disponibles.

Exemple 5.8. date_format

```
<?php

$config['date'] = '%I:%M %p';
$config['time'] = '%H:%M:%S';
$smarty->assign('config',$config);
$smarty->assign('hier', strtotime('-1 day'));

?>
```

Où le template est (utilisation de *$smarty.now*) :

```
{$smarty.now|date_format}
{$smarty.now|date_format:"%D"}
{$smarty.now|date_format:$config.date}
{$yesterday|date_format}
{$yesterday|date_format:"%A, %B %e, %Y"}
{$yesterday|date_format:$config.time}
```

Affichera :

```
Jan 1, 2022
02/06/01
02:33 pm
Dec 31, 2021
Monday, December 1, 2021
14:33:00
```

Conversion de **date_format** :

- %a - Abréviation du jour de la semaine, selon les paramètres locaux.
- %A - Nom du jour de la semaine, selon les paramètres locaux.
- %b - Abréviation du nom du jour, selon les paramètres locaux.
- %B - Nom complet du mois, selon les paramètres locaux.
- %c - Préférences d'affichage selon les paramètres locaux.
- %C - Siècle, (L'année divisée par 100 et tronquée comme un entier, de 00 à 99)
- %d - Jour du mois, en tant que nombre décimal (de 01 à 31)
- %D - même chose que %m/%d/%y
- %e - Jour du mois en tant que nombre décimal. Un chiffre unique est précédé par un espace (de 1 à 31)
- %g - Position de la semaine dans le siècle [00,99]
- %G - Position de la semaine, incluant le siècle [0000,9999]
- %h - identique à %b
- %H - L'heure en tant que décimale, en utilisant une horloge sur 24 (de 00 à 23)
- %I - L'heure en tant que décimale en utilisant une horloge sur 12 (de 01 to 12)
- %j - jour de l'année (de 001 à 366)
- %k - Heure (horloge sur 24). Les numéros à un chiffre sont précédés d'un espace. (de 0 à 23)
- %l - Heure (horloge sur 12). Les numéros à un chiffre sont précédés d'un espace. (de 1 à 12)
- %m - Mois en tant que nombre décimal (de 01 à 12)
- %M - Minute en tant que nombre décimal
- %n - Retour chariot (nouvelle ligne).
- %p - soit am soit pm selon l'heure donnée, ou alors leurs correspondances locales.
- %r - heure en notation a.m. et p.m.
- %R - Heure au format 24 heures
- %S - Secondes en tant que nombre décimal.
- %t - Caractère tabulation.

- %T - Heure courante, équivalent à %H:%M:%S
- %u - Jour de la semaine en tant que nombre décimal [1,7], ou 1 représente le lundi.
- %U - Le numéro de la semaine en nombre décimal, utilisant le premier dimanche en tant que premier jour de la première semaine.
- %V - Le numéro de la semaine de l'année courante selon la norme ISO 8601:1988, de 01 à 53, ou la semaine 1 est la première semaine qui dispose au minimum de 4 jours dans l'année courante et ou Lundi est le premier jour de cette semaine.
- %w - Jour de la semaine en tant que nombre décimal, dimanche étant 0
- %W - Le numéro de la semaine de l'année courante en tant que nombre décimal, ou Lundi est le premier jour de la première semaine.
- %x - Représentation préférée de la date selon les paramètres locaux.
- %X - Représentation préférée de l'heure selon les paramètres locaux, sans la date.
- %y - L'année en tant que nombre décimal, sans le siècle. (de 00 à 99)
- %Y - L'année en tant que nombre décimal, avec le siècle.
- %Z - Zone horraire, nom ou abréviation
- %% - Un caractère litéral `%'

Voir aussi *$smarty.now*, `strftime()` [http://php.net/strftime], `{html_select_date}` et les astuces sur les dates.

default

Utilisé pour définir une valeur par défaut à une variable. Si la variable est vide ou indéfinie, la valeur donnée est affichée en lieu et place. `default` attends un seul argument.

> **NOTE:** Avec `error_reporting(E_ALL)` [http://php.net/error_reporting], les variables non déclarées lanceront toujours une erreur dans le template. Cette fonction est utile pour remplacer les chaînes vides ou de longueurs vides.

Position du paramètre	Type	Requis	Defaut	Description
1	chaîne de caractères	Non	*empty*	La valeur par défaut de la sortie si la variable d'entrée est vide.

Exemple 5.9. Défaut

```
<?php

$smarty->assign('TitreArticle', 'Les portes de la moria restent fermées.');
$smarty->assign('email','');

?>
```

Où le template est :

```
{$articleTitle|default:'Aucun titre'}
{$myTitle|default:'Aucun titre'}
{$email|default:'Aucune adresse email disponible'}
```

Affichera :

```
Les portes de la moria restent fermées.
Aucun titre
Aucune adresse email disponible
```

Voir aussi la gestion des variables par défaut et la gestion de l'effacement des variables.

escape

`escape` est utilisé pour encoder / échapper une variable pour quelles soient compatibles avec les `url` `html`, avec les hexadécimaux, avec les entités hexadécimales, avec `javascript` et avec les e-mails. Par défaut, ce paramètre est `html`.

Position du paramètre	Type	Requis	Valeurs possibles	Défaut	Description
1	chaîne de caractère	Non	`html`, `htmlall`, `url`, `urlpathinfo`, `quotes`, `hex`, `hexentity`, `javascript`, `mail`	`html`	Format d'échappement à utiliser.
2	chaîne de caractère	Non	`ISO-8859-1`, `UTF-8`, ... n'importe quel jeu de caractères supporté par `htmlentities()` [http://php.net/htmlentities]	`ISO-8859-1`	Le jeu de caractères passé à htmlentities()

Exemple 5.10. escape

```
<?php

$smarty->assign('articleTitle',
              "'Stiff Opposition Expected to Casketless Funeral Plan'"
              );
$smarty->assign('EmailAddress','smarty@example.com');
?>
```

Voici des exemples de template avec `escape` suivis par l'affichage produit.

```
{$articleTitle}
'Stiff Opposition Expected to Casketless Funeral Plan'

{$articleTitle|escape}
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;
```

```
{$articleTitle|escape:'html'}    {* échappe les caractères  & " ' < > *}
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;

{$articleTitle|escape:'htmlall'} {* échappe toutes les entités html *}
&#039;Stiff Opposition Expected to Casketless Funeral Plan&#039;

<a href="?title={$articleTitle|escape:'url'}">cliquez-ici</a>
<a href="?title=%27Stiff+Opposition+Expected+to+Casketless+Funeral+Plan%27">cliquez-ici</a>

{$articleTitle|escape:'quotes'}
\'Stiff Opposition Expected to Casketless Funeral Plan\'

<a href="mailto:{$EmailAddress|escape:"hex"}">{$EmailAddress|escape:"hexentity"}</a>
{$EmailAddress|escape:'mail'}    {* ceci convertit un email en texte *}
<a href="mailto:%62%6f%..snip..%65%74">&#x62;&#x6f;&#x62..snip..&#x65;&#x74;</a>

{'mail@example.com'|escape:'mail'}
smarty [AT] example [DOT] com
```

Exemple 5.11. Autres exemples

Les fonctions PHP peuvent être utilisées comme modificateurs, suivant la configuration de `$security`.

```
{* le paramètre "rewind" enregistre l'emplacement courant *}
<a href="{$SCRIPT_NAME}?page=foo&rewind={$smarty.server.REQUEST_URI|urlencode}">click here</a>
```

Et ceci est utile pour les e-mails, mais lisez plutôt la documentation de `{mailto}`

```
{* email address mangled *}
<a href="mailto:{$EmailAddress|escape:'hex'}">{$EmailAddress|escape:'mail'}</a>
```

Voir aussi la l'anayse Smarty d'échappement, `{mailto}` et le mascage des adresses e-mail.

indent

Indente chacune des lignes d'une chaîne. Comme paramètre optionnel, vous pouvez spécifier le nombre de caractères à utiliser pour l'indentation (4 par défaut). Comme second paramètre optionnel, vous pouvez spécifier le caractère à utiliser pour l'indentation (utilisez `"\t"` pour les tabulations).

Position du paramètre	Type	Requis	Defaut	Description
1	entier	Non	4	De combien de caractères l'indentation doit être effectuée.
2	chaîne de caractère	Non	(espace)	Caractère à utiliser pour l'indentation.

Exemple 5.12. indent

```
<?php

$smarty->assign('articleTitle',
```

```
                'NJ judge to rule on nude beach.
Sun or rain expected today, dark tonight.
Statistics show that teen pregnancy drops off significantly after 25.'
                );
?>
```

Où le template est :

```
{$TitreArticle}

{$TitreArticle|indent}

{$TitreArticle|indent:10}

{$TitreArticle|indent:1:"\t"}
```

Affichera :

```
NJ judge to rule on nude beach.
Sun or rain expected today, dark tonight.
Statistics show that teen pregnancy drops off significantly after 25.

    NJ judge to rule on nude beach.
    Sun or rain expected today, dark tonight.
    Statistics show that teen pregnancy drops off significantly after 25.

          NJ judge to rule on nude beach.
          Sun or rain expected today, dark tonight.
          Statistics show that teen pregnancy drops off significantly after 25.

        NJ judge to rule on nude beach.
        Sun or rain expected today, dark tonight.
        Statistics show that teen pregnancy drops off significantly after 25.
```

Voir aussi `strip`, `wordwrap` et `spacify`.

lower

Met une variable en minuscules. C'est l'équivalent de la fonction PHP `strtolower()` [http://php.net/strtolower].

Exemple 5.13. lower

```
<?php
$smarty->assign('TitreArticle', 'Deux Suspects Se Sont Sauvés.');
?>
```

Où le template est :

```
{$TitreArticle}
{$TitreArticle|lower}
```

Affichera :

```
Deux Suspects Se Sont Sauvés.
deux suspects se sont sauvés.
```

Voir aussi `upper` et `capitalize`.

nl2br

Transforme toutes les fins de lignes en balises `<br />`. Équivalent à la fonction PHP `nl2br()` [http://php.net/nl2br].

Exemple 5.14. nl2br

```
<?php

$smarty->assign('articleTitle',
                "Sun or rain expected\ntoday, dark tonight"
               );

?>
```

Où le template est :

```
{$TitreArticle|nl2br}
```

Affichera :

```
Pluie ou soleil attendu<br />aujourd'hui, nuit noire
```

Voir aussi `word_wrap`, `count_paragraphs` et `count_sentences`.

regex_replace

Un rechercher / remplacer avec une expression régulière. Utilise la même syntaxe que la fonction PHP `preg_replace()` [http://php.net/preg_replace].

Position du paramètre	Type	Requis	Defaut	Description
1	chaîne de caractère	Oui	*n/a*	Expression régulière à remplacer.
2	chaîne de caractère	Oui	*n/a*	La chaîne de remplacement.

Exemple 5.15. regex_replace

```
<?php

$smarty->assign('TitreArticle', "L'infertilité est un maux grandissant\n, disent les experts.");

?>
```

Où le template est :

```
{* Remplace tous les retours chariot et les tabulation par une nouvelle ligne avec un espace *}

{$TitreArticle}
```

```
{$TitreArticle|regex_replace:"/[\r\t\n]/":" "}
```

Affichera :

```
L'infertilité est un maux grandissant
, disent les experts.
L'infertilité est un maux grandissant, disent les experts.
```

Voir aussi `replace` et `escape`.

replace

Un simple remplacement de chaîne de caractères. C'est l'équivalent de la fonction PHP `str_replace()` [http://php.net/str_replace].

Position du paramètre	Type	Requis	Defaut	Description
1	chaîne de caractères	Oui	*n/a*	chaîne à remplacer.
2	chaîne de caractères	Oui	*n/a*	chaîne de remplacement.

Exemple 5.16. replace

```
<?php

$smarty->assign('titreArticle', "Child's Stool Great for Use in Garden.");

?>
```

Ou le template est :

```
{$titreArticle}
{$titreArticle|replace:'Garden':'Vineyard'}
{$titreArticle|replace:' ':'   '}
```

Affichera :

```
Child's Stool Great for Use in Garden.
Child's Stool Great for Use in Vineyard.
Child's   Stool   Great   for   Use   in   Garden.
```

Voir aussi `regex_replace` et `escape`.

spacify

`spacify` est un moyen pour insérer un espace entre tous les caractères d'une variable. Optionnellement, vous pouvez lui passer un caractère (ou une chaîne) différent de l'espace à insérer.

Position du paramètre	Type	Requis	Defaut	Description
1	chaîne de caractère	Non	*espace*	Ce qui est inséré entre chaque caractère de la variable.

Exemple 5.17. spacify

```
<?php

$smarty->assign('titreArticle', 'Quelque chose s\'est mal passé et à provoqué
cet accident, disent les experts');

?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|spacify}
{$titreArticle|spacify:"^^"}
```

Affichera :

```
Quelquechose s'est mal passé et à provoqué cet accident, disent les experts.
Q u e l q u e c h o s e   .... snip ....   l e s   e x p e r t s .
Q^^u^^e^^l^^q^^u.... snip .... ^r^^t^^s^^.
```

Voir aussi `wordwrap` et `nl2br`.

string_format

Un moyen pour formater les chaînes de caractères, comme par exemple les nombres décimaux. Utilise la syntaxe de `sprintf()` [http://php.net/sprintf] pour formater les éléments.

Position du paramètre	Type	Requis	Defaut	Description
1	chaîne de caractères	Oui	*n/a*	Le format à utiliser (sprintf)

Exemple 5.18. string_format

```
<?php

$smarty->assign('nombre', 23.5787446);

?>
```

Où le template est :

```
{$nombre}
{$nombre|string_format:"%.2f"}
```

```
{$nombre|string_format:"%d"}
```

Affichera :

```
23.5787446
23.58
24
```

Voir aussi `date_format`.

strip

Remplace les espaces multiples, les nouvelles lignes et les tabulations par un espace simple, ou une chaîne donnée.

> **Note:** Si vous voulez réaliser cette action sur un bloc complet du template, utilisez la fonction `{strip}`.

Exemple 5.19. strip

```
<?php
$smarty->assign('titreArticle', "Une réunion autour\n d'un feu de cheminée\t
est toujours agréable.");
$smarty->display('index.tpl');
?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|strip}
{$titreArticle|strip:' '}
```

Affichera :

```
Une réunion autour
 d'un feu de cheminée     est toujours agréable.
Une réunion autour d'un feu de cheminée est toujours agréable.
Une réunion autour d'un feu de cheminée est toujours
 agréable.
```

Voir aussi `{strip}` et `truncate`.

strip_tags

Supprime toutes les balises, et plus généralement tout ce qui se trouve entre < et >.

Position du paramètre	Type	Requis	Defaut	Description
1	bool	Non	TRUE	Si l'on remplace les éléments par ' ' ou par ''

Exemple 5.20. strip_tags

```
<?php

$smarty->assign('articleTitle',
            "Blind Woman Gets <font face=\"helvetica\">New
 Kidney</font> from Dad she Hasn't Seen in <b>years</b>."
               );

?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|strip_tags} {* identique à {$titreArticle|strip_tags:true} *}
{$titreArticle|strip_tags:false}
```

Affichera :

```
Blind Woman Gets <font face="helvetica">New Kidney</font> from Dad she Hasn't Seen in <b>years</b>.
Blind Woman Gets  New Kidney  from Dad she Hasn't Seen in  years .
Blind Woman Gets New Kidney from Dad she Hasn't Seen in years.
```

Voir aussi `replace` et `regex_replace`.

truncate

Tronque une variable à une certaine longueur, par défaut 80. Un second paramètre optionnel permet de spécifier une chaîne à afficher à la fin de la variable une fois tronquée. Les caractères de fin sont inclus dans la longueur de la chaîne à tronquer. Par défaut, `truncate` tentera de couper la chaîne à la fin d'un mot. Si vous voulez tronquer la chaîne au caractère exact, donnez la valeur `TRUE` au dernier paramètre optionnel.

Position du paramètre	Type	Requis	Defaut	Description
1	entier	Non	80	Le nombre de caractères maximums au-delà duquel on effectue le troncage
2	chaîne de caractère	Non	...	Le texte qui remplace le texte tronqué. Sa longueur est incluse dans la configuration de la longueur à tronquer.
3	booléen	Non	`FALSE`	Détermine si le troncage est effectué sur le dernier mot (`FALSE`), ou au caractère exact (`TRUE`).
4	booléen	Non	`FALSE`	Ceci détermine si le troncage intervient à la fin de la chaîne (`FALSE`), ou au milieu de la chaîne (`TRUE`). Notez que si ceci vaut `TRUE`, alors les limites de mots sont ignorées.

Exemple 5.21. truncate

```
<?php

$smarty->assign('titreArticle', 'Deux soeurs réunies après 18 ans de séparation.');

?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|truncate}
{$titreArticle|truncate:30}
{$titreArticle|truncate:30:""}
{$titreArticle|truncate:30:"---"}
{$titreArticle|truncate:30:"":true}
{$titreArticle|truncate:30:"...":true}
{$articleTitle|truncate:30:'..':true:true}
```

Ce qui donne en sortie :

```
Deux soeurs réunies après 18 ans de séparation.
Deux soeurs réunies après...
Deux soeurs réunies après
Deux soeurs réunies après---
Deux soeurs réunies après 18 a
Deux soeurs réunies après 1...
Deux soeurs ... de séparation.
```

upper

Met une variable en majuscules. C'est l'équivalent de la fonction PHP `strtoupper()` [http://php.net/strtoupper].

Exemple 5.22. upper

```
<?php
$smarty->assign('titreArticle', "Si l'attaque n'est pas mise en place
rapidement, celà risque de durer longtemps.");
?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|upper}
```

Affichera :

```
Si l'attaque n'est pas mise en place rapidement, celà risque de durer longtemps.
SI L'ATTAQUE N'EST PAS MISE EN PLACE RAPIDEMENT, CELÀ RISQUE DE DURER LONGTEMPS.
```

Voir aussi `lower` et `capitalize`.

wordwrap

Ajuste une chaîne de caractères à une taille de colonne, par défaut 80. Un second paramètre optionnel vous permet de spécifier la chaîne à utiliser pour l'ajustement à la nouvelle ligne (retour chariot `"\n"` par défaut). Par défaut, `wordwrap` tente un ajustement à la fin d'un mot. Si vous voulez autoriser le découpage des mots pour un ajustement au caractère près, passez `TRUE` au troisième paramètre optionnel. Ceci est l'équivalent de la fonction PHP `wordwrap()` [http://php.net/wordwrap].

Position du paramètre	Type	Requis	Defaut	Description
1	entier	Non	80	La nombre de colonnes sur lequel ajuster l'affichage.
2	chaîne de caractères	Non	\n	chaîne de caractères utilisée pour l'ajustement.
3	booléen	Non	`FALSE`	Détermine si l'ajustement se fait en fin de mot (`FALSE`) ou au caractère exact (`TRUE`).

Exemple 5.23. wordwrap

```
<?php
$smarty->assign('articleTitle',
                "Blind woman gets new kidney from dad she hasn't seen in years."
              );
?>
```

Où le template est :

```
{$titreArticle}

{$titreArticle|wordwrap:30}

{$titreArticle|wordwrap:20}

{$titreArticle|wordwrap:30:"<br>\n"}

{$titreArticle|wordwrap:30:"\n":true}
```

L'exemple ci-dessus affichera :

```
Une femme aveugle obtient un nouveau rein d'un père qu'elle n'a pas vu depuis des années.

Une femme aveugle obtient un
nouveau rein d'un père
qu'elle n'a pas vu depuis
des années.

Une femme aveugle
obtient un nouveau
rein d'un père
qu'elle n'a pas vu
depuis des années.

Une femme aveugle obtient un<br>;
nouveau rein d'un père<br>;
qu'elle n'a pas vu depuis<br>;
```

```
des années.

Une femme aveugle obtient un n
ouveau rein d'un père qu'elle
n'a pas vu depuis des années.
```

Voir aussi `nl2br` et `{textformat}`.

Chapitre 6. Combiner des modificateurs de variable.

Vous pouvez appliquer un nombre quelquonque de modificateurs à une variable. Ils seront invoqués dans l'ordre d'apparition, de la gauche vers la droite. Ils doivent être séparés par un | (pipe).

Exemple 6.1. Combiner des modificateurs

```
<?php

$smarty->assign('titreArticle', 'Les fumeurs sont productifs, mais la mort
tue l\'efficacitée.');

?>
```

Où le template est :

```
{$titreArticle}
{$titreArticle|upper|spacify}
{$titreArticle|lower|spacify|truncate}
{$titreArticle|lower|truncate:30|spacify}
{$titreArticle|lower|spacify|truncate:30:". . ."}
```

Celà va afficher :

```
Les fumeurs sont productifs, mais la mort tue l'efficacitée.
L E S   F U M E U R S   S O N T   P R O D U C T I F S ,   M A I S   L A   M O R T   T U E   L ' E F F I
L E S   F U M E U R S   S O N T   P R O D U C T I F S ,   M A I S   L A   M...
L E S   F U M E U R S   S O N T   P R O D U C T I F S ,   M A I S   L A   M...
L e s   f u m e u r s   s o n t   p r o d u c t i f s , . . .
L e s   f u m e u r s   s. . .
```

Chapitre 7. Fonctions natives

Table des matières

Smarty est fourni en standard avec plusieurs fonctions natives. Ces fonctions natives sont partie intégrante du moteur de Smarty. Vous ne pouvez pas créer de fonctions utilisateurs qui portent le même nom qu'une fonction native et vous ne pouvez pas non plus en modifier le comportement.

Quelques-unes de ces fonctions ont un attribut *assign* qui récupère le résultat de la fonction et la place dans une variable nommée dans le template plutôt que de l'afficher ; tout comme la fonction `{assign}`.

{capture}

`{capture}` est utilisé pour récupérer la sortie d'éléments dans une variable au lieu de les afficher. Tout contenu situé entre `{capture name='foo'}` et `{/capture}` est intercepté dans une variable dont le nom est spécifié dans l'attribut *name*.

Le contenu capturé peut être utilisé dans le template par l'intermédiaire de la variable spéciale *$smarty.capture.foo* où «foo» est la valeur de l'attribut *name*. Si vous ne donnez pas de valeur à l'attribut *name*, alors «default» est utilisé en tant que nom, i.e. *$smarty.capture.default*.

`{capture}` peut être imbriqué.

Attribut	Type	Requis	Defaut	Description
name	string	non	*default*	Le nom du bloc capturé
assign	string	non	*n/a*	Le nom de la variable à laquelle la sortie sera assignée

Attention: Faîtes attention lorsque vous interceptez la sortie de commandes `{insert}`. Si vous avez configuré le cache pour que ce dernier soit actif, et que vous avez des commandes `{insert}` supposées s'exécuter dans un contenu en cache, ne tentez pas de capturer ce contenu.

Exemple 7.1. {capture} avec le nom de l'attribut

```
{* nous ne voulons afficher une balise div que si le contenu est affiché. *}
{capture name=banner}
  {include file='get_banner.tpl'}
{/capture}
{if $smarty.capture.banner ne ""}
<div id="banner">{$smarty.capture.banner}</div>
{/if}
```

Exemple 7.2. {capture} dans une variable de template

Cet exemple démontre également la fonction `{popup}`

```
{capture name=some_content assign=popText}
Le serveur est {$smarty.server.SERVER_NAME|upper} sur {$smarty.server.SERVER_ADDR}<br>
Votre IP est {$smarty.server.REMOTE_ADDR}.
{/capture}
<a href="#" {popup caption='Information sur le serveur' text=$popText}>Aide</a>
```

Voir aussi *$smarty.capture*, `{eval}`, `{fetch}`, `fetch()` et `{assign}`.

{config_load}

`{config_load}` est utiliseé pour charger des variables *#variables#* depuis un fichier de configuration dans un template.

Nom attribut	Type	Requis	Défaut	Description
file	chaîne de caractère	Oui	*n/a*	Le nom du fichier de configuration à inclure
section	chaîne de caractère	Non	*n/a*	Le nom de la section à charger
scope	chaîne de caractère	non	*local*	La façon dont la portée des variables est définie, soit local, parent ou global. local signifie que la variable est chargée dans le contexte du template. parent indique que la variable est disponible tant dans le template qui l'a inclus que dans le template parent, ayant réalisé l'inclusion du sous template. global signifie que la variable est diponible dans tous les templates.
global	booléen	Non	*No*	Si oui ou non les variables sont disponibles pour les templates parents, identique à scope=parent. Note: Cet attribut est obsolète

Nom attribut	Type	Requis	Défaut	Description
				depuis l'apparition de l'attribut scope, il est toutefois toujours supporté. Si scope est défini, global est ignoré.

Exemple 7.3. Fonction {config_load}

Le fichier `example.conf`

```
#ceci est un commentaire de fichier de configuration

#variables globales
pageTitle = "Menu principal"
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

#section de variables personnalisées
[Customer]
pageTitle = "Info personnalisée"
```

et le template

```
{config_load file='example.conf'}

<html>
<title>{#pageTitle#|default:"No title"}</title>
<body bgcolor="{#bodyBgColor#}">
<table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
  <tr bgcolor="{#rowBgColor#}">
    <td>First</td>
    <td>Last</td>
    <td>Address</td>
  </tr>
</table>
</body>
</html>
```

Les fichiers de configuration peuvent contenir des sections. Vous pouvez charger des variables d'une section donnée avec le nouvel attribut *section*.

> **NOTE:** Les *sections* des fichiers de configuration et la fonction native `{section}` n'ont rien en commun, il s'avère simplement qu'elles portent le même nom.

Exemple 7.4. fonction {config_load} avec section

```
{config_load file='example.conf' section='Customer'}

<html>
<title>{#pageTitle#}</title>
<body bgcolor="{#bodyBgColor#}">
<table border="{#tableBorderSize#}" bgcolor="{#tableBgColor#}">
  <tr bgcolor="{#rowBgColor#}">
    <td>First</td>
    <td>Last</td>
```

```
    <td>Address</td>
  </tr>
</table>
</body>
</html>
```

Voir aussi *$config_overwrite* pour les tableaux de variables de configuration.

Voir aussi les fichiers de configuration, les variables de configuration, *$config_dir*, `get_config_vars()` et `config_load()`.

{foreach},{foreachelse}

`{foreach}` est utilisé pour parcourir un **simple tableau associatif**, contrairement à `{section}` qui effectue une boucle sur les **tableaux de données**. La synthaxe pour `{foreach}` est plus simple que `{section}`, mais **ne peut être utilisé que pour des tableau simple**. Chaque `{foreach}` doit aller de paire avec une balise fermante `{/foreach}`.

Nom attribut	Type	Requis	Defaut	Description
from	tableau	oui	*n/a*	Le tableau à parcourir
item	chaîne de caractère	Oui	*n/a*	Le nom de la variable "élément courant"
key	chaîne de caractère	Non	*n/a*	Le nom de la variable représentant la clef courante.
name	chaîne de caractère	Non	*n/a*	Le nom de la boucle foreach, qui nous permettra d'accéder à ses propriétés.

- Required attributes are *from* and *item*.
- The *name* of the `{foreach}` loop can be anything you like, made up of letters, numbers and underscores, like PHP variables [http://php.net/language.variables].
- `{foreach}` loops can be nested, and the nested `{foreach}` names must be unique from each other.
- The *from* attribute, usually an array of values, determines the number of times `{foreach}` will loop.
- `{foreachelse}` is executed when there are no values in the *from* variable.
- `{foreach}` loops also have their own variables that handle properties. These are accessed with: *{$smarty.foreach.name.property}* with «name» being the *name* attribute.

  Note: The *name* attribute is only required when you want to access a `{foreach}` property, unlike `{section}`. Accessing a `{foreach}` property with *name* undefined does not throw an error, but leads to unpredictable results instead.

- `{foreach}` properties are *index*, *iteration*, *first*, *last*, *show*, *total*.

Exemple 7.5. L'attribut *item*

```
<?php
$arr = array(1000, 1001, 1002);
$smarty->assign('myArray', $arr);
?>
```

Template pour afficher *$myArray* dans une liste non-ordonnée.

```
<ul>
 {foreach from=$myArray item=foo}
 <li>{$foo}</li>
 {/foreach}
</ul>
```

L'exemple ci-dessus affichera :

```
<ul>
 <li>1000</li>
 <li>1001</li>
 <li>1002</li>
</ul>
```

Exemple 7.6. Utilisation des attributs *item* et *key*

```
<?php
$arr = array(9 => 'Tennis', 3 => 'Natation', 8 => 'Programmation');
$smarty->assign('myArray', $arr);
?>
```

Le template affiche le tableau *$myArray* comme paire clé/valeur, comme la fonction PHP `foreach` [http://php.net/foreach].

```
<ul>
 {foreach from=$myArray key=k item=v}
 <li>{$k}: {$v}</li>
 {/foreach}
</ul>
```

L'exemple ci-dessus affichera :

```
<ul>
 <li>9: Tennis</li>
 <li>3: Natation</li>
 <li>8: Programmation</li>
</ul>
```

Exemple 7.7. {foreach} avec un attribut associatif *item*

```
<?php
$items_list = array(23 => array('no' => 2456, 'label' => 'Salad'),
    96 => array('no' => 4889, 'label' => 'Cream')
);
$smarty->assign('items', $items_list);
?>
```

Le template affiche *$items* avec *$myId* dans l'URL.

```
<ul>
 {foreach from=$items key=myId item=i}
 <li><a href="item.php?id={$myId}">{$i.no}: {$i.label}</li>
  {/foreach}
 </ul>
```

L'exemple ci-dessus affichera :

```
<ul>
<li><a href="item.php?id=23">2456: Salad</li>
<li><a href="item.php?id=96">4889: Cream</li>
</ul>
```

Exemple 7.8. {foreach} avec *item* et *key*

Assigne un tableau à Smarty, la clé contient la clé pour chaque valeur de la boucle.

```
<?php
$smarty->assign('contacts', array(
array('phone' => '1',
         'fax' => '2',
         'cell' => '3'),
         array('phone' => '555-4444',
         'fax' => '555-3333',
         'cell' => '760-1234')
));
?>
```

Le template affiche *$contact*.

```
{foreach name=outer item=contact from=$contacts}
    <hr />
    {foreach key=key item=item from=$contact}
        {$key}: {$item}<br />
    {/foreach}
{/foreach}
```

L'exemple ci-dessus affichera :

```
<hr />
phone: 1<br />
fax: 2<br />
cell: 3<br />
<hr />
phone: 555-4444<br />
fax: 555-3333<br />
cell: 760-1234<br />
```

Exemple 7.9. Exemple d'une base de données avec {foreachelse}

Exemple d'un script de recherche dans une base de données (e.g. PEAR ou ADODB), le résultat de la requête est assigné à Smarty.

```
<?php
$search_condition = "where name like '$foo%' ";
$sql = 'select contact_id, name, nick from contacts '.$search_condition.' order by name';
$smarty->assign('results', $db->getAssoc($sql) );
```

```
?>
```

Le template qui affiche «None found» si aucun résultat avec `{foreachelse}`.

```
{foreach key=cid item=con from=$results}
<a href="contact.php?contact_id={$cid}">{$con.name} - {$con.nick}</a><br />
{foreachelse}
Aucun élément n'a été trouvé dans la recherche
{/foreach}
```

.index

index contient l'index courant du tableau, en commançant par zéro.

Exemple 7.10. Exemple avec *index*

```
{* L'en-tête du block est affiché toutes les 5 lignes *}
<table>
 {foreach from=$items key=myId item=i name=foo}
 {if $smarty.foreach.foo.index % 5 == 0}
 <tr><th>Title</th></tr>
 {/if}
 <tr><td>{$i.label}</td></tr>
 {/foreach}
</table>
```

.iteration

iteration contient l'itération courante de la boucle et commence toujours à 1, contrairement à *index*. Il est incrémenté d'un, à chaque itération.

Exemple 7.11. Exemple avec *iteration* et *index*

```
{* this will output 0|1, 1|2, 2|3, ... etc *}
{foreach from=$myArray item=i name=foo}
{$smarty.foreach.foo.index}|{$smarty.foreach.foo.iteration},
{/foreach}
```

.first

first vaut `TRUE` si l'itération courante de `{foreach}` est l'initial.

Exemple 7.12. Exemple avec *first*

```
{* affiche LATEST sur le premier élément, sinon, l'id *}
 <table>
  {foreach from=$items key=myId item=i name=foo}
  <tr>
  <td>{if $smarty.foreach.foo.first}LATEST{else}{$myId}{/if}</td>
  <td>{$i.label}</td>
 </tr>
```

```
 {/foreach}
</table>
```

.last

last est défini à `TRUE` si l'itération courante de `{foreach}` est la dernière.

Exemple 7.13. Exemple avec *last*

```
{* Ajout une barre horizontale à la fin de la liste *}
{foreach from=$items key=part_id item=prod name=products}
<a href="#{$part_id}">{$prod}</a>{if $smarty.foreach.products.last}<hr>{else},{/if}
{foreachelse}
... contenu ...
{/foreach}
```

.show

show est utilisé en tant que paramètre à `{foreach}`. *show* est une valeur booléenne. S'il vaut `FALSE`, `{foreach}` ne sera pas affiché. S'il y a un `{foreachelse}`, il sera affiché alternativement.

.total

total contient le nombre d'itérations que cette boucle `{foreach}` effectuera. Il peut être utilisé dans ou après un `{foreach}`.

Exemple 7.14. Exemple avec *total*

```
{* affiche les lignes retournées à la fin *}
{foreach from=$items key=part_id item=prod name=foo}
{$prod.name><hr/>
{if $smarty.foreach.foo.last}
<div id="total">{$smarty.foreach.foo.total} items</div>
{/if}
{foreachelse}
... quelque chose d'autre ...
{/foreach}
```

Voir aussi `{section}` et *$smarty.foreach*.

{if},{elseif},{else}

L'instruction `{if}` dans Smarty dispose de la même flexibilité que l'instruction PHP if [http://php.net/if], avec quelques fonctionnalités supplémentaires pour le moteur de template. Tous les `{if}` doivent être utilisés de pair avec un `{/if}`. `{else}` et `{elseif}` sont également des balises autorisées. Toutes les conditions et fonctions PHP sont reconnues, comme *||*, *or*, *&&*, *and*, *is_array()*, etc.

Si *$security* est actif, alors le tableau *IF_FUNCS* dans le tableau *$security_settings* (?!).

La liste suivante présente les opérateurs reconnus, qui doivent être entourés d'espaces. Remarquez que les éléments listés

entre [crochets] sont optionnels. Les équivalents PHP sont indiqués lorsque applicables.

Opérateur	Syntaxe alternative	Exemple de syntaxe	Signification
==	eq	$a eq $b	égalité
!=	ne, neq	$a neq $b	différence
>	gt	$a gt $b	supérieur à
<	lt	$a lt $b	inférieur à
>=	gte, ge	$a ge $b	supérieur ou égal à
<=	lte, le	$a le $b	inférieur ou égal à
===		$a === 0	égalité (type et valeur)
!	not	not $a	négation
%	mod	$a mod $b	modulo
is [not] div by		$a is not div by 4	divisible par
is [not] even		$a is not even	est [ou non] un nombre pair
is [not] even by		$a is not even by $b	paritée de groupe
is [not] odd		$a is not odd	est [ou non] un nombre impair
is [not] odd by		$a is not odd by $b	est [ou non] un groupe impair

Exemple 7.15. Instruction {if}

```
{if $name eq 'Fred'}
  Bienvenue, Monsieur.
{elseif $name eq 'Wilma'}
  Bienvenue m'dame.
{else}
  Bienvenue, qui que vous soyez.
{/if}

{* Un exemple avec l'opérateur or *}
{if $name eq 'Fred' or $name eq 'Wilma'}
  ...
{/if}

{* même chose que ci-dessus *}
{if $name == 'Fred' || $name == 'Wilma'}
  ...
{/if}

{* les parenthèses sont autorisées *}
{if ( $amount < 0 or $amount > 1000 ) and $volume >= #minVolAmt#}
  ...
{/if}

{* vous pouvez également faire appel aux fonctions PHP *}
{if count($var) gt 0}
  ...
{/if}

{* Vérifie si c'est un tableau. *}
{if is_array($foo) }
.....
{/if}

{* Vérifie si la variable est nulle. *}
{if isset($foo) }
```

```
  .....
{/if}

{* teste si les valeurs sont paires(even) ou impaires(odd) *}
{if $var is even}
  ...
{/if}
{if $var is odd}
  ...
{/if}
{if $var is not odd}
  ...
{/if}

{* teste si la variable est divisible par 4 *}
{if $var is div by 4}
  ...
{/if}

{* teste si la variable est paire, par groupe de deux i.e.,
0=paire, 1=paire, 2=impaire, 3=impaire, 4=paire, 5=paire, etc. *}
{if $var is even by 2}
  ...
{/if}

{* 0=paire, 1=paire, 2=paire, 3=impaire, 4=impaire, 5=impaire, etc. *}
{if $var is even by 3}
  ...
{/if}
```

Exemple 7.16. Plus d'exemples avec {if}

```
{if isset($name) && $name = 'Blog'}
   {* faire quelque chose *}
{elseif $name == $foo}
   {* faire quelque chose *}
{/if}

{if is_array($foo) && count($foo) > 0)
   {* faire une boucle foreach *}
   {/if}
```

{include}

Les balises `{include}` sont utilisées pour inclure des templates à l'intérieur d'autres templates. Toutes les variables disponibles dans le template réalisant l'inclusion sont disponibles dans le template inclus.

- La balise `{include}` doit contenir l'attribut *file* qui contient le chemin vers la ressource de template.
- La définition de l'attribut optionnel *assign* spécifie la variable de template assignée à la sortie de `{include}` au lieu d'être affichée. Similaire à `{assign}`.
- Les variables peuvent être passées à des templates inclus comme attributs. Toutes les variables explicitement passées à un template inclus ne sont disponibles que dans le contexte du fichier inclus. Les attributs de variables écrasent les variables courantes de template, dans le cas où les noms sont les mêmes.
- Toutes les valeurs de variables assignées sont restaurées une fois le contexte du template inclus refermés. Ceci signifie que vous pouvez utiliser toutes les variables depuis un template inclus dans le template inclus. Mais les modifications faites aux variables dans le template inclus ne sont pas visibles dans le template incluant, parès l'instruction `{include}`

statement.

- Utilisez la synthaxe pour les ressources de template aux fichiers `{include}` en dehors du dossier *$template_dir*.

Nom attribut	Type	Requis	Defaut	Description
file	chaîne de caractères	Oui	*n/a*	Le nom du template à inclure
assign	chaîne de caractères	Non	*n/a*	Le nom de la variable dans laquelle sera assignée la sortie de include
[var ...]	[type de variable]	Non	*n/a*	Variables à passer au template

Exemple 7.17. Exemple avec {include}

```
<html>
 <head>
  <title>{$title}</title>
 </head>
 <body>
  {include file='page_header.tpl'}

  {* Le corps du template va ici, la variable $tpl_name est remplacé par
    une valeur, e.g.'contact.pl' *}
  {include file='$tpl_name.tpl'}

  {include file='page_footer.tpl'}
 </body>
</html>
```

Exemple 7.18. Fonction {include}, passage de variables

```
{include file='links.tpl' title='Newest links' links=$link_array}
{* body of template goes here *}
{include file='footer.tpl' foo='bar'}
```

Le template ci-dessus inclut l'exemple `links.tpl`

```
<div id="box">
 <h3>{$title}{/h3>
  <ul>
   {foreach from=$links item=l}
   .. faites quelques choses ici  ...
  </foreach}
  </ul>
</div>
```

Exemple 7.19. {include} et assignement à une variable

Cet exemple assigne le contenu de `nav.tpl` à la variable `$navbar`, qui est alors affichée en haut et en bas de la page.

```
<body>
 {include file='nav.tpl' assign=navbar}
 {include file='header.tpl' title='Smarty is cool'}
 {$navbar}
 {* le corps du template va ici *}
 {$navbar}
 {include file='footer.tpl'}
</body>
```

Exemple 7.20. Divers {include}, exemple de ressource template

```
{* chemin absolu *}
{include file='/usr/local/include/templates/header.tpl'}

{* chemin absolu (même chose) *}
{include file='file:/usr/local/include/templates/header.tpl'}

{* chemin absolu windows (DOIT utiliser le préfixe "file:") *}
{include file='file:C:/www/pub/templates/header.tpl'}

{* inclusion d'une ressource template "db" *}
{include file='db:header.tpl'}

{* inclusion d'un template $variable - eg $module = 'contacts' *}
{include file="$module.tpl"}
{* ne fonctionne pas avec des simples guillemets ie aucun substitution de variables *}
{include file='$module.tpl'}

{* include a multi $variable template - eg amber/links.view.tpl *}
{include file="$style_dir/$module.$view.tpl"}
```

Voir aussi `{include_php}`, `{insert}`, `{php}`, les ressources de template et les templates composants.

{include_php}

> **Notes techniques:** `{include_php}` est presque obsolète dans Smarty. Vous pouvez obtenir des résultats équivalents en utilisant les fonctions utilisateur. La seule raison qui peut vous pousser à utiliser `{include_php}` est que vous avez besoin de mettre une de vos fonction en quarantaine vis à vis du répertoire `plugins/` ou de votre application. Reportez-vous à l'exemple des templates composants pour plus de détails.

Nom attribut	Type	Requis	Défaut	Description
file	chaîne de caractère	oui	*n/a*	Le nom du fichier PHP à inclure
once	boléen	Non	*TRUE*	Inclure plusieurs fois ou non le fichier PHP si plusieurs demandes d'inclusions sont faîtes.
assign	chaîne de caractère	Non	*n/a*	le nom de la variable PHP dans laquelle la sortie sera assignée plutôt que directement affichée.

Les balises `{include_php}` sont utilisées pour inclure directement un script PHP dans vos templates. Si *$security* est activé, alors le script à exécuter doit être placé dans le chemin *$trusted_dir*. La balise `{include_php}` attends l'attribut *file*,

qui contient le chemin du fichier PHP à inclure, relatif à *$trusted_dir*, ou absolu.

Par défaut, les fichiers PHP ne sont inclus qu'une seule fois, même si la demande d'inclusion survient plusieurs fois dans le template. Vous pouvez demander à ce que ce fichier soit inclus à chaque demande grâce à l'attribut *once*. Mettre l'attribut once à `FALSE` aura pour effet d'inclure le script PHP à chaque fois que demandé dans le template.

Vous pouvez donner une valeur à l'attribut optionnel *assign*, pour demander à la fonction `{include_php}` d'assigner la sortie du script PHP à la variable spécifiée plutôt que d'en afficher directement le résultat.

L'objet Smarty est disponible en tant que *$this* dans le script PHP inclus.

Exemple 7.21. Fonction {include_php}

Le fichier `load_nav.php`

```
<?php

// charge des variables depuis une base de données mysql et les assigne au template.
require_once('MySQL.class.php');
$sql = new MySQL;
$sql->query('select * from site_nav_sections order by name',SQL_ALL);
$this->assign('sections',$sql->record);

?>
```

Où le template est :

```
{* chemin absolu, ou relatif à $trusted_dir *}
{include_php file='/chemin/vers/load_nav.php'}

{foreach item='nav' from=$navigation}
  <a href="{$nav.url}">{$nav.name}</a><br />
{/foreach}
```

Voir aussi `{include}`, *$security*, *$trusted_dir*, `{php}`, `{capture}`, les ressources de template et les composants de templates.

{insert}

Les balises `{insert}` fonctionnent à peu près comme les balises `{include}`, à l'exception que leur sortie n'est PAS placée en cache lorsque le cache du template est activé. Les balises {insert} seront exécutées à chaque appel du template.

Nom attribut	Type	Requis	Defaut	Description
name	chaîne de caractères	Oui	*n/a*	le nom de la fonction insert (insert_name)
assign	chaîne de caractère	Non	*n/a*	Le nom de la variable qui recevra la sortie
script	chaîne de caractères	Non	*n/a*	Le nom du script PHP inclus avant que la fonction insert ne soit appelée.
[var ...]	[var type]	Non	*n/a*	Variable à passer à la fonction insert

Supposons que vous avez un template avec un emplacement pour un bandeau publicitaire en haut de page. Ce bandeau publicitaire peut contenir toutes sortes de contenus HTML, images, flash, etc. Nous ne pouvons pas placer du contenu statique à cet endroit. Nous ne voulons pas non plus que ce contenu fasse partie du cache. Arrive alors la balise {insert}. Le template connais #emplacement_bandeau# et #id_site# (récupérés depuis un fichier de configuration), et à besoin d'un appel de fonction pour récupérer le contenu du bandeau.

Exemple 7.22. Fonction {insert}

```
{* exemple de récupération d'un bandeau publicitaire *}
{insert name="getBanner" lid=#emplacement_bandeau# sid=#id_site#}
```

Dans cet exemple, nous utilisons le nom «getBanner» et lui passons les paramètres #emplacement_bandeau# et #id_site#. Smarty va rechercher une fonction appelée insert_getBanner () dans votre application PHP, et lui passer les valeurs #banner_location_id# et #site_id# comme premier paramètre, dans un tableau associatif. Tous les noms des fonctions {insert} de votre application doivent être prefixées de "insert_" pour remédier à d'éventuels conflits de nommage. Votre fonction insert_getBanner () est supposée traiter les valeurs passées et retourner un résultat. Ces résultats sont affichés dans le template en lieu et place de la balise. Dans cet exemple, Smarty appellera cette fonction insert_getBanner(array("lid" => "12345","sid" => "67890")); et affichera le résultat retourné à la place de la balise {insert}.

- Si vous donnez une valeur à l'attribut *assign*, la sortie de la balise `{insert}` sera assigné à une variable de template de ce nom au lieu d'être affichée directement.

  NOTE: Assigner la sortie à une variable n'est pas très utile lorsque le cache est activé.

- Si vous donnez une valeur à l'attribut *script*, ce script PHP sera inclus (une seule fois) avant l'exécution de la fonction `{insert}`. Le cas peut survenir lorsque la fonction `{insert}` n'existe pas encore, et que le script PHP chargé de sa définission doit être inclus.

  Le chemin doit être absolu ou relatif à *$trusted_dir*. Lorsque *$security* est actif, le script doit être situé dans *$trusted_dir*.

L'objet Smarty est passé comme second argument. De cette façon, vous pouvez utiliser ou modifier des informations sur l'objet Smarty, directement depuis votre fonction `{insert}`.

> **Note technique:** Il est possible d'avoir des portions de template qui ne soient pas gérables par le cache. Même si vous avez activé l'option caching, les balises `{insert}` ne feront pas partie du cache. Elles retourneront un contenu dynamique à chaque invocation de la page. Cette méthode est très pratique pour des éléments tels que les bandeaux publicitaires, les enquêtes, la météo, les résultats de recherche, retours utilisateurs, etc.

Voir aussi `{include}`

{ldelim},{rdelim}

`{ldelim}` et `{rdelim}` sont utilisés pour échapper les délimiteurs en tant que tels, dans notre cas, **{** et **}**. Vous pouvez toujours utiliser `{literal}{/literal}` pour échapper des blocks de texte, e.g. Javascript ou css. Voir aussi *{$smarty.ldelim}*.

Exemple 7.23. {ldelim}, {rdelim}

```
{* Affiche les délimiteurs de template *}

{ldelim}nomFonction{rdelim} est la façon dont sont appelées les fonctions dans Smarty !
```

Affichera :

```
{nomFonction} est la façon dont sont appelées les fonctions dans Smarty !
```

Un autre exemple avec du javascript

```
<script language="JavaScript">
 function foo() {ldelim}
   ... code ...
   {rdelim}
</script>
```

affichera :

```
<script language="JavaScript">
 function foo() {
   .... code ...
 }
</script>
```

Exemple 7.24. un autre exemple avec Javascript

```
<script language="JavaScript" type="text/javascript">
   function myJsFunction(){ldelim}
     alert("Le nom du serveur\n{$smarty.server.SERVER_NAME}\n{$smarty.server.SERVER_ADDR}");
   {rdelim}
</script>
<a href="javascript:myJsFunction()">Cliquez ici pour des informations sur le serveur</a>
```

Voir aussi `{literal}` et la désactivation de l'analyse de Smarty.

{literal}

Les balises `{literal}` permettent à un bloc de données d'être pris tel quel, sans qu'il ne soit interprété par Smarty. Très pratique lors de l'emplois d'éléments tels que javascript, acolades et autres qui peuvent confondre le moteur de template. Tout le contenu situé entre les balises `{literal}{/literal}` ne sera pas interprété, et affiché comme du contenu statique. Si vous voulez inclure des tags de template dans votre block `{literal}`, utilisez plutôt `{ldelim}{rdelim}` pour échapper les délimiteurs individuels.

Exemple 7.25. Balises {literal}

```
{literal}
<script language=javascript>

<!--
 function isblank(field) {
   if (field.value == '')
    { return false; }
   else
    {
      document.loginform.submit();
      return true;
    }
 }
// -->
```

```
</script>
{/literal}
```

Exemple 7.26. Exemple avec Javascript

```
<script language="JavaScript" type="text/javascript">
 {literal}
 function myJsFunction(name, ip){
 alert("Le nom du serveur\n" + name + "\n" + ip);
 }
 {/literal}
</script>
<a href="javascript:myJsFunction('{$smarty.server.SERVER_NAME}','{$smarty.server.SERVER_ADDR}')">Clique
```

Exemple 7.27. Un peu de css dans un template

```
{* inclure ce style... comme une expérimentation ! *}
<style type="text/css">
 {literal}
 /* C'est une idée intéressante pour cette section */
 .madIdea{
 border: 3px outset #ffffff;
 margin: 2 3 4 5px;
 background-color: #001122;
 }
 {/literal}
</style>
<div class="madIdea">Avec Smarty, vous pouvez inclure du css dans le template</div>
```

Voir aussi `{ldelim} {rdelim}` et la désactivation de l'analyse de Smarty.

{php}

Les balises `{php}` permettent de rajouter du code PHP directement dans le template. Ils ne seront pas ignorés, quelle que soit la valeur de *$php_handling*. Pour les utilisateurs avancés seulement, son utilisation n'est normalement pas nécessaire et n'est pas recommandée.

> **Notes techniques:** Pour accéder aux variables PHP dans les blocks `{php}`, vous devriez avoir besoin d'utiliser le mot clé PHP `global` [http://php.net/global].

Exemple 7.28. Exemple avec la balise {php}

```
{php}
   // inclusion directe d'un script PHP depuis le template.
   include('/chemin/vers/display_weather.php');
{/php}
```

Exemple 7.29. Balises {php} avec le mot clé global et assignement d'une variable

```
{* ce template inclut un bloc {php} qui assigne la variable $varX *}
{php}
   global $foo, $bar;
   if($foo == $bar){
     echo 'Ceci apparaîtera dans le template';
   }
$this->assign('varX','Strawberry');
{/php}
{* affichage de la variable *}
<strong>{$varX}</strong> est ma glâce favorite :-)
```

Voir aussi *$php_handling*, `{include_php}`, `{include}`, `{insert}` et les templates composantes.

{section},{sectionelse}

Une `{section}` sert à boucler dans des **tableaux de données**, contrairement à `{foreach}` qui est utilisé pour boucler dans un **simple tableau associatif**. Chaque balise `{section}` doit aller de paire avec une balise `{/section}` fermante.

Nom attribut	Type	Requis	Défaut	Description
name	chaîne de caractère	Oui	*n/a*	Le nom de la section
loop	mixed	Oui	*n/a*	Valeur qui détermine le nombre de fois que la boucle sera exécutée
start	entier	Non	*0*	La position de l'index ou la section commencera son parcours. Si la valeur donnée est négative, la position de départ est calculée depuis la fin du tableau. Par exemple, s'il existe 7 valeurs dans le tableau à parcourir et que start est à -2, l'index de départ sera 5. Les valeurs incorrectes (en dehors de la portée du tableau) sont automatiquements tronquées à la valeur correcte la plus proche
step	entier	Non	*1*	La valeur du pas qui sera utilisé pour parcourir le tableau.Par exemple, step=2 parcourera les indices 0,2,4, etc. Si step est négatif, le tableau sera parcouru en sens inverse
max	entier	Non	*n/a*	Définit le nombre maximum de fois que le tableau sera parcouru
show	booléen	No	*TRUE*	Détermine s'il est

Nom attribut	Type	Requis	Défaut	Description
				nécessaire d'afficher la section ou non

- Les paramètres requis sont *name* et *loop*.
- Le *name* de la `{section}` est, selon votre choix, composé de lettres, chiffres et underscores, comme pour les variables PHP [http://php.net/language.variables].
- Les sections peuvent être imbriquées mais leurs noms doivent être uniques.
- L'attribut *loop*, habituellement un tableau de valeurs, détermine le nombre de fois que `{section}` doit boucler.
- Lors de l'affichage d'une variable dans une `{section}`, le nom de la `{section}` doit être fournis après le nom de la variable entre crochets [].
- `{sectionelse}` est exécuté lorsqu'aucune valeur n'est trouvée dans la variable à parcourir.
- `{section}` a également ces propres variables qui gérent les propriétés de la `{section}`. Ces propriétés sont accessibles comme ceci : *{$smarty.section.name.property}* où «name» est l'attribut *name*.
- Les propriétés de `{section}` sont *index*, *index_prev*, *index_next*, *iteration*, *first*, *last*, *rownum*, *loop*, *show*, *total*.

Exemple 7.30. Boucler dans un simple tableau avec {section}

`assign()` un tableau à Smarty

```
<?php
$data = array(1000,1001,1002);
$smarty->assign('custid',$data);
?>
```

Le template qui affiche le tableau

```
{* Cet exemple affichera toutes les valeurs du tableau $custid *}
{section name=customer loop=$custid}
  id: {$custid[customer]}<br />
{/section}
<hr />
{* Affiche toutes les valeurs du tableau $custid, en ordre inverse *}
{section name=foo loop=$custid step=-1}
  {$custid[foo]}<br />
{/section}
```

L'exemple ci-dessus affichera :

```
id: 1000<br />
id: 1001<br />
id: 1002<br />
<hr />
id: 1002<br />
id: 1001<br />
id: 1000<br />
```

Exemple 7.31. {section} sans un tableau assigné

```
{section name=foo start=10 loop=20 step=2}
  {$smarty.section.foo.index}
{/section}
<hr />
{section name=bar loop=21 max=6 step=-2}
  {$smarty.section.bar.index}
{/section}
```

L'exemple ci-dessus affichera :

```
10 12 14 16 18
<hr />
20 18 16 14 12 10
```

Exemple 7.32. Nommage d'une {section}

Le *name* de la `{section}` peut être ce que vous voulez, voir les variables PHP [http://php.net/language.variables]. Il sera utilisé pour référencer les données de la `{section}`.

```
{section name=anything loop=$myArray}
  {$myArray[anything].foo}
  {$name[anything]}
  {$address[anything].bar}
{/section}
```

Exemple 7.33. Boucler dans un tableau associatif avec {section}

Voici un exemple d'affichage d'un tableau associatif de données avec `{section}`. Ce qui suit est le script PHP assignant le tableau *$contacts* à Smarty.

```
<?php
$data = array(
          array('name' => 'John Smith', 'home' => '555-555-5555',
                'cell' => '666-555-5555', 'email' => 'john@myexample.com'),
          array('name' => 'Jack Jones', 'home' => '777-555-5555',
                'cell' => '888-555-5555', 'email' => 'jack@myexample.com'),
          array('name' => 'Jane Munson', 'home' => '000-555-5555',
                'cell' => '123456', 'email' => 'jane@myexample.com')
        );
$smarty->assign('contacts',$data);
?>
```

Le template pour afficher *$contacts*

```
{section name=customer loop=$contacts}
<p>
  name: {$contacts[customer].name}<br />
  home: {$contacts[customer].home}<br />
  cell: {$contacts[customer].cell}<br />
  e-mail: {$contacts[customer].email}
</p>
{/section}
```

L'exemple ci-dessus affichera :

```
<p>
  name: John Smith<br />
  home: 555-555-5555<br />
  cell: 666-555-5555<br />
  e-mail: john@myexample.com
</p>
<p>
  name: Jack Jones<br />
  home phone: 777-555-5555<br />
  cell phone: 888-555-5555<br />
  e-mail: jack@myexample.com
</p>
<p>
  name: Jane Munson<br />
  home phone: 000-555-5555<br />
  cell phone: 123456<br />
  e-mail: jane@myexample.com
</p>
```

Exemple 7.34. {section} démontrant l'utilisation de la variable `loop`

Cet exemple suppose que *$custid*, *$name* et *$address* sont tous des tableaux contenant le même nombre de valeurs. Tout d'abord, le script PHP qui assigne les tableaux à Smarty.

```
<?php

$id = array(1001,1002,1003);
$smarty->assign('custid',$id);

$fullnames = array('John Smith','Jack Jones','Jane Munson');
$smarty->assign('name',$fullnames);

$addr = array('253 Abbey road', '417 Mulberry ln', '5605 apple st');
$smarty->assign('address',$addr);

?>
```

La variable *loop* détermine uniquement le nombre de fois qu'il faut boucler. Vous pouvez accéder à n'importe quelle variable du template dans la `{section}`

```
{section name=customer loop=$custid}
<p>
  id: {$custid[customer]}<br />
  name: {$name[customer]}<br />
  address: {$address[customer]}
</p>
{/section}
```

L'exemple ci-dessus affichera :

```
<p>
  id: 1000<br />
  name: John Smith<br />
  address: 253 Abbey road
</p>
<p>
  id: 1001<br />
  name: Jack Jones<br />
  address: 417 Mulberry ln
</p>
<p>
```

```
  id: 1002<br />
  name: Jane Munson<br />
  address: 5605 apple st
</p>
```

Exemple 7.35. {section} imbriquée

Les sections peuvent être imbriquées autant de fois que vous le voulez. Avec les sections imbriquées, vous pouvez accéder aux structures de données complexes, comme les tableaux multi-dimentionnels. Voici un script PHP qui assigne les tableaux.

```
<?php

$id = array(1001,1002,1003);
$smarty->assign('custid',$id);

$fullnames = array('John Smith','Jack Jones','Jane Munson');
$smarty->assign('name',$fullnames);

$addr = array('253 N 45th', '417 Mulberry ln', '5605 apple st');
$smarty->assign('address',$addr);

$types = array(
           array( 'home phone', 'cell phone', 'e-mail'),
           array( 'home phone', 'web'),
           array( 'cell phone')
         );
$smarty->assign('contact_type', $types);

$info = array(
           array('555-555-5555', '666-555-5555', 'john@myexample.com'),
           array( '123-456-4', 'www.example.com'),
           array( '0457878')
        );
$smarty->assign('contact_info', $info);

?>
```

Dans ce template, *$contact_type[customer]* est un tableau de types de contacts.

```
{section name=customer loop=$custid}
<hr>
  id: {$custid[customer]}<br />
  name: {$name[customer]}<br />
  address: {$address[customer]}<br />
  {section name=contact loop=$contact_type[customer]}
    {$contact_type[customer][contact]}: {$contact_info[customer][contact]}<br />
  {/section}
{/section}
```

L'exemple ci-dessus affichera :

```
<hr>
  id: 1000<br />
  name: John Smith<br />
  address: 253 N 45th<br />
    home phone: 555-555-5555<br />
    cell phone: 666-555-5555<br />
    e-mail: john@myexample.com<br />
<hr>
  id: 1001<br />
  name: Jack Jones<br />
  address: 417 Mulberry ln<br />
```

```
    home phone: 123-456-4<br />
    web: www.example.com<br />
<hr>
  id: 1002<br />
  name: Jane Munson<br />
  address: 5605 apple st<br />
    cell phone: 0457878<br />
```

Exemple 7.36. Exemple avec une base de données et {sectionelse}

Les résultats d'une recherche dans une base de données (e.g. ADODB ou PEAR) sont assignés à Smarty

```
<?php
$sql = 'select id, name, home, cell, email from contacts '
       ."where name like '$foo%' ";
$smarty->assign('contacts', $db->getAll($sql));
?>
```

Le template pour afficher le résultat de la base de données dans un tableau HTML

```
<table>
<tr><th> </th><th>Name></th><th>Home</th><th>Cell</th><th>Email</th></tr>
{section name=co loop=$contacts}
  <tr>
    <td><a href="view.php?id={$contacts[co].id}">view<a></td>
    <td>{$contacts[co].name}</td>
    <td>{$contacts[co].home}</td>
    <td>{$contacts[co].cell}</td>
    <td>{$contacts[co].email}</td>
  <tr>
{sectionelse}
  <tr><td colspan="5">Aucun élément n'a été trouvé</td></tr>
{/section}
</table>
```

.index

index contient l'index courant du tableau, en commençant par zéro ou par *start* s'il est fourni. Il s'incrémente d'un en un ou de *step* s'il est fourni.

> **Note technique:** Si les propriétés *step* et *start* ne sont pas modifiés, alors le fonctionnement est le même que celui de la propriété *iteration*, mise à part qu'il commence à zéro au lieu de un.

Exemple 7.37. Exemple avec la propriété `index`

> **FYI:** `$custid[customer.index]` et `$custid[customer]` sont identiques.

```
{section name=customer loop=$custid}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}
```

L'exemple ci-dessus affichera :

```
0 id: 1000<br />
1 id: 1001<br />
```

```
2 id: 1002<br />
```

.index_prev

index_prev est l'index de la boucle précédente. Lors de la première boucle, il vaut -1.

.index_next

index_next est l'index de la prochaine boucle. Lors de la prochaine boucle, il vaudra un de moins que l'index courant, suivant la configuration de l'attribut *step*, s'il est fourni.

Exemple 7.38. Exemple avec les propriétés `index`, `index_next` et `index_prev`

```
<?php
$data = array(1001,1002,1003,1004,1005);
$smarty->assign('rows',$data);
?>
```

Le template pour afficher le tableau ci-dessus dans un tableau HTML

```
{* $rows[row.index] et $rows[row] sont identiques *}
<table>
  <tr>
    <th>index</th><th>id</th>
    <th>index_prev</th><th>prev_id</th>
    <th>index_next</th><th>next_id</th>
  </tr>
{section name=row loop=$rows}
  <tr>
    <td>{$smarty.section.row.index}</td><td>{$rows[row]}</td>
    <td>{$smarty.section.row.index_prev}</td><td>{$rows[row.index_prev]}</td>
    <td>{$smarty.section.row.index_next}</td><td>{$rows[row.index_next]}</td>
  </tr>
{/section}
</table>
```

L'exemple ci-dessus affichera un tableau HTML contenant :

```
index  id    index_prev prev_id index_next next_id
0      1001  -1                 1          1002
1      1002  0          1001    2          1003
2      1003  1          1002    3          1004
3      1004  2          1003    4          1005
4      1005  3          1004    5
```

.iteration

iteration contient l'itération courante de la boucle et commence à un.

> **NOTE:** Ceci n'est pas affecté par les propriétés `{section}` *start*, *step* et *max* contrairement à la propriété *index*. *iteration* commence également à un au lieu de zéro contrairement à *index*. *rownum* est un alias de *iteration*, ils sont identiques.

Exemple 7.39. Exemple avec la propriété `iteration`

```
<?php
// array of 3000 to 3015
$id = range(3000,3015);
$smarty->assign('arr',$id);
?>
```

Le template pour afficher tous les autres éléments du tableau `$arr` comme `step=2`

```
{section name=cu loop=$arr start=5 step=2}
  iteration={$smarty.section.cu.iteration}
  index={$smarty.section.cu.index}
  id={$custid[cu]}<br />
{/section}
```

L'exemple ci-dessus affichera :

```
iteration=1 index=5 id=3005<br />
iteration=2 index=7 id=3007<br />
iteration=3 index=9 id=3009<br />
iteration=4 index=11 id=3011<br />
iteration=5 index=13 id=3013<br />
iteration=6 index=15 id=3015<br />
```

Un autre exemple d'utilisation de la propriété *iteration* est d'afficher un bloc d'en-tête d'un tableau toutes les 5 lignes. Utilisez la fonction `{if}` avec l'opérateur mod.

```
<table>
{section name=co loop=$contacts}
  {if $smarty.section.co.iteration % 5 == 1}
    <tr><th> </th><th>Name></th><th>Home</th><th>Cell</th><th>Email</th></tr>
  {/if}
  <tr>
    <td><a href="view.php?id={$contacts[co].id}">view<a></td>
    <td>{$contacts[co].name}</td>
    <td>{$contacts[co].home}</td>
    <td>{$contacts[co].cell}</td>
    <td>{$contacts[co].email}</td>
  <tr>
{/section}
</table>
```

.first

first est défini à `TRUE` si l'itération courante de `{section}` est l'initiale.

.last

last est défini à `TRUE` si l'itération courante de la section est la dernière.

Exemple 7.40. Exemple avec les propriétés `first` et `last`

Cet exemple boucle sur le tableau `$customers`, affiche un bloc d'en-tête lors de la première itération et, lors de la dernière, affiche un bloc de pied de page. Utilise aussi la propriété *total*.

```
{section name=customer loop=$customers}
  {if $smarty.section.customer.first}
    <table>
    <tr><th>id</th><th>customer</th></tr>
```

```
  {/if}

  <tr>
    <td>{$customers[customer].id}</td>
    <td>{$customers[customer].name}</td>
  </tr>

  {if $smarty.section.customer.last}
    <tr><td></td><td>{$smarty.section.customer.total} customers</td></tr>
    </table>
  {/if}
{/section}
```

.rownum

rownum contient l'itération courante de la boucle, commençant à un. C'est un alias de *iteration*, ils fonctionnent exactement de la même façon.

.loop

loop contient le dernier index de la boucle de la section. Il peut être utilisé dans ou après la `{section}`.

Exemple 7.41. Exemple avec la propriété `loop`

```
{section name=customer loop=$custid}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}
There are {$smarty.section.customer.loop} customers shown above.
```

L'exemple ci-dessus affichera :

```
0 id: 1000<br />
1 id: 1001<br />
2 id: 1002<br />
There are 3 customers shown above.
```

.show

show est utilisé en tant que paramètre à la section et est une valeur booléenne. S'il vaut `FALSE`, la section ne sera pas affichée. S'il y a un `{sectionelse}`, il sera affiché de façon alternative.

Exemple 7.42. Exemple avec la propriété `show`

Une valeur booléenne `$show_customer_info` est passée depuis l'application PHP, pour réguler l'affichage ou non de cette section.

```
{section name=customer loop=$customers show=$show_customer_info}
  {$smarty.section.customer.rownum} id: {$customers[customer]}<br />
{/section}

{if $smarty.section.customer.show}
  the section was shown.
{else}
  the section was not shown.
```

```
{/if}
```

L'exemple ci-dessus affichera :

```
1 id: 1000<br />
2 id: 1001<br />
3 id: 1002<br />

the section was shown.
```

.total

total contient le nombre d'itérations que cette `{section}` bouclera. Il peut être utilisé dans ou après une `{section}`.

Exemple 7.43. Exemple avec la propriété `total`

```
{section name=customer loop=$custid step=2}
  {$smarty.section.customer.index} id: {$custid[customer]}<br />
{/section}
   There are {$smarty.section.customer.total} customers shown above.
```

Voir aussi `{foreach}` et *$smarty.section*.

{strip}

Il est fréquent que les designers web rencontrent des problèmes dus aux espaces et retours chariots, qui affectent le rendu HTML ("fonctionnalités" des navigateurs), les obligeant à coller les balises les unes aux autres. Cette solution rend généralement le code illisible et impossible à maintenir.

Tout contenu situé entre les balises `{strip}{/strip}` se verra allégé des espaces superflus et des retours chariots en début ou en fin de ligne, avant qu'il ne soit affiché. De la sorte, vous pouvez conserver vos templates lisibles, sans vous soucier des effets indésirables que peuvent apporter les espaces superflus.

> **NOTE:** `{strip}{/strip}` n'affecte en aucun cas le contenu des variables de template. Voir aussi le modificateur strip pour un rendu identique pour les variables.

Exemple 7.44. Balises strip

```
{* la suite sera affichée sur une seule ligne *}
{strip}
<table border='0'>
 <tr>
  <td>
   <a href="{$url}">
    <font color="red">Un test</font>
   </a>
  </td>
 </tr>
</table>
{/strip}
```

L'exemple ci-dessus affichera :

```
<table border='0'><tr><td><a href="http://mon.example.com"><font color="red">Un test</font></a></td></t
```

Notez que dans l'exemple ci-dessus, toutes les lignes commencent et se terminent par des balises HTML. Sachez que si vous avez du texte en début ou en fin de ligne dans des balises strip, ce dernier sera collé au suivant/précédent et risque de ne pas être affiché selon l'effet désiré.

Voir aussi le modificateur `strip`.

Chapitre 8. Fonctions utilisateur

Table des matières

Smarty est livré avec plusieurs fonctions utilisateurs que vous pouvez appeler dans vos templates.

{assign}

`{assign}` est utilisé pour déclarer des variables de template **durant l'exécution du template**.

Nom attribut	Type	Requis	Defaut	Description
var	chaîne de caractère	Oui	*n/a*	Le nom de la variable assignée
value	chaîne de caractère	Oui	*n/a*	La valeur assignée

Exemple 8.1. {assign}

```
{assign var='name' value='Bob'}

La valeur de $name est {$name}.
```

L'exemple ci-dessus affichera :

```
La valeur de $name est Bob.
```

Exemple 8.2. {assign} avec quelques fonctions mathématiques

Cet exemple complexe doit avoir ces variables entre crochets.

```
{assign var=running_total value=`$running_total+$some_array[loop].some_value`}
```

Exemple 8.3. Accès aux variables {assign} depuis un script PHP

Pour accéder aux variables `{assign}` depuis le script PHP, utilisez `get_template_vars()`. Ci-dessous, le template qui crée la variable *$foo*.

```
{assign var='foo' value='Smarty'}
```

Les variables de template ne sont disponibles que après/durant l'exécution du template, comme dans le script ci-dessous.

```
<?php

// ceci n'affichera rien car le template n'a pas encore été exécuté
echo $smarty->get_template_vars('foo');

// Récupère le template dans une variable
$whole_page = $smarty->fetch('index.tpl');

// Ceci affichera 'smarty' car le template a été exécuté
echo $smarty->get_template_vars('foo');

$smarty->assign('foo','Even smarter');

// Ceci affichera 'Even smarter'
echo $smarty->get_template_vars('foo');

?>
```

Les fonctions suivantes peuvent *optionnellement* assigner des variables de template.

`{capture}`, `{include}`, `{include_php}`, `{insert}`, `{counter}`, `{cycle}`, `{eval}`, `{fetch}`, `{math}` et `{textformat}`.

Voir aussi `assign()` et `get_template_vars()`.

{counter}

`{counter}` affiche un compteur. `{counter}` retient la valeur du compte à chaque itération. Vous pouvez adapter le nombre, l'intervale et la direction du compteur, ainsi que décider d'afficher ou non les valeurs. Vous pouvez lancer plusieurs compteurs simultanément en leur donnant des noms uniques. Si vous ne donnez pas de nom à un compteur, «default» sera utilisé.

Si vous donnez une valeur à l'attribut *assign*, alors la sortie de la fonction `{counter}` sera assignée à la variable de template donnée plutôt que d'être directement affichée.

Nom attribut	Type	Requis	Defaut	Description
name	chaîne de caractère	Non	*default*	Le nom du compteur
start	numérique	Non	*1*	La valeur initiale du compteur
skip	numérique	Non	*1*	L'intervale du compteur
direction	chaîne de caractères	Non	*up*	la direction du

Nom attribut	Type	Requis	Defaut	Description
				compteur (up/down) [compte / décompte]
print	booléen	Non	*TRUE*	S'il faut afficher cette valeur ou non
assign	chaîne de caractères	Non	*n/a*	La variable dans laquelle la valeur du compteur sera assignée.

Exemple 8.4. {counter}

```
{* initialisation du compteur *}
{counter start=0 skip=2}<br />
{counter}<br />
{counter}<br />
{counter}<br />
```

L'exemple ci-dessus affichera :

```
0<br />
2<br />
4<br />
6<br />
```

{cycle}

`{cycle}` est utilisé pour boucler sur un ensemble de valeurs. Très pratique pour alterner entre deux ou plusieurs couleurs dans un tableau, ou plus généralement pour boucler sur les valeurs d'un tableau.

Nom attribut	Type	Requis	Defaut	Description
name	chaîne de caractères	Non	*default*	Le nom du cycle
values	divers	Oui	*N/A*	Les valeurs sur lesquelles boucler, soit une liste séparée par des virgules, (voir l'attribut delimiter), soit un tableau de valeurs
print	booléen	Non	*TRUE*	S'il faut afficher ou non cette valeur
advance	booléen	Non	*TRUE*	Oui ou non aller à la prochaîne valeur
delimiter	chaîne de caractères	Non	,	Le délimiteur à utiliser dans la liste.
assign	chaîne de caractères	Non	*n/a*	La variable de template dans laquelle la sortie sera assignée
reset	booléen	Non	*FALSE*	Le cycle sera défini à la première valeur

- Vous pouvez définir plusieurs `{cycle}` dans votre template, en leur donnant des noms uniques (attribut *name*).
- Vous pouvez empêcher la valeur courante de s'afficher en définissant l'attribut *print* à `FALSE`. Ce procédé peut être utile pour discrètement passer outre une valeur de la liste.
- L'attribut *advance* est utilisé pour répéter une valeur. Lorsque définit à `FALSE`, le prochain appel de `{cycle}` ramènera la même valeur.
- Si vous définissez l'attribut spécial *assign*, la sortie de la fonction `{cycle}` y sera assignée plutôt que d'être directement affichée.

Exemple 8.5. {cycle}

```
{section name=rows loop=$data}
<tr bgcolor="{cycle values="#eeeeee,#d0d0d0"}">
   <td>{$data[rows]}</td>
</tr>
{/section}
```

Le template ci-dessus affichera :

```
<tr bgcolor="#eeeeee">
   <td>1</td>
</tr>
<tr bgcolor="#d0d0d0">
   <td>2</td>
</tr>
<tr bgcolor="#eeeeee">
   <td>3</td>
</tr>
```

{debug}

`{debug}` amène la console de débogage sur la page. Fonctionne quelle que soit la valeur du paramètre debug de Smarty. Comme ce dernier est appelé lors de l'exécution, il n'est capable d'afficher que les variables assignées au template, et non les templates en cours d'utilisation. Toutefois, vous voyez toutes les variables disponibles pour le template courant.

Nom attribut	Type	Requis	Defaut	Description
output	chaîne de caractères	Non	*javascript*	Type de sortie, html ou javascript

Voir aussi la console de débogage.

{eval}

`{eval}` évalue une variable comme si cette dernière était un template. Peut être utile pour embarquer des balises de templates ou des variables de template dans des variables ou des balises/variables dans des variables de fichiers de configuration.

Si vous définissez l'attribut *assign*, la sortie sera assignée à la variable de template désignée plutôt que d'être affichée dans le template.

Nom attribut	Type	Requis	Defaut	Description
var	mixed	Oui	*n/a*	Variable (ou chaîne de caractères) à évaluer
assign	chaîne de caractères	Non	*n/a*	Le nom de la variable PHP dans laquelle la sortie sera assignée

Note technique:

- Les variables évaluées sont traitées de la même façon que les templates. Elles suivent les mêmes règles de traitement et de sécurité, comme si elles étaient réellement des templates.
- Les variables évaluées sont compilées à chaque invocation, et la version compilée n'est pas sauvegardée ! Toutefois, si le cache est activé, la sortie sera placée en cache avec le reste du template.

Exemple 8.6. {eval}

Le contenu du fichier de configuration, `setup.conf`.

```
#setup.conf
#----------
emphstart = <strong>
emphend = </strong>
titre = Bienvenue sur la homepage de {$company} !
ErrorVille = Vous devez spécifier un nom de {#emphstart#}ville{#emphend#}.
ErrorDept = Vous devez spécifier un {#emphstart#}département{#emphend#}.
```

Où le template est :

```
{config_load file='setup.conf'}

{eval var=$foo}
{eval var=#titre#}
{eval var=#ErrorVille#}
{eval var=#ErrorDept# assign='state_error'}
{$state_error}
```

L'exemple ci-dessus affichera :

```
Ceci est le contenu de foo.
Bienvenue sur la homepage de FictifLand.
Vous devez spécifier un nom de <strong>ville</strong>.
Vous devez spécifier un <strong>département</strong>.
```

Exemple 8.7. un autre exemple avec {eval}

Ceci va afficher le nom du serveur (en majuscule) et son IP. La variable *$str* également venir d'une requête de base de données.

```
<?php
$str = 'Le nom du serveur est {$smarty.server.SERVER_NAME|upper} '
.'at {$smarty.server.SERVER_ADDR}';
```

```
$smarty->assign('foo',$str);
?>
```

Où le template est :

```
{eval var=$foo}
```

{fetch}

`{fetch}` est utilisé pour récupérer des fichiers depuis le système de fichier local, depuis un serveur http ou ftp, et en afficher le contenu.

- Si le nom du fichier commence par *http://*, la page internet sera récupérée, puis affichée.

  NOTE: Ceci ne supporte pas les redirections http. Assurez vous d'inclure les slash de fin sur votre page web si nécessaire.

- Si le nom du fichier commence par *ftp://*, le fichier sera récupéré depuis le serveur ftp, et affiché.

- Pour les fichiers du système local, le chemin doit être absolu ou relatif au chemin d'exécution du script PHP.

  NOTE: Si la variable de template *$security* est activée et que vous récupérez un fichier depuis le système de fichiers local, `{fetch}` ne permettra que les fichiers se trouvant dans un des dossiers définis dans les dossiers sécurisés.

- Si l'attribut *assign* est défini, l'affichage de la fonction `{fetch}` sera assignée à cette variable de template au lieu d'être affichée dans le template.

Nom attribut	Type	Requis	Defaut	Description
file	chaîne de caractères	Oui	*n/a*	Le fichier, site http ou ftp à récupérer
assign	chaîne de caractères	Non	*n/a*	Le nom de la variable PHP dans laquelle la sortie sera assignée plutôt que d'être directement affichée.

Exemple 8.8. Exempe avec {fetch}

```
{* Inclus du javascript dans votre template *}
{fetch file='/export/httpd/www.example.com/docs/navbar.js'}

{* récupère les informations météo d'un autre site sur votre page *}
{fetch file='http://www.myweather.com/68502/'}

{* récupère les titres depuis un fichier ftp *}
{fetch file='ftp://user:password@ftp.example.com/path/to/currentheadlines.txt'}
{* comme ci-dessus mais avec des variables *}
{fetch file="ftp://`$user`:`$password`@`$server`/`$path`"}

{* assigne le contenu récupéré à une variable de template *}
{fetch file='http://www.myweather.com/68502/' assign='weather'}
{if $weather ne ''}
```

```
<div id="weather">{$weather}</div>
{/if}
```

Voir aussi `{capture}`, `{assign}` `{eval}` et `fetch()`.

{html_checkboxes}

`{html_checkboxes}` est une fonction utilisateur qui crée un groupe de cases à cocher avec les données fournies. Elle prend en compte la liste des éléments sélectionnés par défaut.

Nom attribut	Type	Requis	Defaut	Description
name	chaîne de caractères	Non	*checkbox*	Nom de la liste de cases à cocher
values	array	Oui, à moins que vous n'utilisiez l'attribut option	*n/a*	Un tableau de valeurs pour les cases à cocher
output	array	Oui, à moins que vous n'utilisiez l'attribut option	*n/a*	Un tableau de sortie pour les cases à cocher
selected	chaîne de caractères/ tableau	Non	*empty*	Les éléments cochés de la liste
options	Tableau associatif	Oui, à moins que vous n'utilisiez values et output	*n/a*	Un tableau associatif de valeurs et sorties
separator	chaîne de caractères	Non	*empty*	chaîne de caractère pour séparer chaque case à cocher
assign	chaîne de caractères	Non	*empty*	Assigne les balises d'un checkbox à un tableau plutôt que de les afficher
labels	booléen	Non	*true*	Ajoute la balise <label>- à la sortie
assign	chaîne de caractères	Non	*empty*	Assigne la sortie à un tableau dont chaque checkbox est un élément.

- Les attributs requis sont *values* et *output*, à moins que vous utilisez *options* à la place.
- Tous les affichages sont conformes XHTML.
- Tous les paramètres qui ne sont pas dans la liste ci-dessus sont affichés sous la forme de paires nom/valeur dans chaque balise <input> crées.

Exemple 8.9. {html_checkboxes}

```
<?php
```

```
$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names', array(
'Joe Schmoe',
'Jack Smith',
'Jane Johnson',
'Charlie Brown')
);
$smarty->assign('customer_id', 1001);

?>
```

où index.tpl est :

```
{html_checkboxes name='id' values=$cust_ids output=$cust_names
         selected=$customer_id  separator='<br />'}
```

ou bien, le code PHP est :

```
<?php

$smarty->assign('cust_checkboxes', array(
1000 => 'Joe Schmoe',
1001 => 'Jack Smith',
1002 => 'Jane Johnson',
1003 => 'Charlie Brown')
);
$smarty->assign('customer_id', 1001);

?>
```

et index.tpl est :

```
{html_checkboxes name='id' options=$cust_checkboxes
          selected=$customer_id separator='<br />'}
```

Les deux examples donnent à l'écran :

```
<label><input type="checkbox" name="id[]" value="1000" />Joe Schmoe</label><br />
<label><input type="checkbox" name="id[]" value="1001" checked="checked" />Jack Smith</label>
<br />
<label><input type="checkbox" name="id[]" value="1002" />Jane Johnson</label><br />
<label><input type="checkbox" name="id[]" value="1003" />Charlie Brown</label><br />
```

Exemple 8.10. Exemple avec une base de données (eg PEAR ou ADODB) :

```
<?php

$sql = 'select type_id, types from contact_types order by type';
$smarty->assign('contact_types',$db->getAssoc($sql));

$sql = 'select contact_id, contact_type, contact from contacts where contact_id=12';
$smarty->assign('contact',$db->getRow($sql));

?>
```

Le résultat des requêtes de la base de données sera affiché avec :

```
{html_checkboxes name='contact_type_id' options=$contact_types
         selected=$contact.contact_type_id separator='<br />'}
```

Voir aussi {html_radios} et {html_options}.

{html_image}

{html_image} est une fonction utilisateur qui génère la balise HTML pour une image. La hauteur et la longueur de l'image sont calculés automatiquement depuis le fichier image si aucune n'est spécifiée.

Nom attribut	Type	Requis	Défaut	Description
file	chaîne de caractères	Oui	*n/a*	nom/chemin des images
height	chaîne de caractères	Non	*Hauteur de l'image actuelle*	Hauteur de l'image à afficher
width	chaîne de caractères	Non	*Longueur de l'image actuelle*	Longueur de l'image à afficher
basedir	chaîne de caractères	non	*racine du serveur web*	Répertoire depuis lequel baser le calcul des chemins relatifs
alt	chaîne de caractères	non	«»	Description alternative de l'image
href	chaîne de caractères	non	*n/a*	valeur de l'attribut href, indiquant le lien vers l'image
path_prefix	chaîne de caractères	non	*n/a*	Préfixe pour le chemin de la sortie

- *basedir* est le dossier de base dans lequel les images sont basées. S'il n'est pas fourni, la variable d'environnement $_ENV['DOCUMENT_ROOT'] sera utilisée. Si *$security* est activé, le chemin vers l'image doit être présent dans le dossier de sécurité.
- *href* est la valeur de l'attribut href de l'image. Si le lien est fourni, une balise <a href="LINKVALUE"><a> sera placée autour de la balise de l'image.
- *path_prefix* est un préfixe optionnel que vous pouvez fournir. Il est utile si vous voulez fournir un nom de serveur différent pour l'image.
- Tous les paramètres qui ne sont pas dans la liste ci-dessus sont affichés sous la forme d'une paire nom/valeur dans la balise <img> créée.

> **Note technique:** {html_image} requiert un accès au disque dur pour lire l'image et calculer ses dimensions. Si vous n'utilisez pas un cache, il est généralement préférable d'éviter d'utiliser {html_image} et de laisser les balises images statiques pour de meilleures performances.

Exemple 8.11. Exemple avec {html_image}

```
{html_image file='pumpkin.jpg'}
{html_image file='/path/from/docroot/pumpkin.jpg'}
{html_image file='../path/relative/to/currdir/pumpkin.jpg'}
```

L'affichage possible du template ci-dessus pourrait être :

```
<img src="pumpkin.jpg" alt="" width="44" height="68" />
<img src="/path/from/docroot/pumpkin.jpg" alt="" width="44" height="68" />
<img src="../path/relative/to/currdir/pumpkin.jpg" alt="" width="44" height="68" />
```

{html_options}

`{html_options}` est une fonction personnalisée qui crée un groupe d'options avec les données fournies. Elle prend en charge les éléments sélectionnés par défaut.

Nom attribut	Type	Requis	Defaut	Description
values	array	Oui, à moins que vous n'utilisiez l'attribut options	*n/a*	Un tableau de valeurs pour les listes déroulantes
output	array	Oui, à moins que vous n'utilisiez l'attribut options	*n/a*	Un tableau de libellés pour la liste déroulante
selected	chaîne de caractères/ tableau	Non	*empty*	Les éléments sélectionnés
options	Tableau associatif	Oui, à moins que vous n'utilisiez option et values	*n/a*	Un tableau associatif valeur / libellé
name	chaîne de caractères	Non	*empty*	Nom du goupe d'options

- Les attributs requis sont *values* et *output*, à moins que vous n'utilisiez *options* à la place.
- Si l'attribut optionnel *name* est fourni, les balises `<select></select>` seront créées, sinon, UNIQUEMENT la liste `<option>` sera générée.
- Si la valeur fournie est un tableau, il sera traité comme un `<optgroup>` HTML, et affichera les groupes. La récursivité est supportée avec `<optgroup>`.
- Tous les paramètres qui ne sont pas dans la liste ci-dessus sont affichés sous la forme de paire nom/valeur dans la balise `<select>`. Ils seront ignorés si le paramètre optionnel *name* n'est pas fourni.
- Tous les affichages sont conformes XHTML.

Exemple 8.12. Un tableau associatif avec l'attribut `options`

```
<?php
$smarty->assign('myOptions', array(
                                1800 => 'Joe Schmoe',
                                9904 => 'Jack Smith',
                                2003 => 'Charlie Brown')
);
$smarty->assign('mySelect', 9904);
?>
```

Le template suivant génèrera une liste. Notez la présence de l'attribut *name* qui crée les balises `<select>`.

```
{html_options name=foo options=$myOptions selected=$mySelect}
```

L'affichage de l'exemple ci-dessus sera :

```
<select name="foo">
 <option label="Joe Schmoe" value="1800">Joe Schmoe</option>
 <option label="Jack Smith" value="9904" selected="selected">Jack Smith</option>
 <option label="Charlie Brown" value="2003">Charlie Brown</option>
</select>
```

Exemple 8.13. Tableaux séparés pour `values` et `ouptut`

```
<?php
$smarty->assign('cust_ids', array(56,92,13));
$smarty->assign('cust_names', array(
                                'Joe Schmoe',
                                'Jane Johnson',
                                'Charlie Brown'));
$smarty->assign('customer_id', 92);
?>
```

Les tableaux ci-dessus seront affichés avec le template suivant (notez l'utilisation de la fonction PHP `count()` [http://php.net/function.count] en tant que modificateur pour définir la taille du select).

```
<select name="customer_id" size="{$cust_names|@count}">
 {html_options values=$cust_ids output=$cust_names selected=$customer_id}
</select>
```

L'exemple ci-dessous affichera :

```
<select name="customer_id">
 <option label="Joe Schmoe" value="56">Joe Schmoe</option>
 <option label="Jack Smith" value="92" selected="selected">Jane Johnson</option>
 <option label="Charlie Brown" value="13">Charlie Brown</option>
</select>
```

Exemple 8.14. Exemple avec une base de données (e.g. ADODB ou PEAR)

```
<?php

$sql = 'select type_id, types from contact_types order by type';
$smarty->assign('contact_types',$db->getAssoc($sql));

$sql = 'select contact_id, name, email, contact_type_id
from contacts where contact_id='.$contact_id;
$smarty->assign('contact',$db->getRow($sql));

?>
```

Où le template pourrait être celui-ci. Notez l'utilisation du modificateur `truncate`.

```
<select name="type_id">
 <option value='null'>-- none --</option>
 {html_options options=$contact_types|truncate:20 selected=$contact.type_id}
</select>
```

Exemple 8.15. Exemple avec <optgroup>

```
<?php
$arr['Sport'] = array(6 => 'Golf', 9 => 'Cricket',7 => 'Swim');
$arr['Rest']  = array(3 => 'Sauna',1 => 'Massage');
$smarty->assign('lookups', $arr);
$smarty->assign('fav', 7);
?>
```

Le script ci-dessus et le template suivant

```
{html_options name=foo options=$myOptions selected=$mySelect}
```

affichera :

```
<select name="breakTime">
 <optgroup label="Sport">
  <option label="Golf" value="6">Golf</option>
  <option label="Cricket" value="9">Cricket</option>
  <option label="Swim" value="7" selected="selected">Swim</option>
 </optgroup>
 <optgroup label="Rest">
  <option label="Sauna" value="3">Sauna</option>
  <option label="Massage" value="1">Massage</option>
 </optgroup>
</select>
```

Voir aussi `{html_checkboxes}` et `{html_radios}`

{html_radios}

`{html_radios}` est une fonction personnalisée qui crée des boutons radio html à partir des données fournies. Elle prend en charge les éléments sélectionnés par défaut.

Nom attribut	Type	Requis	Defaut	Description
name	chaîne de caractères	Non	*radio*	Nom de la liste boutons radio
values	tableau	Oui, à moins que vous n'utilisiez l'attribut options	*n/a*	Le tableau des valeurs des boutons radio
output	tableau	Oui, à moins que vous n'utilisiez l'attribut options	*n/a*	Un tableau de libellés pour les boutons radio
checked	chaîne de caractères	Non	*empty*	Les boutons radios sélectionnés
options	tableau associatif	Oui, à moins que vous n'utilisiez values et outputs	*n/a*	Un tableau associatif valeurs / libellés
separator	chaîne de caractères	Non	*empty*	Chaîne de séparation à placer entre les boutons radio
assign	chaîne de caractères	Non	*empty*	Assigne les balises des boutons radio à un

Nom attribut	Type	Requis	Defaut	Description
				tableau plutôt que de les afficher

- Les attributs requis sont *values* et *output*, à moins que vous n'utilisez *options* à la place.
- Tous les affichages sont conformes XHTML.
- Tous les paramètres qui ne sont pas dans la liste ci-dessus sont affichés sous la forme de paire nom/valeur dans la balise `<input>` créées.

Exemple 8.16. {html_radios} : Première exemple

```
<?php

$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names', array(
                                'Joe Schmoe',
                                'Jack Smith',
                                'Jane Johnson',
                                'Charlie Brown')
                              );
$smarty->assign('customer_id', 1001);

?>
```

Où le template est :

```
{html_radios name='id' values=$cust_ids output=$cust_names
       selected=$customer_id separator='<br />'}
```

Exemple 8.17. {html_radios} : Deuxième exemple

```
<?php

$smarty->assign('cust_radios', array(
                                1000 => 'Joe Schmoe',
                                1001 => 'Jack Smith',
                                1002 => 'Jane Johnson',
                                1003 => 'Charlie Brown'));
$smarty->assign('customer_id', 1001);

?>
```

Où le template est :

```
{html_radios name='id' options=$cust_radios
   selected=$customer_id separator='<br />'}
```

Les deux exemples ci-dessus afficheront :

```
<label for="id_1000">
 <input type="radio" name="id" value="1000" id="id_1000" />Joe Schmoe</label><br />
<label for="id_1001"><input type="radio" name="id" value="1001" id="id_1001" checked="checked" />Jack S
<label for="id_1002"><input type="radio" name="id" value="1002" id="id_1002" />Jane Johnson</label><br
```

```
<label for="id_1003"><input type="radio" name="id" value="1003" id="id_1003" />Charlie Brown</label><br
```

Exemple 8.18. {html_radios} - Exemple avec une base de données (e.g. PEAR ou ADODB):

```
<?php

$sql = 'select type_id, types from contact_types order by type';
$smarty->assign('types',$db->getAssoc($sql));

$sql = 'select contact_id, name, email, contact_type_id
     from contacts where contact_id='.$contact_id;
$smarty->assign('contact',$db->getRow($sql));

?>
```

La variable assignée depuis la base de données ci-dessus sera affichée avec le template :

```
{html_radios name='contact_type_id' options=$contact_types
     selected=$contact.contact_type_id separator='<br />'}
```

Voir aussi `{html_checkboxes}` et `{html_options}`.

{html_select_date}

`{html_select_date}` est une fonction personnalisée qui crée des listes déroulantes pour saisir la date. Elle peut afficher n'importe quel jour, mois et année. Tous les paramètres qui ne sont pas dans la liste ci-dessous sont affichés sous la forme pair nom/valeur dans les balises `<select>` des jours, mois et années.

Nom attribut	Type	Requis	Défaut	Description
prefix	chaîne de caractères	Non	Date_	Avec quoi préfixer le nom de variable
time	timestamp/ YYYY-MM-DD	Non	la date courante au format unix YYYY-MM-DD format	La date / heure à utiliser
start_year	chaîne de caractères	Non	current year	La première année dans la liste déroulante, soit le numéro de l'année, soit un nombre relatif à l'année courante (+/- N).
end_year	chaîne de caractères	Non	même chose que start_year	La dernière année dans la liste déroulante, soit le numéro de l'année, soit un nombre relatif à l'année courante (+/- N).
display_days	boolean	Non	true	Si l'on souhaite afficher les jours ou pas.
display_months	boolean	Non	true	Si l'on souhaite afficher les mois ou pas.
display_years	boolean	Non	true	Si l'on souhaite afficher

Nom attribut	Type	Requis	Défaut	Description
				les années ou pas.
month_format	chaîne de caractères	Non	%B	le format du mois (strftime)
day_format	chaîne de caractères	Non	%02d	Le format du jour (sprintf)
day_value_format	chaîne de caractères	Non	%d	Le format de la valeur du jour (sprintf)
year_as_text	boolean	Non	false	S'il faut afficher l'année au format texte
reverse_years	boolean	Non	false	Affiche les années dans l'ordre inverse
field_array	chaîne de caractères	Non	null	Si un nom est donné, la liste déroulante sera affichée de telle façon que les résultats seront retournés à PHP sous la forme nom[Day] (jour), nom[Year] (année), nom[Month] (Mois).
day_size	chaîne de caractères	Non	null	Ajoute un attribut size à la liste déroulante des jours.
month_size	chaîne de caractères	Non	null	Ajoute un attribut size à la liste déroulante des mois.
year_size	chaîne de caractères	Non	null	Ajoute un attribut size à la liste déroulante des années.
all_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires à toutes les balises select/input.
day_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires aux balises select/input du jour.
month_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires aux balises select/input du mois.
year_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires aux balises select/input de l'année.
field_order	chaîne de caractères	Non	MDY	L'ordre dans lequel afficher les listes déroulantes.
field_separator	chaîne de caractères	Non	\n	la chaîne de caractères affichée entre les différents champs.
month_value_format	chaîne de caractères	Non	%m	Le format strftime de la

Nom attribut	Type	Requis	Défaut	Description
				valeur des mois, par défaut %m pour les numéros.
year_empty	chaîne de caractères	Non	null	S'il est renseigné, alors le premier élément de la boite de sélection affiche le texte donné en tant que libellé et dispose de la valeur «». Utile par exemple lorsque vous souhaitez que la boite de sélection affiche «Sélectionnez une année». A savoir que vous pouvez spécifier des valeurs de la forme «-MM-DD» pour l'attribut time afin d'indiquer une année non sélectionnée.
month_empty	chaîne de caractères	Non	null	S'il est renseigné, le premier élément de la boite de sélection affiche le texte donné en tant que libellé et dispose de la valeur «». A savoir que vous pouvez spécifier des valeurs de la forme «YYYY--DD» pour l'attribut time afin d'indiquer qu'il manque le moi.
day_empty	chaîne de caractères	Non	null	S'il est renseigné, le premier élément de la boite de sélection affiche le texte donné en tant que libellé et dispose de la valeur «». A savoir que vous pouvez spécifier des valeurs de la forme «YYYY-MM-» pour l'attribut time afin d'indiquer qu'il manque le jour.

NOTE: Il y a une fonction PHP utile sur la page des astuces sur les dates pour convertir les valeurs `{html_select_date}` en un timestamp.

Exemple 8.19. {html_select_date} : Premier exemple

Code du template

```
{html_select_date}
```

Ce qui donne en sortie :

```
<select name="Date_Month">
  <option value="1">January</option>
  <option value="2">February</option>
  <option value="3">March</option>
  ..... coupé .....
  <option value="10">October</option>
  <option value="11">November</option>
  <option value="12" selected="selected">December</option>
</select>
<select name="Date_Day">
  <option value="1">01</option>
  <option value="2">02</option>
  <option value="3">03</option>
  ..... coupé .....
  <option value="11">11</option>
  <option value="12">12</option>
  <option value="13" selected>13</option>
  <option value="14">14</option>
  <option value="15">15</option>
  ..... coupé .....
  <option value="29">29</option>
  <option value="30">30</option>
  <option value="31">31</option>
</select>
<select name="Date_Year">
  <option value="2006" selected="selected">2006</option>
</select>
```

Exemple 8.20. {html_select_date} : Deuxième exemple

```
{* le démarage et la fin de l'année peuvent être relatif à l'année courante *}
{html_select_date prefix="StartDate" time=$time start_year="-5"
   end_year="+1" display_days=false}
```

Ce qui donne en sortie: (L'année courante est 2000)

```
<select name="StartDateMonth">
  <option value="1">January</option>
  <option value="2">February</option>
  ..... coupé .....
  <option value="11">November</option>
  <option value="12" selected="selected">December</option>
</select>
<select name="StartDateYear">
 <option value="1995">1995</option>
  ..... coupé .....
  <option value="1999">1999</option>
  <option value="2000" selected="selected">2000</option>
  <option value="2001">2001</option>
</select>
```

Voir aussi `{html_select_time}`, `date_format`, *$smarty.now* et les astuces sur les dates.

{html_select_time}

`{html_select_time}` est une fonction personnalisée qui crée des listes déroulantes pour saisir une heure. Elle prends en charge l'heure, les minutes, les secondes et le méridian.

L'attribut *time* accepte comme paramètre différents formats. Ils peuvent être un timestamp unique, une chaîne respectant le format `YYYYMMDDHHMMSS` ou une chaîne valide pour la fonction php `strtotime()` [http://php.net/strtotime].

Nom attribut	Type	Requis	Défaut	Description
prefix	chaîne de caractères	Non	Time_	Par quoi préfixer la variable.
time	timestamp	Non	current time	Quel jour / heure utiliser.
display_hours	boolean	Non	`TRUE`	S'il faut afficher l'heure.
display_minutes	boolean	Non	`TRUE`	S'il faut afficher les minutes.
display_seconds	boolean	Non	`TRUE`	S'il faut afficher les secondes.
display_meridian	boolean	Non	`TRUE`	S'il faut afficher le méridian (am/pm)
use_24_hours	boolean	Non	`TRUE`	S'il faut utiliser l'horloge 24 heure.
minute_interval	integer	Non	1	Intervalle des minutes dans la liste déroulante
second_interval	integer	Non	1	Intervalle des secondes dans la liste déroulante
field_array	chaîne de caractères	Non	n/a	Nom du tableau dans lequel les valeures seront stockées.
all_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires aux balises select / input.
hour_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires aux balises select / input de l'heure.
minute_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires aux balises select / input des minutes.
second_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires aux balises select / input des secondes.
meridian_extra	chaîne de caractères	Non	null	Ajoute des attributs supplémentaires aux balises select / input du méridian.

Exemple 8.21. html_select_time

```
{html_select_time use_24_hours=true}
```

À 9:20 et 23 secondes du matin, le template ci-dessus affichera :

```
<select name="Time_Hour">
<option value="00">00</option>
<option value="01">01</option>
...coupé...
<option value="08">08</option>
<option value="09" selected>09</option>
<option value="10">10</option>
...coupé...
<option value="22">22</option>
<option value="23">23</option>
</select>
<select name="Time_Minute">
<option value="00">00</option>
<option value="01">01</option>
...coupé...
<option value="19">19</option>
<option value="20" selected>20</option>
<option value="21">21</option>
...coupé...
<option value="58">58</option>
<option value="59">59</option>
</select>
<select name="Time_Second">
<option value="00">00</option>
<option value="01">01</option>
...coupé...
<option value="22">22</option>
<option value="23" selected>23</option>
<option value="24">24</option>
...coupé...
<option value="58">58</option>
<option value="59">59</option>
</select>
<select name="Time_Meridian">
<option value="am" selected>AM</option>
<option value="pm">PM</option>
</select>
```

Voir aussi *$smarty.now*, `{html_select_date}` et les astuces sur les dates.

{html_table}

`{html_table}` est une fonction personnalisée qui transforme un tableau de données dans un tabeau HTML.

Nom de l'attribut	Type	Requis	Défaut	Description
loop	tableau	Oui	*n/a*	Tableau de données à parcourir
cols	mixed	Non	*3*	Nombre de colonnes de la table ou une liste de noms de colonnes séparés par une virgule ou un tableau contenant les noms des colonnes. Si l'attribut "cols" est

Nom de l'attribut	Type	Requis	Défaut	Description
				vide, mais que des lignes sont données, alors le nombre de colonnes sera calculé en utilisant le nombre de lignes et le nombre d'éléments à afficher pour qu'il y ait juste assez de colonnes pour afficher tous les éléments. Si les lignes et les colonnes sont omis tous les deux, la valeur par défaut de "cols" sera appliquée, à savoir 3. Si fourni en tant que liste ou tableau, le nombre de colonnes est calculé par rapport au nombre d'éléments de la liste ou du tableau.
rows	entier	No	*empty*	Nombre de lignes de la table. Si l'attribut "rows" est vide, mais que des colonnes sont données, alors le nombre de lignes sera calculée en utilisant le nombre de colonnes et le nombre d'éléments à afficher pour qu'il y ait juste assez de lignes pour afficher tous les éléments.
inner	chaîne de caractères	No	*cols*	La direction du rendu des éléments consécutifs dans la boucle du tableau. *cols* signifie que les éléments doivent être afficher colonnes par colonnes. *rows* signifie que les éléments doivent être afficher lignes par lignes.
caption	chaîne de caractères	No	*empty*	Texte à utiliser pour l'élément `<caption>` du tableau.
table_attr	chaîne de caractères	Non	*border="1"*	attributs pour la balise `<table>`
th_attr	chaîne de caractères	No	*empty*	Attributs pour les balises `<th>` (les tableaux sont parcourus)

Nom de l'attribut	Type	Requis	Défaut	Description
tr_attr	chaîne de caractères	Non	*empty*	Attributs pour les balises `<tr>` (les tableaux sont parcourus)
td_attr	chaîne de caractères	Non	*empty*	Attributs pour les balises `<td>` (les tableaux sont parcourus)
trailpad	chaîne de caractères	Non	* *	Valeur avec laquelle remplir les cellules restantes de la dernière ligne (si il y en a)
hdir	chaîne de caractères	Non	*right*	Direction du rendu. Les valeurs possibles sont *right* (left-to-right), *left* (right-to-left)
vdir	chaîne de caractères	Non	*down*	Direction des colonnes lors du rendu. Les valeurs possibles sont : *down* (top-to-bottom), *up* (bottom-to-top)

- L'attribut *cols* détermine le nombre de colonnes dans le tableau.
- Les valeurs *table_attr*, *tr_attr* et *td_attr* déterminent les attributs fournis dans les balises `<table>`, `<tr>` et `<td>`.
- Si *tr_attr* ou *td_attr* est un tableau, il sera parcourru.
- *trailpad* est la valeur utilisée pour compléter les cellules vides de la dernière ligne s'il y en a.

Exemple 8.22. {html_table}

```
<?php
$smarty->assign('data',array(1,2,3,4,5,6,7,8,9));
$smarty->assign('tr',array('bgcolor="#eeeeee"','bgcolor="#dddddd"'));
$smarty->display('index.tpl');
?>
```

Les variables assignées depuis PHP peuvent être affichées comme le démontre cet exemple.

```
{**** Premier exemple ****}
{html_table loop=$data}

<table border="1">
 <tbody>
  <tr><td>1</td><td>2</td><td>3</td></tr>
  <tr><td>4</td><td>5</td><td>6</td></tr>
  <tr><td>7</td><td>8</td><td>9</td></tr>
 </tbody>
</table>


{**** Deuxième exemple ****}
{html_table loop=$data cols=4 table_attr='border="0"'}
```

```
<table border="0">
 <tbody>
  <tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
  <tr><td>5</td><td>6</td><td>7</td><td>8</td></tr>
  <tr><td>9</td><td> </td><td> </td><td> </td></tr>
 </tbody>
</table>


{**** Troisième exemple ****}
{html_table loop=$data cols="first,second,third,fourth" tr_attr=$tr}

<table border="1">
 <thead>
  <tr>
   <th>first</th><th>second</th><th>third</th><th>fourth</th>
  </tr>
 </thead>
 <tbody>
  <tr bgcolor="#eeeeee"><td>1</td><td>2</td><td>3</td><td>4</td></tr>
  <tr bgcolor="#dddddd"><td>5</td><td>6</td><td>7</td><td>8</td></tr>
  <tr bgcolor="#eeeeee"><td>9</td><td> </td><td> </td><td> </td></tr>
 </tbody>
</table>
```

{mailto}

`{mailto}` crée un lien `mailto:` automatiquement encodé (optionnel). L'encodage rend la tâche de récupération des e-mails sur votre site plus difficiles aux "web spiders".

> **Note technique:** Javascript n'est certainement pas la forme d'encodage la plus robuste. Vous pouvez également utiliser un encodage hexadécimal.

Nom attribut	Type	Requis	Défaut	Description
address	chaîne de caractères	Oui	*n/a*	L'adresse email
text	chaîne de caractères	Non	*n/a*	Le texte à afficher, par défaut l'adresse email
encode	chaîne de caractères	Non	*none*	Comment encoder l'adresse email. `none`, `hex`, `javascript` et `javascript_charcode` sont des valeurs correctes.
cc	chaîne de caractères	Non	*n/a*	Les adresses email en copie (Cc). Séparez les entrées par une virgule.
bcc	chaîne de caractères	Non	*n/a*	Les adresses email en copie cachées (Bcc). Séparez les entrées par une virgule.
subject	chaîne de caractères	Non	*n/a*	Sujet de l'email.
newsgroups	chaîne de caractères	Non	*n/a*	Newsgroup où poster le message. Séparez les entrées par une virgule.
followupto	chaîne de caractères	Non	*n/a*	Adresses où transmettre le message.

Nom attribut	Type	Requis	Défaut	Description
				Séparez les entrées par une virgule.
extra	chaîne de caractères	Non	*n/a*	Toute information que vous souhaitez passer au lien, par exemple une classe css.

Exemple 8.23. Exemple avec {mailto}

```
{mailto address="moi@example.com"}
<a href="mailto:moi@example.com" >moi@example.com</a>

{mailto address="moi@example.com" text="envoie moi un email"}
<a href="mailto:moi@example.com" >envoie-moi un email</a>

{mailto address="moi@example.com" encode="javascript"}
<script type="text/javascript" language="javascript">
 eval(unescape('%64%6f% ... coupé ...%61%3e%27%29%3b'))
</script>

{mailto address="moi@example.com" encode="hex"}
<a href="mailto:%6d%65.. coupé..3%6f%6d">&#x6d;&..coupé...#x6f;&#x6d;</a>

{mailto address="moi@example.com" subject="Hello to you!"}
<a href="mailto:moi@example.com?subject=Hello%20to%20you%21" >me@example.com</a>

{mailto address="moi@example.com" cc="toi@example.com,eux@example.com"}
<a href="mailto:moi@example.com?cc=toi@example.com%2Ceux@example.com" >moi@example.com</a>

{mailto address="moi@example.com" extra='class="email"'}
<a href="mailto:moi@example.com" class="email">moi@example.com</a>

{mailto address="moi@example.com" encode="javascript_charcode"}
<script type="text/javascript" language="javascript">
<!--
{document.write(String.fromCharCode(60,97, ... coupé ....60,47,97,62))}
//-->
</script>
```

Voir aussi `escape`, `{textformat}` et le camouflage des adresses E-mail.

{math}

`{math}` autorise les designers de templates à effectuer des opérations dans le template.

- Toute valeur numérique peut être utilisée dans une opération, et le résultat sera affiché à la place des balises "equation".
- Les variables utilisées dans l'opération sont passées en tant que paramètre, et peuvent être des variables de templates ou des valeurs statiques.
- +, -, /, *, abs, ceil, cos, exp, floor, log, log10, max, min, pi, pow, rand, round, sin, sqrt, srans et tan sont tous des opérateurs valides. Voir la documentation PHP pour plus d'informations sur ces fonctions mathématiques [http://php.net/eval].
- Si vous spécifiez l'attribut *assign*, la sortie de la fonction `{math}` sera assignée à la variable donnée plutôt que d'être directement affichée.

Note technique: `{math}` est une fonction coûteuse en terme de performances, du fait qu'elle utilise la fonction PHP `eval()` [http://php.net/eval]. Effectuer les calculs dans votre code PHP est beaucoup plus efficient, donc, chaque fois que possible, effectuez vos calculs directement dans PHP et assignez le résultat au template. Evitez coût que coût les appels répétitifs à la fonction `{math}`, comme on pourait le faire une une boucle `{section}`.

Nom attribut	Type	Requis	Défaut	Description
equation	chaîne de caractères	Oui	*n/a*	L'opération à éxécuter
format	chaîne de caractères	Non	*n/a*	Le format du résultat (sprintf)
var	numeric	Oui	*n/a*	Les variables de l'opération
assign	chaîne de caractères	Non	*n/a*	Variable de template dans laquelle la sortie sera assignée
[var ...]	numeric	Oui	*n/a*	Valeurs des variables de l'opération

Exemple 8.24. {math}

Exemple a :

```
{* $height=4, $width=5 *}

{math equation="x + y" x=$height y=$width}
```

L'exemple ci-dessus affichera :

```
9
```

Exemple b :

```
{* $row_height = 10, $row_width = 20, #col_div# = 2, assigned in template *}

{math equation="height * width / division"
      height=$row_height
      width=$row_width
      division=#col_div#}
```

L'exemple ci-dessus affichera :

```
100
```

Exemple c :

```
{* vous pouvez utiliser des parenthèses *}

{math equation="(( x + y ) / z )" x=2 y=10 z=2}
```

L'exemple ci-dessus affichera :

```
6
```

Exemple d :

```
{* vous pouvez définir un format sprintf pour l'affichage *}
{math equation="x + y" x=4.4444 y=5.0000 format="%.2f"}
```

L'exemple ci-dessus affichera :

```
9.44
```

{popup}

{popup} est utilisé pour créer une fenêtre popup javascript. {popup_init} DOIT être appelé en premier pour que cela fonctionne.

Nom attribut	Type	Requis	Defaut	Description
text	chaîne de caractères	Oui	*n/a*	Le texte/code html à afficher dans la popup
trigger	chaîne de caractères	Non	*onMouseOver*	L'évènement utilisé pour rendre la popup active, onMouseOver ou onClick.
sticky	booléen	Non	*FALSE*	Rends la popup active jusqu'a ce qu'elle soit explicitement fermée.
caption	chaîne de caractères	Non	*n/a*	Défini le libellé du titre
fgcolor	chaîne de caractères	Non	*n/a*	Couleur interne de la popup
bgcolor	chaîne de caractères	Non	*n/a*	Couleur de la bordure de la popup
textcolor	chaîne de caractères	Non	*n/a*	Couleur du texte à l'intérieur de la popup
capcolor	chaîne de caractères	Non	*n/a*	Couleur du libellé de la popup
closecolor	chaîne de caractères	Non	*n/a*	Couleur du texte de fermeture
textfont	chaîne de caractères	Non	*n/a*	La police à utiliser dans le texte principal
captionfont	chaîne de caractères	Non	*n/a*	La police à utiliser dans le libellé
closefont	chaîne de caractères	Non	*n/a*	La police pour le texte de fermeture
textsize	chaîne de caractères	Non	*n/a*	Taille de la police texte prinicpal
captionsize	chaîne de caractères	Non	*n/a*	Taille de la police du libellé
closesize	chaîne de caractères	Non	*n/a*	Taille de la police du bouton "fermer"
width	entier	Non	*n/a*	Longeur de la popup

Nom attribut	Type	Requis	Defaut	Description
height	entier	Non	*n/a*	Hauteur de la popup
left	booléen	Non	*FALSE*	La popup va à gauche de la souris
right	booléen	Non	*FALSE*	La popup va à droite de la souris
center	booléen	Non	*FALSE*	La popup est centrée par rapport à la position de la souris
above	booléen	Non	*FALSE*	la popup est au dessus de la souris. NOTE: possible uniquement si la hauteur est définie
below	booléen	Non	*FALSE*	La popup apparait en dessous de la souris
border	entier	Non	*n/a*	Rends la bordure de la popup plus épaisse ou plus fine
offsetx	entier	Non	*n/a*	A quelle distance du curseur la popup apparaitra horizontalement.
offsety	entier	Non	*n/a*	A quelle distance du curseur la popup apparaitra verticalement.
fgbackground	url vers l'image	Non	*n/a*	Une image à utiliser à la place de la couleur de fonds dans la popup
bgbackground	url vers l'image	Non	*n/a*	Image à utiliser à la place de la bordure de la popup. NOTE: vous veillerez à définir bgcolor à "" ou la couleur apparaitra de même. NOTE: Lorsque vous avez un lien de fermeture, Netscape effectuera un nouveau rendu des cellules du tableau, affichant mal les éléments
closetext	chaîne de caractères	Non	*n/a*	Définit le texte de fermeture par autre chose que "Close"
noclose	booléen	Non	*n/a*	N'affiche pas le bouton "Close" pour les fenêtres "collantes".
status	chaîne de caractères	Non	*n/a*	Défini le texte de la barre de statut du navigateur
autostatus	booléen	Non	*n/a*	Défini le texte de la barre de statut au

Nom attribut	Type	Requis	Defaut	Description
				contenu de la popup. NOTE: Ecrase l'attribut status.
autostatuscap	chaîne de caractères	Non	*n/a*	Défini le texte de la barre de statut au libellé de la popup. NOTE: Ecrase l'attribut status.
inarray	entier	Non	*n/a*	Indique à overLib de lire le texte à cet index dans le tableau ol_text, situé dans overlib.js. Ce paramètre peut être utilisé à la place de text.
caparray	entier	Non	*n/a*	Indique à overlib de lire le libellé depuis le tableau ol_caps
capicon	url	Non	*n/a*	Affiche l'image spécifiée avant le libellé de la popup
snapx	entier	Non	*n/a*	Aligne la popup sur une grille horizontale
snapy	entier	Non	*n/a*	Aligne la popup sur une grille verticale
fixx	entier	Non	*n/a*	Vérrouille la popup à une position horizontale. Note: remplace les autres paramètres de position horizontale
fixy	entier	Non	*n/a*	Vérouille la popup à une position verticale Note: remplace les autres paramètres de position verticale
background	url	Non	*n/a*	Défini l'image à utiliser plutôt que le tableau de fond
padx	entier, entier	Non	*n/a*	Écarte l'image de fond du reste des éléments avec un espace horizontal, pour le positionnement du texte. Note: c'est un attribut à deux paramètres.
pady	entier, entier	Non	*n/a*	Écarte l'image de fond du reste des éléments avec un espace vertical, pour le positionnement du texte. Note: c'est un attribut à deux

Nom attribut	Type	Requis	Defaut	Description
				paramètres.
fullhtml	booléen	Non	*n/a*	Vous autorise à placer du code html en tant que contenu de la popup. Le code html est attendu dans l'attribut text.
frame	chaîne de caractères	Non	*n/a*	Contrôle la popup dans un cadre différent. Voir la documentation d'overlib pour plus de détails sur cette fonction.
function	chaîne de caractères	Non	*n/a*	Appelle la fonction javascript spécifiée et prends sa valeur de retour comme texte devant être affiché dans la popup.
delay	entier	Non	*n/a*	La popup se comporte comme une infobulle. Elle disparaitra au bout d'un certain délai, en millisecondes.
hauto	booléen	Non	*n/a*	Détermine automatiquement si la popup doit être à gauche ou à droite de la souris
vauto	booléen	Non	*n/a*	Détermine automatiquement si la popup doit être au-dessus ou au-dessous de la souris

Exemple 8.25. {popup}

```
{* popup_init doit être appelé en haut de votre page *}
{popup_init src='/javascripts/overlib.js'}

{* création d'un lien avec une popup qui apparait sur l'évènement onMouseOver *}
<A href="mypage.html" {popup text='Ce lien vous amène sur ma page!'}>mypage</A>

{* vous pouvez utiliser du html, des liens, etc. dans vos popup *}
<a href="mypage.html" {popup sticky=true caption='Contenu de la page'
text="<ul><li>links</li><li>pages</li><li>images</li></ul>"
snapx=10 snapy=10 trigger='onClick'}>ma page</a>

{* un popup via une cellule du tableau *}
<tr><td {popup caption='Détails de cette partie' text=$part_long_description}>{$part_number}</td></tr>
```

Il y a également un autre bon exemple sur la page de la documentation de {capture}.

Voir aussi {popup_init} et overLib [http://www.bosrup.com/web/overlib/].

{popup_init}

{popup} est une intégration de overLib [http://www.bosrup.com/web/overlib/], une librairie capable de réaliser des fenêtres surgissantes (nous parlerons de "popup"). Ce type de fenêtre est utilisé pour apporter des informations contextuelles, comme des infobulles d'aides ou astuces.

- `{popup_init}` doit être appelé une *seule fois*, de préférence dans la balise `<head>`, dans toutes les pages si vous comptez utiliser la fonction `{popup}`.
- Le chemin est relatif au script exécuté ou un chemin complet (i.e. non relatif au template).
- overLib [http://www.bosrup.com/web/overlib/] a été écrit par Erik Bosrup, et le site de l'auteur/le téléchargement est disponible à l'adresse sur http://www.bosrup.com/web/overlib/.

Exemple 8.26. {popup_init}

```
<head>
{* popup_init doit être appelé une fois en début de page. *}
{popup_init src='/javascripts/overlib.js'}

{* exemple avec une url complète *}
{popup_init src='http://myserver.org/my_js_libs/overlib/overlib.js'}
</head>

// le premier exemple affichera
<head>
 <div id="overDiv" style="position:absolute; visibility:hidden; z-index:1000;"></div>
 <script type="text/javascript" language="JavaScript" src="javascripts/overlib/overlib.js"></script>
</head>
```

> **Validation XHTML:** `{popup_init}` ne valide pas en validation stricte et vous devriez obtenir l'erreur : `document type does not allow element "div" here;` (i.e. une balise `<div>` dans la balise `<head>`). Ceci signifie que vous devez inclure les balises `<script>` et `<div>` manuellement.

{textformat}

`{textformat}` est une fonction de bloc utilisée pour formater du texte. Elle nettoie la chaîne de ses espaces et caractères spéciaux, puis formate les paragraphes en ajustant ces derniers à une certaine limite, puis en indentant les lignes.

Vous pouvez soit utiliser un style prédéfini, soit définir explicitement chaque attribut. Actuellement, seul le style prédéfini «email» est disponible.

Nom attribut	Type	Requis	Defaut	Description
style	chaîne de caractères	Non	*n/a*	Style prédéfini
indent	number	Non	*0*	Taille de l'indentation pour chaque ligne
indent_first	number	Non	*0*	Taille de l'indentation de la première ligne
indent_char	chaîne de caractères	Non	*(single space)*	Le caractère (ou la chaîne) à utiliser pour

Nom attribut	Type	Requis	Defaut	Description
				indenter
wrap	number	Non	*80*	À combien de caractères doit on ajuster chaque ligne
wrap_char	chaîne de caractères	Non	*\n*	Le caractère (ou chaîne de caractères) avec lequel terminer les lignes
wrap_cut	boolean	Non	*FALSE*	Si true, wrap réduira les lignes au caractère exact au lieu d'ajuster à la fin d'un mot
assign	chaîne de caractères	Non	*n/a*	Le nom de la variable PHP dans laquelle la sortie sera assignée

Exemple 8.27. {textformat}

```
{textformat wrap=40}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}
```

L'exemple ci-dessus affichera :

```
This is foo. This is foo. This is foo.
This is foo. This is foo. This is foo.

This is bar.

bar foo bar foo foo. bar foo bar foo
foo. bar foo bar foo foo. bar foo bar
foo foo. bar foo bar foo foo. bar foo
bar foo foo. bar foo bar foo foo.
```

```
{textformat wrap=40 indent=4}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
```

```
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}
```

L'exemple ci-dessus affichera :

```
This is foo. This is foo. This is
    foo. This is foo. This is foo. This
    is foo.

    This is bar.

    bar foo bar foo foo. bar foo bar foo
    foo. bar foo bar foo foo. bar foo
    bar foo foo. bar foo bar foo foo.
    bar foo bar foo foo. bar foo bar
    foo foo.
```

```
{textformat wrap=40 indent=4 indent_first=4}

This is foo.
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}
```

L'exemple ci-dessus affichera :

```
This is foo. This is foo. This
    is foo. This is foo. This is foo.
    This is foo.

    This is bar.

    bar foo bar foo foo. bar foo bar
    foo foo. bar foo bar foo foo. bar
    foo bar foo foo. bar foo bar foo
    foo. bar foo bar foo foo. bar foo
    bar foo foo.
```

```
{textformat style="email"}

This is foo.
```

```
This is foo.
This is foo.
This is foo.
This is foo.
This is foo.

This is bar.

bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.
bar foo bar foo     foo.

{/textformat}
```

L'exemple ci-dessus affichera :

```
This is foo. This is foo. This is foo. This is foo. This is foo. This is
foo.

This is bar.

bar foo bar foo foo. bar foo bar foo foo. bar foo bar foo foo. bar foo
bar foo foo. bar foo bar foo foo. bar foo bar foo foo. bar foo bar foo
foo.
```

Voir aussi {strip} et {wordwrap}.

Chapitre 9. Fichiers de configuration

Les fichiers de configuration sont un moyen interressant pour gérer des variables depuis un seul et même fichier. L'exemple le plus courant étant le schéma de couleurs du template. Normalement, pour changer le schéma de couleur d'une application, vous devriez aller dans chaque template et changer la couleur des éléments (ou les classes css). Avec un fichier de configuration, il vous est possible de conserver la couleur dans un seul endroit, puis de la mettre à jour une seule fois.

Exemple 9.1. Exemple de fichier de configuration

```
# variables globales
titrePage = "Menu principal"
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

[client]
titrePage = "Infos client"

[Login]
titrePage = "Login"
focus = "utilisateur"
Intro = """Une valeur qui tient sur
           plusieur lignes. Vous devez la placer
           entre trois guillemets."""

# hidden section
[.Database]
host=mon.example.com
db=ADDRESSBOOK
user=php-user
pass=foobar
```

Les valeurs des variables de fichiers de configuration peuvent être entre guillemets, sans que celà soit nécessaire. Si vous vouler utiliser des valeurs sur plusieurs lignes, vous devrez les entourer de triples guillemets ("""). Vous pouvez insérer des commentaires dans les fichiers de configuration en utilisant une syntaxe quelquonque, non valide. Nous recommandons l'utilisation de # (dièse) en début de ligne.

Cet exemple de fichier de configuration contient deux sections. Les noms des sections sont entourés de crochets []. Les noms de section peuvent être des chaînes, ne contenant aucun des symboles `[` et `]`. Dans notre exemple, les 4 variables du début sont des variables dites globales, qui ne sont pas contenue dans une section. Ces variables sont toujours chargées depuis le fichier de configuration. Si une section est chargée, alors toutes les variables de cette section ainsi que les variables globales sont chargées. Si une variable existe à la fois en tant que globale et à la fois en tant que variable de section, la variable de section est prioritaire. Si vous appelez deux variables dans une même section de la même façon, la dernière déclarée prime. (voir *$config_overwrite*)

Les fichiers de configuration sont chargés dans le template grâce aux fonctions `{config_load}` (voir aussi `config_load()`).

Vous pouvez masquer des variables ou des sections entières en préfixant le nom de la variable ou le nom de la section avec une virgule. Ce procédé est utile si votre application récupère ses données depuis plusieurs fichiers de configuration et récupère des données sensibles dont vos templates n'ont pas besoin. Si des tiers éditent des templates, vous êtes sûr que ces derniers n'accèderont pas à ces données de configuration en les chargeant depuis le template.

Voir aussi `{config_load}`, *$config_overwrite*, `get_config_vars()`, `clear_config()` et `config_load()`.

Chapitre 10. Console de débogage

Il existe une console de débogage dans Smarty. La console vous indique toutes les templates incluses, les variables assignées et chargées depuis un fichier de configuration pour le template courant. Un template appelé `debug.tpl` est inclus dans la distribution de Smarty qui contrôle le formattage de la console. Définissez $debugging à `TRUE` dans Smarty et, si besoin, vous pouvez définir *$debug_tpl* de façon à ce que ce dernier contienne le chemin du template à utiliser(dans `SMARTY_DIR` par defaut). Lorsque vous chargez la page, une console javascript est censée surgir et vous donner les noms de toutes les variables inclues et assignées dans votre page courante. Pour voir toutes les variables d'un template particulier, voir la fonction `{debug}`. Pour désactiver la console de débogage, définissez *$debugging* à `FALSE`. Vous pouvez également temporairement activer le débogage en indiquant `SMARTY_DEBUG` dans l'url si tant est que l'option *$debugging_ctrl* soit activée.

> **Note technique:** La console de débogage ne fonctionne pas si vous utilisez l'API `fetch()`, mais seulement lorsque vous utilisez `display()`. C'est en effet un jeu d'instructions javascripts à la fin du template qui déclenchent l'ouverture de la fenêtre. Si vous n'aimez pas javascript, vous pouvez modifier `debug.tpl` pour formater les données de la façon qui vous conviendra le mieux. Les données de débogage ne sont pas mises en cache et les informations de debug.tpl ne sont pas incluses dans la sortie de la console de débogage.
>
> **NOTE:** Le temps de chargement des templates et des fichiers de configuration sont indiqués en secondes.

Voir aussi troubleshooting, *$error_reporting* et `trigger_error()`.

Partie III. Smarty pour les programmeurs

Table des matières

Chapitre 11. Constantes

Table des matières

SMARTY_DIR

Il doit s'agir du **chemin complet** du répertoire où se trouvent les fichiers classes de Smarty. S'il n'est pas défini dans votre script, Smarty essaiera alors d'en déterminer automatiquement la valeur. S'il est défini, le chemin **doit se terminer par un slash**.

Exemple 11.1. SMARTY_DIR

```
<?php
// définit le chemin du répertoire de Smarty sur un système *nix
define('SMARTY_DIR', '/usr/local/lib/php/Smarty-v.e.r/libs/');

// définit le chemin du répertoire de Smarty sur un système Windows
define('SMARTY_DIR', 'c:/webroot/libs/Smarty-v.e.r/libs/');

// inclut la classe Smarty. Notez le 'S' en majuscule
require_once(SMARTY_DIR . 'Smarty.class.php');
?>
```

Voir aussi *$smarty.const* et *$php_handling constants*.

SMARTY_CORE_DIR

Il doit s'agir du *chemin complet* du répertoire où se trouvent les fichiers internes de Smarty. S'il n'est pas défini, Smarty placera comme valeur par défaut la valeur de la constante précédente `SMARTY_DIR`. S'il est défini, le chemin doit se terminer par un slash. Utilisez cette constante lorsque vous incluez manuellement n'importe quel fichier core.*.

Exemple 11.2. SMARTY_CORE_DIR

```
<?php

// chargement de core.get_microtime.php
require_once(SMARTY_CORE_DIR . 'core.get_microtime.php');

?>
```

Voir aussi *$smarty.const*.

Chapitre 12. Variables

Table des matières

$template_dir

C'est le nom par défaut du répertoire des templates. Si vous ne spécifiez aucun chemin lors de l'utilisation de templates, Smarty les cherchera à cet emplacement. Par défaut, il s'agit de `./templates`, ce qui signifie qu'il va chercher le répertoire `templates/` dans le répertoire où se trouve le script PHP en cours d'exécution.

Note technique: Il n'est pas conseillé de mettre ce répertoire dans l'arborescence Web.

$compile_dir

C'est le nom du répertoire où se trouvent les templates compilés. Par défaut, il s'agit de `./templates_c`, ce qui signifie que

Smarty va chercher ce répertoire dans le même répertoire que le script PHP en cours d'exécution. **Ce dossier doit être accessible en écriture par le serveur web.** (Voir l'installation pour plus d'informations).

> **Note technique:** Ce réglage doit être soit un chemin absolu, soit un chemin relatif. include_path n'est pas utilisé pour écrire des fichiers.

> **Note technique:** Il n'est pas conseillé de mettre ce répertoire sous la racine de l'arborescence Web.

Voir aussi *$compile_id* et *$use_sub_dirs*.

$config_dir

Il s'agit du répertoire utilisé pour stocker les fichiers de configuration utilisés dans les templates. La valeur par défaut est `./configs`, ce qui signifie que Smarty va chercher ce répertoire dans le même répertoire que le script PHP qui s'exécute.

> **Note technique:** Il n'est pas conseillé de mettre ce répertoire sous la racine de l'arborescence Web.

$plugins_dir

C'est le répertoire (ou les répertoires) dans lequel Smarty ira chercher les plugins dont il a besoin. La valeur par défaut est `plugins/` sous le répertoire `SMARTY_DIR`. Si vous donnez un chemin relatif, Smarty regardera d'abord relativement au `SMARTY_DIR`, puis relativement au répertoire de travail courant, puis relativement à chaque entrée de votre répertoire d'inclusion PHP. Si *$plugins_dir* est un tableau de répertoires, Smarty cherchera les plugins dans chaque répertoire de plugins, **dans l'ordre donné**.

> **Note technique:** Pour des raisons de performances, ne réglez pas votre *$plugins_dir* pour qu'il utilise votre include_path PHP. Utilisez un chemin absolu ou un chemin relatif a `SMARTY_DIR` ou au répertoire de travail courant.

Exemple 12.1. Ajout d'un dossier local de plugins

```
<?php

$smarty->plugins_dir[] = 'includes/my_smarty_plugins';

?>
```

Exemple 12.2. Plusieurs $plugins_dir

```
<?php

$smarty->plugins_dir = array(
            'plugins', // the default under SMARTY_DIR
            '/path/to/shared/plugins',
            '../../includes/my/plugins'
           );

?>
```

$debugging

Celà active la console de débogage. La console est une fenêtre javascript qui vous informe des templates inclus et des variables assignées depuis PHP et des variables des fichiers de configuration pour le script courant. Il ne montre pas les

variables assignées dans un template avec {assign}.

Voir aussi *$debugging_ctrl* sur la façon d'activer le débogage depuis l'url.

Voir aussi {debug}, *$debug_tpl* et *$debugging_ctrl*.

$debug_tpl

C'est le nom du fichier template utilisé pour la console de débogage. Par défaut debug.tpl, il se situe dans SMARTY_DIR.

Voir aussi *$debugging* et la console de débogage.

$debugging_ctrl

Cela permet d'avoir diffÃ©rents moyens pour activer le dÃ©bogage. NONE signifie qu'aucune mÃ©thode alternative n'est autorisÃ©e. URL signifie que si SMARTY_DEBUG se trouve dans QUERY_STRING, le dÃ©bogage est activÃ© Ã l'invocation du script. Si $debugging est Ã TRUE, cette valeur est sans effet.

Exemple 12.3. $debugging_ctrl sur localhost

```
<?php
// affiche la console de dÃ©bogage uniquement sur localhost ie
// http://localhost/script.php?foo=bar&SMARTY_DEBUG
$smarty->debugging = false; // the default
$smarty->debugging_ctrl = ($_SERVER['SERVER_NAME'] == 'localhost') ? 'URL' : 'NONE';
?>
```

Voir aussi la console de dÃ©bogage et *$debugging*.

$autoload_filters

Si vous désirez charger des filtres a chaque invocation de templates, vous pouvez le spécifier en utilisant cette variable. Les types de filtres et les valeurs sont des tableaux comportant le nom des filtres.

```
$smarty->autoload_filters = array('pre' => array('trim', 'stamp'),
                                  'output' => array('convert'));
```

Voir aussi register_outputfilter(), register_prefilter(), register_postfilter() et load_filter().

$compile_check

A chaque invocation de l'application PHP, Smarty fait un test pour voir si le template courant a été modifié (date de dernière modification différente) depuis sa dernière compilation. S'il a changé, le template est recompilé. Si le template n'a pas encore été compilé, il le sera quelque soit la valeur de ce réglage. Par défaut cette valeur est à TRUE.

Quand une application est mise en production (les templates ne changent plus), cette vérification n'est pas nécessaire. Assurez-vous de mettre *$compile_check* à FALSE pour des performances maximales. Notez que si vous mettez ce paramètre à FALSE et qu'un template est modifié, vous ne verrez *pas* le changement car le template ne sera *pas* recompilé. Si le processus de cache est activé et que *$compile_check* l'est aussi, alors les fichiers du cache seront regénérés si un template concerné ou un fichier de configuration concerné est modifié. Voir aussi *$force_compile* ou clear_compiled_tpl().

$force_compile

Celà oblige Smarty à (re)compiler les templates à chaque invocation. Ce réglage supplante *$compile_check*. Par défaut, il vaut `FALSE`. Ceci est commode pour le développement et le débogage mais ne devrait jamais être utilisé dans un environnment de production. Si le système de cache est actif, les fichiers du cache seront regénérés à chaque appel.

$caching

Ce paramètre demande à Smarty de mettre ou non en cache la sortie des templates. Par défaut, ce réglage est à 0 (désactivé). Si vos templates génèrent du contenu redondant, il est conseillé d'activer le cache. Celà permettra un gain de performance conséquent.

Vous pouvez aussi avoir de nombreux fichiers de cache pour un même template.

- Une valeur de 1 ou 2 active le cache.
- 1 indique a Smarty d'utiliser la variable *$cache_lifetime* pour déterminer si le fichier de cache a expiré.
- Une valeur de 2 indique à Smarty d'utiliser la valeur *$cache_lifetime* spécifiée à la génération du cache. Ainsi vous pouvez régler la durée de vie d'un fichier de cache avant de récupérer le template pour avoir un certain contrôle quand ce fichier en particulier expire. Voir aussi `is_cached()`.
- Si *$compile_check* est actif, le contenu du cache sera regénéré si un des templates ou un des fichiers de configuration qui fait partie de ce fichier de cache a été modifié.
- Si *$force_compile* est actif, le contenu du cache est toujours regénéré.

Voir aussi *$cache_dir*, *$cache_lifetime*, *$cache_handler_func*, *$cache_modified_check*, `is_cached()` et la section sur le cache.

$cache_dir

Il s'agit du nom du répertoire où les caches des templates sont stockés. Par défaut il s'agit de `./cache`, ce qui signifie que Smarty va chercher ce répertoire dans le même répertoire que le script PHP en cours d'exécution. **Ce dossier doit être accessible en écriture par le serveur web** (Voir l'installation pour plus d'informations). Vous pouvez aussi utiliser votre propre fonction de gestion de cache personnalisé pour contrôler les fichiers de cache, qui ignorera cette configuration. Voir aussi *$use_sub_dirs*.

> **Note technique:** Ce réglage doit être soit un chemin absolu, soit un chemin relatif. include_path n'a aucune influence lors de l'écriture des fichiers.
>
> **Note technique:** Il n'est pas conseillé de mettre ce répertoire dans l'arborescence Web.

Voir aussi *$caching*, *$use_sub_dirs*, *$cache_lifetime*, *$cache_handler_func*, *$cache_modified_check* et la section sur le cache.

$cache_lifetime

Il s'agit de la durée en secondes pendant laquelle un cache de template est valide. Une fois cette durée dépassée, le cache est regénéré.

- *$caching* doit être activé (soit 1 ou 2) pour que *$cache_lifetime* ait une quelconque utilité.
- Avec une valeur de -1, le cache n'expire jamais.

- Avec une valeur de 0, le cache est toujours regénéré (utile a des fins de tests seulement. Une meilleure façon de désactiver le cache est de mettre *$caching* = 0).
- Si vous souhaitez donner a certains templates leur propre durée de vie en cache, vous pouvez le faire en réglant *$caching* à 2, puis *$cache_lifetime* à une unique valeur juste avant d'appeler `display()` ou `fetch()`.

Si *$force_compile* est activé, les fichiers du cache seront regénérés a chaque fois, désactivant ainsi le cache. Vous pouvez effacer tous les fichiers du cache avec la function `clear_all_cache()` ou de façon individuelle (ou groupée) avec la fonction `clear_cache()`.

$cache_handler_func

Vous pouvez utiliser votre propre fonction de gestion du cache plutôt que d'utiliser celle livrée avec Smarty (*$cache_dir*). Référez-vous à la section sur la fonction de gestion de cache personnalisée pour plus de détails.

$cache_modified_check

Si cette variable est à `TRUE`, Smarty respectera l'en-tête If-Modified-Since envoyé par le client. Si la date de dernière modification du fichier de cache n'a pas changé depuis la dernière visite, alors un en-tête `'304: Not Modified'` sera envoyé à la place du contenu. Celà ne fonctionne qu'avec du contenu mis en cache hors de la balise `{insert}`.

Voir aussi *$caching*, *$cache_lifetime*, *$cache_handler_func* et la section sur le cache.

$config_overwrite

Si cette variable est à `TRUE` (par défaut), les variables lues dans les fichiers de configuration peuvent s'écraser entre elles. Sinon les variables seront mises dans un tableau. Très utile si vous voulez stocker des tableaux de données dans des fichiers de configuration, listez simplement chaque élément plusieurs fois.

Exemple 12.4. Tableau de variables de configuration

Cet exemple utilise `{cycle}` pour afficher un tableau dont les lignes sont alternativement rouge/verte/bleu avec *$config_overwrite* = `FALSE`.

Le fichier de configuration

```
# couleur des lignes
rowColors = #FF0000
rowColors = #00FF00
rowColors = #0000FF
```

Le template avec une boucle `{section}`.

```
<table>
 {section name=r loop=$rows}
 <tr bgcolor="{cycle values=#rowColors#}">
  <td> ....etc.... </td>
 </tr>
 {/section}
</table>
```

Voir aussi `{config_load}`, `get_config_vars()`, `clear_config()`, `config_load()` et les fichiers de configuration.

$config_booleanize

Si cette variable est à TRUE, les valeurs on/true/yes et off/false/no dans les fichiers de configuration sont automitiquement converties en booléen. De cette façon vous pouvez utiliser ces valeurs dans le template de la façon suivante : {if #foobar#}...{/if}. Si foobar est à on, true ou yes, l'instruction {if} sera exécutée. TRUE par défaut.

$config_read_hidden

Si cette variable est à TRUE, les sections cachées (dont les noms commencent par un point) dans les fichiers de configuration peuvent être lues depuis les templates. On laisse habituellement celà à FALSE, de cette façon vous pouvez stocker des données sensibles dans les fichiers de configuration, par exemple des paramètres de base de données, sans vous soucier de la façon dont les templates les chargent. Mise à FALSE par défaut.

$config_fix_newlines

Si cette variable est mise à TRUE, les caractéres de nouvelles lignes mac et dos ('\r' et '\r\n') sont convertis en '\n' quand ils sont analysés. TRUE par défaut.

$default_template_handler_func

Cette fonction est appelée quand un template ne peut pas être obtenu avec sa ressource.

$php_handling

Indique à Smarty comment interpréter le code PHP intégré dans les templates. Il y a quatre valeurs possibles, par défaut SMARTY_PHP_PASSTHRU. Notez que celà n'affecte PAS le code PHP entouré des balises {php}{/php} dans le template.

- SMARTY_PHP_PASSTHRU - Smarty écrit les balises telles quelles.
- SMARTY_PHP_QUOTE - Smarty transforme les balises en entités HTML.
- SMARTY_PHP_REMOVE - Smarty supprime les balises des templates.
- SMARTY_PHP_ALLOW - Smarty exécute les balises comme du code PHP.

> **NOTE:** Intégrer du code PHP dans les templates est vivement déconseillé. Préférez les fonctions utilisateurs ou les modificateurs de variables.

$security

Cette variable vaut FALSE par défaut. La sécurité est de rigueur quand vous n'êtes pas complétement sûr des personnes qui éditent les templates (par ftp par exemple) et que vous voulez réduire le risque que la sécurité du système soit compromise par le langage de template. Activer cette option de sécurité applique les régles suivantes au langage de template, à moins que *$security_settings* ne spécifie le contraire :

- Si *$php_handling* est réglée à SMARTY_PHP_ALLOW, cela est implicitement changé à SMARTY_PHP_PASSTHRU.
- Les fonctions PHP ne sont pas autorisées dans les instructions {if}, à part celles déclarées dans *$security_settings*.
- Les templates ne peuvent être inclus que depuis des répertoires listés dans le tableau *$secure_dir*.

- Les fichiers locaux ne peuvent être récupérés que depuis les répertoires listés dans le tableau *$secure_dir* en utilisant `{fetch}`.
- Les balises `{php}{/php}` ne sont pas autorisées.
- Les fonctions PHP ne sont pas autorisées en tant modificateurs, à part celles spécifiées dans *$security_settings*.

$secure_dir

Il s'agit d'un tableau contenant tous les fichiers et répertoires locaux qui sont considérés comme sécurisés. `{include}` et `{fetch}` l'utilisent quand la sécurité est activée.

Exemple 12.5. Exemple avec $secure_dir

```
<?php
$secure_dirs[] = '/path/to/site/root/templates/';
$secure_dirs[] = '/path/to/includes/';
$smarty->secure_dir = $secure_dirs;
?>
```

Voir aussi la configuration pour la sécuritéet *$trusted_dir*.

$security_settings

Ces réglages servent à écraser ou spécifier les paramètres de sécurité quand celle-ci est activée. Les réglages possibles sont les suivants :

- `PHP_HANDLING` - booléen. Si `TRUE`, le réglage *$php_handling* n'est pas vérifié.
- `IF_FUNCS` - Le tableau des noms de fonctions PHP autorisées dans les intructions `{if}`.
- `INCLUDE_ANY` - booléen. Si `TRUE`, les templates peuvent être inclus de n'importe où, quelque soit le contenu de *$secure_dir*.
- `PHP_TAGS` - booléen. Si `TRUE`, les balises `{php}{/php}` sont autorisées dans les templates.
- `MODIFIER_FUNCS` - Le tableau des noms de fonctions autorisées à être utilisées comme modificateurs de variables.
- `ALLOW_CONSTANTS` - booléen. Si l'accès aux constantes via la syntaxe {$smarty.const.name} est autorisé ou non.

$trusted_dir

$trusted_dir n'est utilisée lorsque *$security* est activée. C'est un tableau de tous les répertoires qui peuvent être considérés comme svrs. Les répertoires svrs sont ceux qui contiennent des scripts PHP qui sont exécutés directement depuis les templates avec `{include_php}`.

$left_delimiter

Il s'agit du délimiteur gauche utilisé par le moteur de templates. La valeur par défaut est `{`.

Voir aussi *$right_delimiter* et l'analyse d'échapement Smarty.

$right_delimiter

Il s'agit du délimiteur droit utilisé par le moteur de templates. La valeur par défaut est `}`.

Voir aussi *$left_delimiter* et l'analyse d'échappement Smarty.

$compiler_class

Spécifie le nom de la classe du compilateur qui va être utilisée pour compiler les templates. Le compilateur par défaut est 'Smarty_Compiler'. Réservé aux utilisateurs avancés.

$request_vars_order

L'ordre dans lequel les variables de requêtes sont enregistrées, identique a variables_order dans php.ini.

Voir aussi *$smarty.request* et *$request_use_auto_globals*.

$request_use_auto_globals

Spécifie si Smarty doit utiliser les variables PHP `$HTTP_*_VARS[]` ($request_use_auto_globals=`FALSE` qui est la valeur par défaut) ou `$_*[]` ($request_use_auto_globals=`TRUE`). Cela affecte les templates qui utilisent `{$smarty.request.*}`, `{$smarty.get.*}` etc..

> **Attention:** Si vous configurez `$request_use_auto_globals to true` à `TRUE`, *$request_vars_order* n'a plus d'effets et la valeur de la directive de configuration `gpc_order` de PHP est utilisée.

$error_reporting

Lorsque cette valeur est configurée à une valeur non nulle, sa valeur est utilisée comme le `error_reporting` [http://php.net/error_reporting]-level de PHP à l'intérieur de `display()` et `fetch()`. Lorsque le déboguage est ignoré, cette valeur est ignorée et error-level est non-modifié.

Voir aussi `trigger_error()`, le débogage et Troubleshooting.

$compile_id

Identifiant persistant du compilateur. On peut passer le même *$compile_id* a chaque appel de fonction mais une alternative consiste à régler ce *$compile_id*, qui sera utilisé implicitement.

Avec un *$compile_id*, vous pouvez contourner la limitation qui fait que vous ne pouvez pas utiliser le même *$compile_dir* pour différents *$template_dirs*. Si vous définissez un *$compile_id* distinct pour chaque *$template_dir*, alors Smarty indique aux templates compilés à part par leur *$compile_id*.

Si vous avez par exemple un pré-filtre qui traduit vos templates au moment de la compilation, alors, vous devriez utiliser le langage courant comme *$compile_id* et vous devriez obtenir un jeu de templates compilés pour chaque langage que vous utiliserez.

Un autre exemple serait d'utiliser le même dossier de compilation à travers de multiples domaines / vhosts.

Exemple 12.6. $compile_id dans un environement d'hôte virtuel

```
<?php

$smarty->compile_id = $_SERVER['SERVER_NAME'];
$smarty->compile_dir = '/chemin/vers/shared_compile_dir';

?>
```

$use_sub_dirs

Smarty va créer des sous-dossiers dans les dossiers templates_c et cache si la variable *$use_sub_dirs* est défini à `TRUE` (Par défaut, vaut `FALSE`). Dans un environnement où il peut y avoir potentiellement des centaines de milliers de fichiers de créés, ceci peut rendre le système de fichiers plus rapide. D'un autre côté, quelques environnements n'acceptent pas que les processus PHP créent des dossiers, donc, cette variable doit être désactivée par défaut.

Les sous-dossiers sont plus efficaces, utilisez-les donc si vous le pouvez. Théoriquement, vous obtiendrez plus de performance sur un système de fichier contenant 10 dossiers contenant chaque, 100 fichiers plutôt qu'un dossier contenant 1000 fichiers. C'est par exemple le cas avec Solaris 7 (UFS)... avec les systèmes de fichiers récents comme ext3 ou reiserfs, la différence est proche de zéro.

> **Note technique:** `$use_sub_dirs=true` ne fonctionne pas avec safe_mode=On [http:/ / php.net/ features.safe-mode], raison pour laquelle c'est paramétrable et que c'est désactivé par défaut. `$use_sub_dirs=true` sous Windows peut causer des problèmes.Safe_mode est obsolète depuis PHP6.

Voir aussi *$compile_id*, *$cache_dir* et *$compile_dir*.

$default_modifiers

Il s'agit d'un tableau de modificateurs utilisé pour assigner une valeur par défaut a chaque variable dans un template. Par exemple, pour par défaut échapper les caractéres HTML de chaque variable, utilisez `array('escape:"htmlall"')`. Pour rendre une variable indépendante des modificateurs par défaut, passez-lui en paramètre le modificateur `nodefaults` : `{$var|smarty:nodefaults}`.

$default_resource_type

Ceci dit à smarty quel type de ressource utiliser implicitement. La valeur par défaut est `file`, signifiant que `$smarty->display('index.tpl')` et `$smarty->display('file:index.tpl')` sont la même chose. Voyez le chapitre ressource pour plus de détails.

Chapitre 13. Méthodes

Table des matières

append()

append()Ajoute un élément à un tableau assigné

append()

Description

void **append** (mixed var)
void **append** (string varname, mixed var [, bool merge])

Si vous utilisez cette fonction avec une chaîne de caractères, elle est convertie en tableau auquel on ajoute ensuite l'élément. Vous pouvez explicitement passer des paires nom/valeur. Si vous passez le troisième paramètre (optionel) à `TRUE`, la valeur sera fusionnée avec le tableau plutôt que d'être ajoutée.

> **Note technique:** Le paramètre *merge* respecte les clés du tableau, donc, si vous fusionnez deux tableaux indexés numériquement, ils peuvent se recouvrir les uns les autres ou aboutir à des clés non séquentielles. Ceci est diférent de la fonction PHP array_merge() [http://php.net/array_merge] qui élimine des clés numériques et les renumérote.

Exemple 13.1. Exemple avec append

```
<?php
// passe des paires nom/valeur
$smarty->append("Nom","Fred");
$smarty->append("Adresse",$address);

$array = array(1 => 'un', 2 => 'deux');
$smarty->append('X', $array);
$array2 = array(3 => 'trois', 4 => 'quatre');
// La ligne suivante ajoute un second élément au tableau X
$smarty->append('X', $array2);

// passe un tableau associatif
$smarty->append(array('Ville' => 'Lincoln','Pays' => 'Nebraska'));
?>
```

Voir aussi `append_by_ref()`, `assign()` et `get_template_vars()`.

append_by_ref()

append_by_ref()Ajoute des valeurs par référence

append_by_ref()

Description

void **append_by_ref** (string varname, mixed var [, bool merge])

Utilisée pour ajouter des valeurs à un template par référence plutôt que par copie. Si vous ajoutez une variable par référence puis changez sa valeur, le changement est aussi répercuté sur la valeur assignée. Pour les objets, `append_by_ref()` ne fait pas de copie en mémoire de l'objet assigné. Voir la documentation PHP pour plus d'informations sur les références de variable. Si vous passez le troisième paramètre à `TRUE`, la valeur sera fusionnée avec le tableau courant plutôt que d'être ajoutée.

> **Note technique:** Le paramètre *merge* respecte les clés du tableau, donc, si vous fusionnez deux tableaux indexés numériquement, ils peuvent se recouvrir les uns les autres ou aboutir à des clés non séquentielles. Ceci est diférent de la fonction PHP array_merge() [http://php.net/array_merge] qui élimine des clés numériques et les renumérote.

Exemple 13.2. Exemple avec append_by_ref

```
<?php
// ajoute des paires nom/valeur
$smarty->append_by_ref('Nom',$myname);
$smarty->append_by_ref('Adresse',$address);
?>
```

Voir aussi `append()`, `assign()` et `get_template_vars()`.

assign()

assign()Assigne des valeurs au template

assign()

Description

void **assign** (mixed var)
void **assign** (string varname, mixed var)

Vous pouvez explicitement passer des paires nom/valeur, ou des tableaux associatifs contenant des paires nom/valeur.

Exemple 13.3. Exemple avec assign()

```
<?php
// passe des paires nom/valeur
$smarty->assign("Nom","Fred");
$smarty->assign("Adresse",$address);

// passe un tableau associatif
$smarty->assign(array('Ville' => 'Lincoln','Pays' => 'Nebraska'));

// passe un tableau
$myArray = array('no' => 10, 'label' => 'Peanuts');
$smarty->assign('foo',$myArray);

// Passe une ligne d'une base de données (eg adodb)
$sql = 'select id, name, email from contacts where contact ='.$id;
$smarty->assign('contact', $db->getRow($sql));
?>
```

Accéder à cela dans un template avec

```
{* notez que les variables sont sensibles à la casse, comme en PHP *}
{$Name}
{$Address}
{$city}
{$state}

{$foo.no}, {$foo.label}
{$contact.id}, {$contact.name},{$contact.email}
```

Pour des assignements plus complexes de tableaux, lisez `{foreach}` et `{section}`.

Voir aussi `assign_by_ref()`, `get_template_vars()`, `clear_assign()`, `append()` et `{assign}`.

assign_by_ref()

assign_by_ref()Assigne des valeurs par référence

assign_by_ref()

Description

void **assign_by_ref** (string varname, mixed var)

Utilisée pour assigner des valeurs aux templates par référence plutôt que par copie. Référez-vous au manuel PHP pour une explication plus précise sur les références des variables.

> **Note technique:** Si vous assignez une variable par référence puis changez sa valeur, le changement est aussi répercuté sur la valeur assignée. Pour les objets, `assign_by_ref()` ne fait pas de copie en mémoire de l'objet assigné. Référez-vous au manuel PHP pour une explication plus précise sur les références de variable.

Exemple 13.4. Exemple avec assign_by_ref()

```
<?php
// passe des paires noms/valeurs
$smarty->assign_by_ref("Nom",$myname);
$smarty->assign_by_ref("Adresse",$address);
?>
```

Voir aussi `assign()`, `clear_all_assign()`, `append()`, `{assign}` et `get_template_vars()`.

clear_all_assign()

clear_all_assign()Efface les valeurs de toutes les variables assignées

clear_all_assign()

Description

void **clear_all_assign** (void)

Exemple 13.5. Exemple avec clear_all_assign()

```
<?php
// passe des paires nom/valeur
$smarty->assign('Name', 'Fred');
$smarty->assign('Address', $address);

// affichera
print_r( $smarty->get_template_vars() );

// efface toutes les variables assignées
$smarty->clear_all_assign();

// n'affichera rien
print_r( $smarty->get_template_vars() );

?>
```

Voir aussi `clear_assign()`, `clear_config()`, `get_template_vars()`, `assign()` et `append()`.

clear_all_cache()

clear_all_cache()Efface les fichiers de cache des templates

clear_all_cache()

Description

void **clear_all_cache** ([int expire_time])

Vous pouvez passer un paramètre optionnel afin d'indiquer l'âge minimun que doivent avoir les fichiers de cache pour qu'ils soient effacés.

Exemple 13.6. Exemple avec clear_all_cache

```
<?php
// efface le cache
$smarty->clear_all_cache();

// efface tous les fichiers vieux d'une heure
$smarty->clear_all_cache(3600);
?>
```

Voir aussi `clear_cache()`, `is_cached()` et le cache.

clear_assign()

clear_assign()Efface la valeur d'une variable assignée

clear_assign()

Description

void **clear_assign** (mixed var)

Il peut s'agir d'une simple valeur ou d'un tableau de valeur.

Exemple 13.7. Exemple avec clear_assign()

```
<?php
// efface une variable
$smarty->clear_assign('Name');

// efface plusieurs variables
$smarty->clear_assign(array('Name','Address','Zip'));
?>
```

Voir aussi `clear_all_assign()`, `clear_config()`, `get_template_vars()`, `assign()` et `append()`.

clear_cache()

clear_cache()Efface le cache d'un template spécifique

clear_cache()

Description

void **clear_cache** (string template [, string cache_id [, string compile_id [, int expire_time]]])

- Si vous avez plusieurs fichiers de cache pour ce template, vous pouvez en spécifier un en particulier en passant son identifiant *cache_id* en deuxième paramètre.
- Vous pouvez aussi passer un identifiant de compilation *$compile_id* en troisième paramètre. Vous pouvez grouper des templates ensemble afin qu'ils puissent être supprimés en groupe. Référez-vous à la section sur le cache pour plus d'informations.
- Vous pouvez passer un quatrième paramètre pour indiquer un âge minimum en secondes que le fichier en cache doit avoir avant d'être effacé.

Exemple 13.8. Exemple avec clear_cache()

```
<?php
// efface le fichier de cache de ce template
$smarty->clear_cache('index.tpl');

// efface un fichier de cache grâce à son identifiant de cache
$smarty->clear_cache('index.tpl','CACHEID');
?>
```

Voir aussi le `clear_all_cache()` et la section sur le cache.

clear_compiled_tpl()

clear_compiled_tpl()Efface la version compilée d'un template spécifié

clear_compiled_tpl()

Description

void **clear_compiled_tpl** ([string tpl_file [, string compile_id [, int exp_time]]])

Utilisée pour effacer la version compilée du template spécifié ou de tous les templates si aucun n'est spécifié. Si vous passez uniquement un *$compile_id*, le template compilé correspondant à ce *$compile_id* sera effacé. Si vous passez un exp_time, les templates compilés plus vieux que *exp_time* secondes seront effacés, par défaut, tous les templates compilés seront effacés au vû de leurs âges. Cette fonction est destinée à un usage avancé et n'est habituellement pas utilisée.

Exemple 13.9. Exemple avec clear_compiled_tpl()

```
<?php
// efface la version compilée du template spécifié
$smarty->clear_compiled_tpl('index.tpl');

// efface tout le contenu du répertoire des templates compilés
$smarty->clear_compiled_tpl();
?>
```

Voir aussi `clear_cache()`.

clear_config()

clear_config()Efface toutes les variables de configuration assignées

clear_config()

Description

void **clear_config** ([string var])

Utilisée pour effacer toutes les variables de configuration assignées. Si un nom de variable est spécifié, seule cette variable sera effacée.

Exemple 13.10. Exemple avec clear_config()

```
<?php
// efface toutes les variables de configuration assignées
$smarty->clear_config();

// efface une seule variable
$smarty->clear_config('foobar');
?>
```

Voir aussi les `get_config_vars()`, les variables de configuration, les fichiers de configuration, `{config_load}`, `config_load()` et `clear_assign()`.

config_load()

config_load()Charge les données d'un fichier de configuration et les assigne au template

config_load()

Description

void **config_load** (string file [, string section])

Utilisée pour charger des données d'un fichier de configuration et les assigner a un template. Cette fonction fonctionne exactement comme la fonction de template `{config_load}`.

> **Note technique:** Comme pour Smarty 2.4.0, les variables de templates assignées sont conservées entre chaque appel à `fetch()` et `display()`. Les variables de configuration chargées avec `config_load()` sont globales. Les fichiers de configuration sont aussi compilés pour une exécution plus rapide et respecte les réglages de *$force_compile* et de *$compile_check*.

Exemple 13.11. Exemple avec config_load()

```
<?php
// charge les variables de configuration et les assigne
$smarty->config_load('my.conf');

// charge une section
$smarty->config_load('my.conf','foobar');
?>
```

Voir aussi `{config_load}`, `get_config_vars()`, `clear_config()` et les variables de configuration.

display()

display()Affiche le template

display()

Description

void **display** (string template [, string cache_id [, string compile_id]])

Utilisée pour afficher un template. Il faut fournir un type et un chemin de ressource template valides. Vous pouvez passer en second paramètre un identifiant de fichier de *$cache id*. Reportez-vous à la section cache pour plus de renseignements.

En tant que troisième paramètre optionnel, vous pouvez passer un identifiant de compilation *$compile_id*. C'est au cas où vous voudriez compiler plusieurs versions du même template, par exemple, pour avoir des templates compilés pour différents langages. Une autre utilité pour l'identifiant de compilation *$compile_id* est lorsque vous utilisez plus d'un *$template_dir* mais seulement un *$compile_dir*. Définissez un *$compile_id* séparé pour chaque *$template_dir*, sinon, les templates du même nom s'effaceront. Vous pouvez également définir la variable $compile_id une seule fois plutôt que de la passer à chaque appel à la fonction.

Exemple 13.12. Exemple avec display()

```
<?php
include(SMARTY_DIR.'Smarty.class.php');
$smarty = new Smarty();
$smarty->caching = true;

// ne fait un appel à la base de données que si le fichier
// de cache n'existe pas
if(!$smarty->is_cached('index.tpl')) {

    // quelques données
    $address = '245 N 50th';
    $db_data = array(
                      'Ville' => 'Lincoln',
                      'Pays' => 'Nebraska',
                      'Code postal' = > '68502'
     );

    $smarty->assign('Nom','Fred');
    $smarty->assign('Adresse',$address);
    $smarty->assign($db_data);

}

// affichage
$smarty->display('index.tpl');
?>
```

Utilisez la syntaxe des ressources templates pour afficher des fichiers en-dehors du répertoire *$template_dir*.

Exemple 13.13. Exemples de fonction d'affichage de ressources templates

```
<?php
// chemin absolu
$smarty->display('/usr/local/include/templates/header.tpl');
```

```
// chemin absolu (mêm chose)
$smarty->display('file:/usr/local/include/templates/header.tpl');

// chemin absolu Windows (on DOIT utiliser le préfixe "file:")
$smarty->display('file:C:/www/pub/templates/header.tpl');

// inclue à partir de la ressource template nommée "db"
$smarty->display('db:header.tpl');
?>
```

Voir aussi `fetch()` et `template_exists()`.

fetch()

fetch()Retourne le résultat du template

fetch()

Description

string **fetch** (string template [, string cache_id [, string $compile_id]])

Utilisée pour renvoyer le résultat du template plutôt que de l'afficher. Il faut passer un type et un chemin de ressource template valides. Vous pouvez passer un identifiant de cache *$cache id* en deuxième paramètre. Reportez-vous à la section cache pour plus de renseignements.

En tant que troisième paramètre optionnel, vous pouvez passer un identifiant de compilation *$compile_id*. C'est au cas où vous voudriez compiler plusieurs versions du même template, par exemple, pour avoir des templates compilés pour différents langages. Une autre utilité pour l'identifiant de compilation *$compile_id* est lorsque vous utilisez plus d'un *$template_dir* mais seulement un *$compile_dir*. Définissez un *$compile_id* séparé pour chaque *$template_dir*, sinon, les templates du même nom s'effaceront. Vous pouvez également définir la variable $compile_id une seule fois plutôt que de la passer à chaque appel à la fonction.

Exemple 13.14. Exemple avec fetch()

```
<?php
include('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// ne fait un appel à la base de données que si le fichier
// de cache n'existe pas
if(!$smarty->is_cached('index.tpl'))
{

    // quelques données
    $address = '245 N 50th';
    $db_data = array(
                        'Ville' => 'Lincoln',
                        'Pays' => 'Nebraska',
                        'Code postal' = > '68502'
    );

    $smarty->assign('Nom','Fred');
    $smarty->assign('Adresse',$address);
    $smarty->assign($db_data);

}

// récupère le résultat
$output = $smarty->fetch('index.tpl');

// fait quelque chose avec $output ici

echo $output;
?>
```

Exemple 13.15. Utilisation de fetch() pour envoyer un email

Le template `email_body.tpl` :

```
Cher {$contact.name},

Bienvenu et merci d'être devenu membre de notre groupe d'utilisateur,

Cliquez sur le lien ci-dessous pour vous identifier avec votre nom d'utilisateur '{$contact.login_id}'
et vous pourrez utiliser nos forums.

http://{$smarty.server.SERVER_NAME}/login/

Liste principale
Quelques groupes d'utilisateurs

{include file="email_disclaimer.tpl"}
```

Le template `email_disclaimer.tpl` qui utilise le modificateur `{textformat}`.

```
{textformat wrap=40}
Unless you are named "{$contact.name}", you may read only the "odd numbered
words" (every other word beginning with the first) of the message above. If you have
violated that, then you hereby owe the sender 10 GBP for each even
numbered word you have read
{/textformat}
```

et le script PHP utilisant la fonction PHP `mail()` [http://php.net/function.mail]

```
<?php

// Récupération du contact depuis une base de données eg utilisation de pear ou adodb
$query  = 'select name, email, login_id from contacts where contact_id='.$contact_id;
$contact = $db->getRow($sql);
$smarty->assign('contact', $contact);

mail($contact['email'], 'Subject', $smarty->fetch('email_body.tpl'));

?>
```

Voir aussi `{fetch}` `display()`, `{eval}` et `template_exists()`.

get_config_vars()

get_config_vars()Retourne la valeur de la variable de configuration passée en paramètre

get_config_vars()

Description

array **get_config_vars** ([string varname])

Si aucun paramètre n'est donné, un tableau de toutes les variables de configuration chargées est retourné.

Exemple 13.16. Exemple avec get_config_vars()

```
<?php

// récupère la variable de configuration chargée #foo#
$foo = $smarty->get_config_vars('foo');

// récupère toutes les variables de configuration chargées
$all_config_vars = $smarty->get_config_vars();

// les affiche a l'écran
print_r($all_config_vars);
?>
```

Voir aussi `clear_config()`, `{config_load}`, `config_load()` et `get_template_vars()`.

get_registered_object()

get_registered_object()Retourne la référence d'un objet enregistré

get_registered_object()

Description

array **get_registered_object** (string object_name)

Utile quand vous voulez accéder directement à un objet enregistré avec une fonction utilisateur. Lisez la documentation sur les objets pour plus d'informations.

Exemple 13.17. Exemple avec get_registered_object()

```
<?php
function smarty_block_foo($params, &$smarty)
{
  if (isset[$params['object']]) {
    // récupère la référence de l'objet enregistré
    $obj_ref =&amp; $smarty->get_registered_object($params['object']);
    // $obj_ref est maintenant une référence vers l'objet
  }
}
?>
```

Voir aussi `register_object()`, `unregister_object()` et la section sur les objets.

get_template_vars()

get_template_vars()Retourne la valeur assignée passée en paramètre

get_template_vars()

Description

array **get_template_vars** ([string varname])

Si aucun paramètre n'est donné, un tableau de toutes les variables assignées est retourné.

Exemple 13.18. Exemple avec get_template_vars

```
<?php
// récupère la variable 'foo' assignée au template
// get assigned template var 'foo'
$myVar = $smarty->get_template_vars('foo');

// récupère toutes les variables assignées a ce template
$all_tpl_vars = $smarty->get_template_vars();

// les affiche a l'écran
print_r($all_tpl_vars);
?>
```

Voir aussi `assign()`, `{assign}`, `append()`, `clear_assign()`, `clear_all_assign()` et `get_config_vars()`.

is_cached()

is_cached()Retourne TRUE s'il y a un fichier de cache valide pour ce template

is_cached()

Description

bool **is_cached** (string template [, string cache_id [, string compile_id]])

- Celà fonctionne seulement si *$caching* est défini à TRUE, voir aussi la section sur le cache pour plus d'informations.
- Vous pouvez aussi passer en second paramètre un identifiant de *$cache_id* au cas où vous voudriez plusieurs fichiers de cache pour ce template.
- Vous pouvez donner un *$compile id* en tant que troisième paramètre. Si vous ne spécifiez pas ce paramètre, le *$compile_id* persistant sera utilisé.
- Si vous ne voulez pas passer un *$cache_id* mais plutôt un *$compile_id*, vous devez passer NULL en tant que *$cache_id*.

> **Note technique:** Si is_cached() retourne TRUE, il charge en fait le cache existant et le stocke en interne. Tout appel supplémentaire à display() ou fetch() retournera ce contenu stocké en interne sans tenter de recharger le fichier en cache. Celà évite des problématiques d'accès concurents, lorsqu'un second processus efface le cache entre l'appel de is_cached() et l'appel à display() comme dans l'un de nos exemples ci-dessus. Celà signifie également que les appels à clear_cache() et les changements de paramètres du cache peuvent n'avoir aucun effet alors que is_cached() a retourné TRUE.

Exemple 13.19. Exemple avec is_cached()

```
<?php
$smarty->caching = true;

if(!$smarty->is_cached('index.tpl')) {
//aucun appel à la base de donnée
}

$smarty->display('index.tpl');
?>
```

Exemple 13.20. Exemple avec is_cached() et plusieurs templates

```
<?php
$smarty->caching = true;

if(!$smarty->is_cached('index.tpl', 'FrontPage')) {
  //appel de la base de données, assignation des variables
}

$smarty->display('index.tpl', 'FrontPage');
?>
```

Voir aussi clear_cache(), clear_all_cache() et la section sur le cache.

load_filter()

load_filter()Charge un plugin de filtrage

load_filter()

Description

void **load_filter** (string type, string name)

Le premier argument spécifie le type du filtre et peut prendre l'une des valeurs suivantes : `pre`, `post` ou `output`. Le second argument spécifie le nom du plugin de filtrage.

Exemple 13.21. Chargement de plugins de filtrage

```
<?php

// charge un pré-filtre nommé 'trim'
$smarty->load_filter('pre', 'trim');

// charge un autre pré-filtre nommé 'datefooter'
$smarty->load_filter('pre', 'datefooter');

// charge un filtre de sortie nommé 'compress'
$smarty->load_filter('output', 'compress');

?>
```

Voir aussi `register_prefilter()`, `register_postfilter()`, `register_outputfilter()`, *$autoload_filters* et les fonctionnalités avancées.

register_block()

register_block()Déclare dynamiquement des plugins de fonction de blocs

register_block()

Description

void **register_block** (string name, mixed impl, bool cacheable, mixed cache_attrs)

Utilisée pour déclarer dynamiquement des plugins de fonction de blocs. Il faut passer en paramètre le nom de la fonction de blocs, suivi du nom de la fonction PHP qui l'implémente.

La fonction PHP de callback *function* peut être soit :

- Une chaîne de caractères contenant la fonction *name*
- Un tableau sous la forme `array(&$object, $method)` où `&$object` est une référence d'objet et `$method` une chaîne contenant le nom de la méthode
- Un tableau sous la forme `array($class, $method)` où `$class` est le nom de la classe et `$method` est une méthode de la classe.

Les paramètre *cacheable* et *cache_attrs* peuvent être omis dans la plupart des cas. Voir Contrôler la mise en cache des sorties des Plugins pour plus d'informations concernant cette utilisation.

Exemple 13.22. Exemple avec register_block()

```
<?php
// Déclaration de la fonction
function do_translation ($params, $content, &$smarty, &$repeat) {
    if ($content) {
        $lang = $params['lang'];
        // fait de la traduction avec la variable $content
        echo $translation;
    }
}

// Enregistrement avec Smarty
$smarty->register_block('translate', 'do_translation');
?>
```

Le template Smarty :

```
{translate lang='br'}Bonjour le monde !{/translate}
```

Voir aussi `unregister_block()` et les plugins de fonction de blocs.

register_compiler_function()

register_compiler_function()Déclare dynamiquement un plugin de fonction de compilation

register_compiler_function()

Description

bool **register_compiler_function** (string name, mixed impl, bool cacheable)

Il faut passer en paramètres le nom de la fonction de compilation, suivi par la fonction PHP qui l'implémente.

La fonction PHP de callback *function* peut être soit :

- Une chaîne de caractères contenant la fonction *name*
- Un tableau sous la forme `array(&$object, $method)` où `&$object` est une référence d'objet et `$method` une chaîne contenant le nom de la méthode
- Un tableau sous la forme `array($class, $method)` où `$class` est le nom de la classe et `$method` est une méthode de la classe.

Le paramètre *cacheable* peut être omis dans la plupart des cas. Voir Contrôler la mise en cache des sorties des Plugins pour plus d'informations concernant cette utilisation.

Voir aussi `unregister_compiler_function()` et les plugins de fonction de compilation.

register_function()

register_function()Déclare dynamiquement des plugins de fonction de templates

register_function()

Description

void **register_function** (string name, mixed impl [, bool cacheable [, mixed cache_attrs]])

Il faut passer en paramètres le nom de la fonction de templates, suivi par le nom de la fonction PHP qui l'implémente.

La fonction PHP de callback *function* peut être soit :

- Une chaîne de caractères contenant la fonction *name*
- Un tableau sous la forme `array(&$object, $method)` où `&$object` est une référence d'objet et `$method` une chaîne contenant le nom de la méthode
- Un tableau sous la forme `array($class, $method)` où `$class` est le nom de la classe et `$method` est une méthode de la classe.

Les paramètres *cacheable* et *cache_attrs* peut être omis dans la plupart des cas. Voir Contrôler la mise en cache des sorties des Plugins pour plus d'informations concernant cette utilisation.

Exemple 13.23. Exemple avec register_function()

```
<?php
$smarty->register_function('date_now', 'print_current_date');

function print_current_date ($params) {
    extract($params);
    if(empty($format))
        $format="%b %e, %Y";
    echo strftime($format,time());
}

?>
```

Où le template est :

```
{date_now}

{* ou, formaté différemment *}
{date_now format="%Y/%m/%d"}
```

Voir aussi `unregister_function()` et les plugins de fonction.

register_modifier()

register_modifier()Déclare dynamiquement un plugin de modificateur

register_modifier()

Description

void **register_modifier** (string name, mixed impl)

Il faut passer en paramètre le nom du modificateur de variables, suivi de la fonction PHP qui l'implémente.

La fonction PHP de callback *function* peut être soit :

- Une chaîne de caractères contenant la fonction *name*
- Un tableau sous la forme `array(&$object, $method)` où `&$object` est une référence d'objet et `$method` une chaîne contenant le nom de la méthode
- Un tableau sous la forme `array($class, $method)` où `$class` est le nom de la classe et `$method` est une méthode de la classe.

Exemple 13.24. register_modifier()

```
<?php
// Associons la fonction PHP stripslashes a un modificateur Smarty.
$smarty->register_modifier('ss', 'stripslashes');
?>
```

Où le template est :

```
<?php
{* utiliser 'sslash' pour utiliser la fonction PHP strislashes() *}
{$var|sslash}
?>
```

Voir aussi `unregister_modifier()`, `register_function()`, les modifieurs, l'extension de Smarty avec des plugins et la création de plugins modifieurs.

register_object()

register_object()Enregistre un objet à utiliser dans un template

register_object()

Description

void **register_object** (string object_name, object object, array allowed_methods_properties, boolean format, array block_methods)

Reportez-vous à la section objet de ce manuel pour plus d'informations.

Voir aussi `get_registered_object()` et `unregister_object()`.

register_outputfilter()

register_outputfilter()Déclare dynamiquement des filtres de sortie

register_outputfilter()

Description

void **register_outputfilter** (mixed function)

Utilisée pour déclarer dynamiquement des filtres de sortie, pour agir sur la sortie d'un template avant qu'il ne soit affiché. Reportez-vous à la section filtres de sortie pour plus d'information sur le sujet.

La fonction PHP de callback *function* peut être soit :

- Une chaîne de caractères contenant la fonction *name*
- Un tableau sous la forme `array(&$object, $method)` où `&$object` est une référence d'objet et `$method` une chaîne contenant le nom de la méthode
- Un tableau sous la forme `array($class, $method)` où `$class` est le nom de la classe et `$method` est une méthode de la classe.

Voir aussi `unregister_outputfilter()`, `load_filter()`, *$autoload_filters* et les filtres de sortie de template.

register_postfilter()

register_postfilter()Déclare dynamiquement des filtres de post-compilation

register_postfilter()

Description

void **register_postfilter** (mixed function)

Utilisée pour déclarer dynamiquement des filtres de post-compilation pour y faire passer des templates une fois qu'ils ont été compilés. Reportez-vous à la section filtres de post-compilation de templates pour avoir plus de renseignements sur la façon de paramétrer les fonctions de post-compilation.

La fonction PHP de callback *function* peut être soit :

- Une chaîne de caractères contenant la fonction *name*
- Un tableau sous la forme `array(&$object, $method)` où `&$object` est une référence d'objet et `$method` une chaîne contenant le nom de la méthode
- Un tableau sous la forme `array($class, $method)` où `$class` est le nom de la classe et `$method` est une méthode de la classe.

Voir aussi `unregister_postfilter()`, `register_prefilter()`, `load_filter()`, *$autoload_filters* et les filtres de sortie de template.

register_prefilter()

register_prefilter()Déclare dynamiquement des filtres de pré-compilation

register_prefilter()

Description

void **register_prefilter** (mixed function)

Utilisée pour déclarer dynamiquement des filtres de pré-compilation pour y faire passer des templates avant qu'ils ne soient compilés. Reportez-vous à la section filtres de pré-compilation de templates pour avoir plus de renseignements sur la façon de paramétrer les fonctions de pré-compilation.

La fonction PHP de callback *function* peut être soit :

- Une chaîne de caractères contenant la fonction *name*
- Un tableau sous la forme `array(&$object, $method)` où `&$object` est une référence d'objet et `$method` une chaîne contenant le nom de la méthode
- Un tableau sous la forme `array($class, $method)` où `$class` est le nom de la classe et `$method` est une méthode de la classe.

Voir aussi `unregister_prefilter()`, `register_postfilter()`, `register_ouputfilter()`, `load_filter()`, *$autoload_filters* et les filtres de sortie de template.

register_resource()

register_resource()Déclare dynamiquement une ressource plugin

register_resource()

Description

void **register_resource** (string name, array resource_funcs)

Utilisée pour déclarer dynamiquement une ressource plugin dans Smarty. Il faut passer en paramètre le nom de la ressource et le tableau des fonctions PHP qui l'implémentent. Reportez-vous à la section ressources templates pour avoir plus d'informations sur la façon de paramétrer une fonction récupérant des templates.

> **Note technique:** Un nom de ressource doit être composé d'au moins deux caractères. Les noms de ressources d'un seul caractère seront ignorés et utilisés comme étant une partie du chemin du fichier, comme avec $smarty->display('c:/path/to/index.tpl');

- Le tableau de fonctions PHP *resource_funcs* doit être composé de 4 ou 5 éléments.
- S'il est composé de 4 éléments, les éléments seront les noms de fonctions pour, respectivement, `source`, `timestamp`, `secure` et `trusted` de la ressource.
- S'il est composé de 5 éléments, le premier élément devra être une référence sur un objet ou le nom d'une classe de l'objet ou une classe implémentant la ressource et les 4 éléments suivants doivent être les noms des méthodes implémentant `source`, `timestamp`, `secure` et `trusted`.

Exemple 13.25. Exemple avec register_resource()

```
<?php
$smarty->register_resource('db', array(
                           'db_get_template',
                           'db_get_timestamp',
                           'db_get_secure',
                           'db_get_trusted')
                           );
?>
```

Voir aussi `unregister_resource()` et les ressources de template.

trigger_error()

trigger_error()Affiche un message d'erreur

trigger_error()

Description

void **trigger_error** (string error_msg [, int level])

Cette fonction peut-être utilisée pour afficher un message d'erreur en utilisant Smarty. Le paramètre *level* peut prendre l'une des valeures utilisées par la fonction PHP `trigger_error()` [http:/ / php.net/ trigger_error], i.e. `E_USER_NOTICE`, `E_USER_WARNING`, etc. Par défaut il s'agit de `E_USER_WARNING`.

Voir aussi *$error_reporting*, le débogage et Troubleshooting.

template_exists()

template_exists()Vérifie si un template spécifique existe

template_exists()

Description

bool **template_exists** (string template)

Elle accepte soit un chemin vers le template, soit une ressource de type chaîne de caractères spécifiant le nom du template.

Exemple 13.26. template_exists()

Cet exemple utilise `$_GET['page']` pour inclure le contenu d'un template. Si le template n'existe pas, une page d'erreur sera affiché à la place. Le fichier `page_container.tpl` :

```
<html>
 <head><title>{$title}</title></head>
 <body>
  {include file='page_top.tpl'}

  {* inclure le contenu du milieu de la page *}
  {include file=$page_mid}

  {include file='page_footer.tpl'}
 </body>
```

Et le script PHP

```
<?php

// Définit le nom du fichier eg index.inc.tpl
$mid_template = $_GET['page'].'.inc.tpl';

if( !$smarty->template_exists($mid_template) ){
   $mid_template = 'page_not_found.inc.tpl';
}
$smarty->assign('page_mid', $mid_template);

$smarty->display('page_container.tpl');

?>
```

Voir aussi `display()`, `fetch()`, `{include}` et `{insert}`.

unregister_block()

unregister_block()Désalloue dynamiquement un plugin de fonction de blocs

unregister_block()

Description

void **unregister_block** (string name)

Utilisée pour désallouer dynamiquement un plugin de fonction de blocs. Passez en paramètre le nom *name* du bloc.

Voir aussi `register_block()` et les plugins de fonctions de blocs.

unregister_compiler_function()

unregister_compiler_function()Désalloue dynamiquement une fonction de compilation

unregister_compiler_function()

Description

void **unregister_compiler_function** (string name)

Passez en paramètre le nom *name* de la fonction de compilation.

Voir aussi `register_compiler_function()` et les plugins de fonction de compilation.

unregister_function()

unregister_function()Désalloue dynamiquement un plugin de fonction de templates

unregister_function()

Description

void **unregister_function** (string name)

Passez en paramètres le nom de la fonction de templates.

Exemple 13.27. Exemple avec unregister_function()

```
<?php

// nous ne voulons pas que les designers de templates aient accès
// au système de fichiers.
$smarty->unregister_function('fetch');

?>
```

Voir aussi `register_function()`.

unregister_modifier()

unregister_modifier()Désalloue dynamiquement un plugin modificateur de variable

unregister_modifier()

Description

void **unregister_modifier** (string name)

Passez en paramètre le nom du modificateur de templates.

Exemple 13.28. Exemple avec unregister_modifier()

```
<?php

// nous ne voulons pas que les designers de templates
// suppriment les balises des élements

$smarty->unregister_modifier('strip_tags');

?>
```

Voir aussi `register_modifier()` et les plugins modificateur.

unregister_object()

unregister_object()Désalloue dynamiquement un objet

unregister_object()

Description

void **unregister_object** (string object_name)

Voir aussi `register_object()` et la section sur les objets.

unregister_outputfilter()

unregister_outputfilter()Désalloue dynamiquement un filtre de sortie

unregister_outputfilter()

Description

void **unregister_outputfilter** (string function_name)

Utilisée pour désallouer dynamiquement un filtre de sortie.

Voir aussi `register_outputfilter()` et les filtres de sortie de template.

unregister_postfilter()

unregister_postfilter()Désallouer dynamiquement un filtre de post-compilation

unregister_postfilter()

Description

void **unregister_postfilter** (string function_name)

Voir aussi `register_postfilter()` et les filtres de post-compilation.

unregister_prefilter()

unregister_prefilter()Désalloue dynamiquement un filtre de pré-compilation

unregister_prefilter()

Description

void **unregister_prefilter** (string function_name)

Voir aussi `register_prefilter()` et les pré-filtres.

unregister_resource()

unregister_resource()Désalloue dynamiquement un plugin ressource

unregister_resource()

Description

void **unregister_resource** (string name)

Passez en paramètre le nom de la ressource.

Exemple 13.29. Exemple avec unregister_resource()

```
<?php
$smarty->unregister_resource("db");
?>
```

Voir aussi `register_resource()` et les ressources de template.

Chapitre 14. Cache

Table des matières

Le cache est utilisée pour accélérer l'appel de `display()` ou de `fetch()` en sauvegardant leur résultat dans un fichier. Si un fichier de cache est disponible lors d'un appel, il sera affiché sans qu'il ne soit nécessaire de regénérer le résultat. Le système de cache peut accélérer les traitements de façon impressionnante, en particulier les templates dont la compilation est très longue. Comme le résultat de `display()` ou de `fetch()`est dans le cache, un fichier de cache peut être composé de plusieurs fichiers de templates, plusieurs fichiers de configuration, etc.

Comme les templates sont dynamiques, il est important de faire attention à la façon dont les fichiers de cache sont générés, et pour combien de temps. Si par exemple vous affichez la page d'accueil de votre site Web dont le contenu ne change pas souvent, il peut être intéressant de mettre cette page dans le cache pour une heure ou plus. A l'inverse, si vous affichez une page de météo mise a jour toutes les minutes, mettre cette page en cache n'a aucun sens.

Paramétrer le cache

La première chose à faire est d'activer le cache en mettant *$caching* `= 1 (ou 2)`.

Exemple 14.1. Activation du cache

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = 1;

$smarty->display('index.tpl');
?>
```

Avec le cache activé, la fonction `display('index.tpl')` va afficher le template mais sauvegardera par la même occasion une copie du résultat dans un fichier (de cache) du répertoire *$cache_dir*. Au prochain appel de `display('index.tpl')`, le fichier de cache sera préféré à la réutilisation du template.

> **Note technique:** Les fichiers situés dans *$cache_dir* sont nommés de la même façon que les templates. Bien qu'ils aient une extension `.php`, ils ne sont pas vraiment directement exécutable. N'éditez surtout pas ces fichiers !

Tout fichier de cache a une durée de vie limitée déterminée par *$cache_lifetime*. La valeur par défaut est 3600 secondes, i.e. 1 heure. Une fois que cette durée est dépassée, le cache est regénéré. Il est possible de donner une durée d'expiration propre à chaque fichier de cache en réglant *$caching*=2. Se reporter à la documentation de *$cache_lifetime* pour plus de détails.

Exemple 14.2. Réglage individuel de $cache_lifetime

```
<?php
```

```
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = 2; // régler la durée de vie individuellement

// règle la durée de vie du cache a 15 minutes pour index.tpl
$smarty->cache_lifetime = 300;
$smarty->display('index.tpl');

// règle la durée de vie du cache à 1 heure pour home.tpl
$smarty->cache_lifetime = 3600;
$smarty->display('home.tpl');

// NOTE : le réglage suivant ne fonctionne pas quand $caching = 2. La durée de vie
// du fichier de cache de home.tpl a déja été réglée a 1 heure et ne respectera
// plus la valeur de $cache_lifetime. Le cache de home.tpl expirera toujours
// dans 1 heure.
$smarty->cache_lifetime = 30; // 30 secondes
$smarty->display('home.tpl');
?>
```

Si *$compile_check* est actif, chaque fichier de template et de configuration qui a un rapport avec le fichier de cache sera vérifié pour détecter une éventuelle modification. Si l'un de ces fichiers a été modifié depuis que le fichier de cache a été généré, le cache est immédiatement regénéré. Ce processus est couteux, donc, pour des raisons de performances, mettez ce paramètre à `FALSE` pour une application en production.

Exemple 14.3. Activation de $compile_check

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = 1;
$smarty->compile_check = true;

$smarty->display('index.tpl');
?>
```

Si *$force_compile* est actif, les fichiers de cache sont toujours regénérés. Ceci revient finalement à désactiver le cache. *$force_compile* est utilisé à des fins de débogage, un moyen plus efficace de désactiver le cache est de régler *$caching* `= 0`.

La fonction `is_cached()` permet de tester si un template a ou non un fichier de cache valide. Si vous disposez d'un template en cache qui requiert une requête à une base de données, vous pouvez utiliser cette méthode plutôt que $compile_check.

Exemple 14.4. Exemple avec is_cached()

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = 1;

if(!$smarty->is_cached('index.tpl')) {
   // pas de cache disponible, on assigne
   $contents = get_database_contents();
   $smarty->assign($contents);
}

$smarty->display('index.tpl');
```

```
?>
```

Vous pouvez rendre dynamiques seulement certaines parties d'une page avec la fonction de template `{insert}`. Imaginons que toute une page doit être mise en cache à part une bannière en bas à droite. En utilisant une fonction `{insert}` pour la bannière, vous pouvez garder cet élément dynamique dans le contenu qui est en cache. Reportez-vous à la documentation `{insert}` pour plus de détails ainsi que des exemples.

Vous pouvez effacer tous les fichiers du cache avec la fonction `clear_all_cache()`, ou de façon individuelle (ou par groupe) avec la fonction `clear_cache()`.

Exemple 14.5. Nettoyage du cache

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = 1;

// efface le fichier de cache du template 'index.tpl'
$smarty->clear_cache('index.tpl');

// efface tous les fichiers du cache
$smarty->clear_all_cache();

$smarty->display('index.tpl');
?>
```

Caches multiples pour une seule page

Vous pouvez avoir plusieurs fichiers de caches pour un même appel aux fonctions `display()` ou `fetch()`. Imaginons qu'un appel a display('index.tpl') puisse avoir plusieurs résultats, en fonction de certaines conditions, et que vous vouliez des fichiers de cache séparés pour chacun d'eux. Vous pouvez faire celà en passant un identifiant de cache (*$cache_id*) en deuxiéme paramètre à l'appel de fonction.

Exemple 14.6. Passage d'un $cache_id à display()

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

$smarty->display('index.tpl',$my_cache_id);
?>
```

Nous passons ci-dessus la variable `$my_cache_id` à `display()` comme identifiant de cache. Pour chaque valeur distincte de `$my_cache_id`, un fichier de cache distinct va être créé. Dans cet exemple, `article_id` a été passé dans l'URL et est utilisé en tant qu'identifiant de cache.

> **Note technique:** Soyez prudent en passant des valeurs depuis un client (navigateur Web) vers Smarty (ou vers n'importe quelle application PHP). Bien que l'exemple ci-dessus consistant à utiliser article_id depuis l'URL puisse paraetre commode, le résultat peut s'avérer mauvais. L'identifiant de cache est utilisé pour créer un répertoire sur le

système de fichiers, donc si l'utilisateur décide de donner une trés grande valeur à article_id ou d'écrire un script qui envoie des article_id de façon aléatoire, celà pourra causer des problémes coté serveur. Assurez-vous de bien tester toute donnée passée en paramètre avant de l'utiliser. Dans cet exemple, peut-être savez-vous que article_id a une longueur de 10 caractéres, est exclusivement composé de caractéres alph-numériques et doit avoir une valeur contenue dans la base de données. Vérifiez-le bien !

Assurez-vous de bien passer le même identifiant aux fonctions `is_cached()` et `clear_cache()`.

Exemple 14.7. Passer un cache_id a is_cached()

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

$my_cache_id = $_GET['article_id'];

if(!$smarty->is_cached('index.tpl',$my_cache_id)) {
  // pas de fichier de cache dispo, on assigne donc les variables
  $contents = get_database_contents();
  $smarty->assign($contents);
}

$smarty->display('index.tpl',$my_cache_id);
?>
```

Vous pouvez effacer tous les fichiers de cache pour un identifiant de cache particulier en passant `NULL` en tant que premier paramètre à `clear_cache()`.

Exemple 14.8. Effacement de tous les fichiers de cache pour un identifiant de cache particulier

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// efface tous les fichiers de cache avec "sports" comme identifiant
$smarty->clear_cache(null,'sports');

$smarty->display('index.tpl','sports');
?>
```

De cette manière, vous pouvez "grouper" vos fichiers de cache en leur donnant le même identifiant.

Groupes de fichiers de cache

Vous pouvez faire des groupements plus élaborés en paramétrant les groupes de *$cache_id*. Il suffit de séparer chaque sous-groupe avec une barre verticale `|` dans la valeur de *$cache_id*. Vous pouvez faire autant de sous-groupes que vous le désirez.

- Vous pouvez voir les groupes de cache comme une hiérarchie de dossiers. Par exemple, un groupe de cache `'a|b|c'` peut être considéré comme la hiérarchie de dossiers `'/a/b/c/'`.

- `clear_cache(null,'a|b|c')` supprimera les fichiers `'/a/b/c/*'`. `clear_cache(null,'a|b')` supprimera les fichiers `'/a/b/*'`.
- Si vous spécifiez un *$compile_id* de cette façon `clear_cache(null,'a|b','foo')` il sera traité comme un groupe de cache apposé `'/a/b/c/foo/'`.
- Si vous spécifiez un nom de template de cette façon `clear_cache('foo.tpl','a|b|c')` alors Smarty tentera d'effacer `'/a/b/c/foo.tpl'`.
- Vous ne POUVEZ PAS effacer un nom de template spécifié sous un groupe de cache multiple comme `'/a/b/*/foo.tpl'`, le groupement de cache fonctionne UNIQUEMENT de gauche à droite. Vous pourriez vouloir grouper vos templates sous un groupe de cache simple hiérarchisé pour être capable de les effacer comme un groupe.

Le groupement de cache ne devrait pas être confondu avec votre hiérarchie de dossiers de templates, le groupement de cache n'a aucune connaissance de la façon dont vos templates sont structurés. Donc, par exemple, si vous avez une structure de template comme `themes/blue/index.tpl` et que vous voulez être capable d'effacer tous les fichiers de cache pour le thème «blue», vous devriez créer une structure de groupe de cache qui reflète la structure de fichiers de vos templates, comme `display('themes/blue/index.tpl','themes|blue')`, et les effacer avec `clear_cache(null,'themes|blue')`.

Exemple 14.9. Groupes d'identifiants de cache

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;

$smarty->caching = true;

// efface tous les fichiers de cache avec "sports|basketball" comme premiers
// groupes d'identifiants de cache
$smarty->clear_cache(null,'sports|basketball');

// efface tous les fichiers de cache "sports" comme premier groupe d'identifiants.
// Inclue donc "sports|basketball" ou "sports|nimportequoi|nimportequoi|..."
$smarty->clear_cache(null,'sports');

$smarty->display('index.tpl','sports|basketball');
?>
```

Contrôler la mise en cache des sorties des Plugins

Depuis Smarty-2.6.0, la mise en cache des plugins peut être déclarée lors de leur inscription. Les troisièmes paramètres de `register_block()`, `register_compiler_function()` et `register_function()` sont appelés *$cacheable* et valent `TRUE` par défaut, ce qui est aussi le comportement par défaut des versions de Smarty précédent la 2.6.0

Lors de l'inscription d'un plugin avec `$cacheable=false`, le plugin est appelé à chaque fois que la page est affichée, même si la page vient du cache. La fonction plugin se comporte presque comme la fonction `{insert}`.

Contrairement à `{insert}` les attributs pour le plugin ne sont pas mis en cache par défaut. Celà peut être le cas en utilisant le quatrième paramètre *$cache_attrs*. *$cache_attrs* est un tableau de noms d'attributs qui doivent être mis en cache, pour que la fonction plugin reçoive les valeurs telles qu'elles étaient définies lorsque la page a été mise en cache, à chaque récupération à partir du cache.

Exemple 14.10. Eviter la mise en cache du résultat d'un plugin

```
<?php
$smarty->caching = 1;

function remaining_seconds($params, &$smarty) {
    $remain = $params['endtime'] - time();
    if ($remain >=0) {
        return $remain . " second(s)";
    } else {
        return "done";
    }
}

$smarty->register_function('remaining', 'remaining_seconds', false, array('endtime'));

if (!$smarty->is_cached('index.tpl')) {
    // récupération de $obj à partir de la page et assignation...
    $smarty->assign_by_ref('obj', $obj);
}

$smarty->display('index.tpl');
?>
```

Où `index.tpl` contient :

```
Time Remaining: {remaining endtime=$obj->endtime}
```

Le nombre de secondes avant que la date de fin de `$obj` ne soit atteinte change à chaque affichage de la page, même si la page est mise en cache. Comme l'attribut endtime est mis en cache, il n'y a que l'objet qui ait besoin d'être extrait de la base de données lors de la mise en cache de la page, mais pas lors des affichages ultérieurs de la page.

Exemple 14.11. Eviter la mise en cache d'une portion du template

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->caching = true;

function smarty_block_dynamic($param, $content, &$smarty) {
    return $content;
}
$smarty->register_block('dynamic', 'smarty_block_dynamic', false);

$smarty->display('index.tpl');
?>
```

Où `index.tpl` contient :

```
Création de la page : {'0'|date_format:'%D %H:%M:%S'}

{dynamic}

Heure actuelle : {'0'|date_format:'%D %H:%M:%S'}

... faîtes quelque chose ici ...

{/dynamic}
```

Lors du rechargement de la page, vous remarquerez que les deux dates sont différentes. L'une est «dynamic» et l'autre est «static». Vous pouvez faire ce que vous voulez entre `{dynamic}...{/dynamic}` et être sûrs que cela ne sera pas mis en cache comme le reste de la page.

Chapitre 15. Fonctionnalités avancées

Table des matières

Objets

Smarty donne l'accès aux objets [http://php.net/object] PHP à travers les templates. Il y a 2 moyens d'y avoir accès.

- Le premier consiste à allouer les objets au template puis de les utiliser avec une syntaxe similaire a celles des fonctions personnalisées.
- Le deuxième moyen consiste à assigner des objets aux templates et de les utiliser comme n'importe quelle variable.

La première méthode a une syntaxe beaucoup plus sympathique. Elle est aussi plus sécurisée, puisqu'un objet alloué ne peut avoir accès qu'a certaines méthodes et propriétés. Néanmoins, **un objet alloué ne peut pas avoir de lien sur lui-même ou être mis dans un tableau d'objet**, etc. Vous devez choisir la méthode qui correspond a vos besoins, mais tGchez d'utiliser la première méthode autant que possible afin de réduire la syntaxe des templates au minimum.

Si l'option de sécurité est activée, aucune méthode ou fonctions privées n'est accessible (commentant par "_"). S'il existe une méthode et une propriété du même nom, c'est la méthode qui sera utilisée.

Vous pouvez restreindre l'accès aux méthodes et aux propriétés en les listant dans un tableau en tant que troisième paramètre d'allocation.

Par défaut, les paramètres passés aux objets depuis le template le sont de la même façon que les fonctions utilisateurs les récupèrent. Le premier paramètre correspond à un tableau associatif, le second à l'objet Smarty. Si vous souhaitez que les paramètres soient passés un à un, comme dans un appel traditionnel, définissez registration, quatrième paramètre optionnel, à FALSE.

Le cinquième paramètre optionnel n'a d'effet que si le paramètre *format* vaut `true` et il contient une liste de méthodes qui doivent être traitées comme des blocks. Celà signifie que ces méthodes ont un tag fermant dans le template (`{foobar->meth2}...{/foobar->meth2}`) et que les paramètres de ces méthodes fonctionnent de la même façon que les paramètres des blocks de fonctions des plugins : Ils contiennent 4 paramètres *$params*, *$content*, *&$smarty* et *&$repeat* et ils fonctionnent de la même façon que les blocks de fonctions des plugins.

Exemple 15.1. Utilisation d'un objet alloué ou assigné

```
<?php
// la classe

class My_Object() {
  function meth1($params, &$smarty_obj) {
    return 'Ceci est ma methode 1';
  }
}
```

```
$myobj = new My_Object;
// enregistre l'objet
$smarty->register_object('foobar',$myobj);
// on restreint l'accès a certaines méthodes et propriétés en les listant
$smarty->register_object('foobar',$myobj,array('meth1','meth2','prop1'));
// pour utiliser le format habituel de paramètre objet, passez le booléen = false
$smarty->register_object('foobar',$myobj,null,false);

// on peut aussi assigner des objets. Assignez par référence quand c'est possible
$smarty->assign_by_ref('myobj', $myobj);

$smarty->display('index.tpl');
?>

?>
```

Et voici comment accéder à vos objets dans `index.tpl` :

```
{* accès a notre objet enregistré *}
{foobar->meth1 p1="foo" p2=$bar}

{* on peut aussi assigner la sortie *}
{foobar->meth1 p1="foo" p2=$bar assign="output"}
the output was {$output)

{* access our assigned object *}
{$myobj->meth1("foo",$bar)}
```

Voir aussi `register_object()` et `assign()`.

Filtres de pré-compilation

Les filtres de pré-compilation sont des fonctions PHP que vos templates exécutent *avant qu'ils ne soient compilés*. Celà peut être utile pour pré-traiter vos templates afin d'enlever les commentaires inutiles, garder un oeil sur ce que les gens mettent dans leurs templates, etc.

Les filtre de pré-compilations peuvent être soit déclarés, soit chargés à partir des répertoires de plugins en utilisant la fonction `load_filter()` ou en réglant la variable *$autoload_filters*.

Smarty passera à la fonction le code source en tant que premier argument, et attendra en retour le code modifié.

Exemple 15.2. Utilisation un filtre de pré-compilation de template

Ceci va effacer tous les commentaires de la source du template.

```
<?php
// mettre ceci dans votre application
function remove_dw_comments($tpl_source, &$smarty)
{
    return preg_replace("/<!--#.*-->/U",'',$tpl_source);
}

// enregistrer le filtre de pré-compilation
$smarty->register_prefilter('remove_dw_comments');
$smarty->display('index.tpl');
?>
```

Voir aussi `register_prefilter()`, les post-filtres et `load_filter()`.

Filtres de post-compilation

Les filtres de post-compilation sont des fonctions PHP que vos templates exécutent *après avoir été compilés*. Les filtres de post-compilation peuvent être soit déclarés, soit chargés depuis les répertoires des plugins en utilisant la fonction `load_filter()` ou en réglant la variable *$autoload_filters*. Smarty passera le template compilé en tant que premier paramètre et attendra de la fonction qu'elle retourne le résultat de l'exécution.

Exemple 15.3. Utilisation d'un filtre de post-compilation de templates

```
<?php
// mettez celà dans votre application
function add_header_comment($tpl_source, &$smarty)
{
    return "<?php echo \"<!-- Créé par Smarty ! -->\n\"; ?>\n".$tpl_source;
}

// enregistre le filtre de post-compilation
$smarty->register_postfilter('add_header_comment');
$smarty->display('index.tpl');
?>
```

Votre template Smarty `index.tpl` ressemblera, après compilation à :

```
<!-- Créé par Smarty ! -->
{* reste du contenu du template... *}
```

Voir aussi `register_postfilter()`, les pré-filtres et `load_filter()`.

Filtres de sortie

Quand le template est appelé via les fonctions `display()` ou `fetch()`, sa sortie est envoyée à travers un ou plusieurs filtres de sorties. Ils diffèrent des filtres de post-compilation dans le sens où ils agissent sur la sortie des templates, une fois exécutés, et non sur les sources des templates.

Les filtres de sortie peuvent être soit déclarés soit chargés depuis les répertoires des plugins en utilisant la fonction `load_filter()` ou en réglant la variable *$autoload_filters*. Smarty passera la sortie du template en premier argument et attendra de la fonction qu'elle retourne le résultat de l'exécution.

Exemple 15.4. Utilisation d'un filtre de sortie

```
<?php
// mettez ceci dans votre application
function protect_email($tpl_output, &$smarty)
{
    $tpl_output =
       preg_replace('!(\S+)@([a-zA-Z0-9\.\-]+\.([a-zA-Z]{2,3}|[0-9]{1,3}))!',
                    '$1%40$2', $tpl_output);
    return $tpl_output;
}

// enregistre le filtre de sortie
$smarty->register_outputfilter('protect_email');
$smarty->display('index.tpl');

// dorénavant toute occurence d'un adresse email dans le résultat du template
// aura un protection simple contre les robots spammers
?>
```

Voir aussi `register_outpurfilter()`, `load_filter()`, *$autoload_filters*, les filtres de post-compilation et *$plugins_dir*.

Fonction de gestion du cache

Une alternative au mécanisme de cache par défaut (basé sur des fichiers de cache) consiste à spécifier une fonction de gestion de cache utilisateur qui sera utilisée pour lire, écrire et effacer les fichiers de cache.

Il suffit de créer dans votre application une fonction que Smarty utilisera pour la gestion du cache et d'assigner le nom de cette fonction à la variable de classe $cache_handler_func. Smarty utilisera alors cette fonction pour gérer les données du cache.

- Le premier argument est l'action, qui sera `read`, `write` and `clear`.
- Le second paramètre est l'objet Smarty.
- Le troisième est le contenu du cache. Pour écrire, Smarty passe le contenu du cache dans ces paramètres. Pour lire, Smarty s'attend à ce que votre fonction accepte ce paramètre par référence et que vous le remplissiez avec les données du cache. Pour effacer, il suffit de passer une variable fictive car cette dernière n'est pas utilisée.
- Le quatrième paramètre est le nom du fichier de template (utile pour lire/écrire).
- Le cinquième paramètre est l'identifiant *$cache_id*.
- Le sixième est l'identifiant optionnel *$compile_id*.
- Le septième et dernier paramètre *$exp_time* a été ajouté dans Smarty-2.6.0.

Exemple 15.5. Exemple d'utilisation de MySQL pour la source du cache

```
<?php
/*****

exemple d'usage :

include('Smarty.class.php');
include('mysql_cache_handler.php');

$smarty = new Smarty;
$smarty->cache_handler_func = 'mysql_cache_handler';

$smarty->display('index.tpl');


la base mysql est attendu dans ce format :

create database SMARTY_CACHE;

create table CACHE_PAGES(
CacheID char(32) PRIMARY KEY,
CacheContents MEDIUMTEXT NOT NULL
);

*****/

function mysql_cache_handler($action, &$smarty_obj, &$cache_content, $tpl_file=null, $cache_id=null, $c
{
  // l'hôte de la bd, l'utilisateur, et le mot de passe
  $db_host = 'localhost';
  $db_user = 'myuser';
```

```
  $db_pass = 'mypass';
  $db_name = 'SMARTY_CACHE';
  $use_gzip = false;

  // crée un identifiant de cache unique
  $CacheID = md5($tpl_file.$cache_id.$compile_id);

  if(! $link = mysql_pconnect($db_host, $db_user, $db_pass)) {
 $smarty_obj->_trigger_error_msg("cache_handler: could not connect to database");
    return false;
  }
  mysql_select_db($db_name);

  switch ($action) {
    case 'read':
      // récupère le cache dans la base de données
      $results = mysql_query("select CacheContents from CACHE_PAGES where CacheID='$CacheID'");
      if(!$results) {
        $smarty_obj->_trigger_error_msg('cache_handler: query failed.');
      }
      $row = mysql_fetch_array($results,MYSQL_ASSOC);

      if($use_gzip && function_exists('gzuncompress')) {
        $cache_content = gzuncompress($row['CacheContents']);
      } else {
        $cache_content = $row['CacheContents'];
      }
      $return = $results;
      break;
    case 'write':
      // sauvegarde le cache dans la base de données

      if($use_gzip && function_exists("gzcompress")) {
        // compresse le contenu pour gagner de la place
        $contents = gzcompress($cache_content);
      } else {
        $contents = $cache_content;
      }
      $results = mysql_query("replace into CACHE_PAGES values(
                                            '$CacheID',
                                            '".addslashes($contents)."')
                                            ");
      if(!$results) {
        $smarty_obj->_trigger_error_msg('cache_handler: query failed.');
      }
      $return = $results;
      break;
    case 'clear':
      // efface les données du cache
      if(empty($cache_id) && empty($compile_id) && empty($tpl_file)) {
        // les efface toutes
        $results = mysql_query('delete from CACHE_PAGES');
      } else {
        $results = mysql_query("delete from CACHE_PAGES where CacheID='$CacheID'");
      }
      if(!$results) {
        $smarty_obj->_trigger_error_msg('cache_handler: query failed.');
      }
      $return = $results;
      break;
    default:
      // erreur, action inconnue
      $smarty_obj->_trigger_error_msg("cache_handler: unknown action \"$action\"");
      $return = false;
      break;
  }
  mysql_close($link);
  return $return;

}
```

```
?>
```

Ressources

Les templates peuvent provenir d'une grande variété de ressources. Quand vous affichez (`display()`) ou récupérez (`fetch()`) un template, ou quand vous incluez un template dans un autre template, vous fournissez un type de ressource, suivi par le chemin approprié et le nom du template. Si une ressource n'est pas explicitement donnée, la valeur de la variable *$default_resource_type* sera utilisée.

Templates depuis $template_dir

Les templates du répertoire *$template_dir* n'ont pas besoin d'une ressource template, bien que vous puissiez utiliser la ressource "file" pour être cohérent. Vous n'avez qu'à fournir le chemin vers le template que vous voulez utiliser, relatif au répertoire racine *$template_dir*.

Exemple 15.6. Utilisation de templates depuis $template_dir

```
<?php
$smarty->display("index.tpl");
$smarty->display("admin/menu.tpl");
$smarty->display("file:admin/menu.tpl"); // le même que ci-dessus
?>

{* le template Smarty *}
{include file="index.tpl"}
{include file="file:index.tpl"} {* le même que ci-dessus *}
```

Templates à partir de n'importe quel répertoire

Les templates en-dehors du répertoire *$template_dir* nécessitent le type de ressource template, suivi du chemin absolu et du nom du template.

Exemple 15.7. Utilisation d'un template depuis n'importe quel répertoire

```
<?php
$smarty->display('file:/export/templates/index.tpl');
$smarty->display('file:/path/to/my/templates/menu.tpl');
?>
```

Le template Smarty :

```
{include file='file:/usr/local/share/templates/navigation.tpl'}
```

Chemin de fichiers Windows

Si vous utilisez Windows, les chemins de fichiers sont la plupart du temps sur un disque identifié par une lettre (c:) au début du chemin. Assurez-vous de bien mettre `file:` dans le chemin pour éviter des conflits d'espace de noms et obtenir les résultats escomptés.

Exemple 15.8. Utilisation de templates avec des chemins de fichiers Windows

```
<?php
$smarty->display('file:C:/export/templates/index.tpl');
$smarty->display('file:F:/path/to/my/templates/menu.tpl');
?>
```

Le template Smarty :

```
{include file='file:D:/usr/local/share/templates/navigation.tpl'}
```

Templates depuis d'autres sources

Vous pouvez récupérer les templates à partir de n'importe quelle source à laquelle vous avez accès avec PHP : base de données, sockets, LDAP et ainsi de suite. Il suffit d'écrire les fonctions de ressource plugins et de les enregistrer auprès de Smarty.

Reportez-vous à la section ressource plugins pour plus d'informations sur les fonctions que vous êtes censé fournir.

> **NOTE:** Notez que vous ne pouvez pas écraser la ressource `file:` native, toutefois, vous pouvez fournir une ressource qui récupère un template depuis le système de fichier par un autre moyen en l'enregistrant sous un autre nom de ressource.

Exemple 15.9. Utilisation de ressources utilisateurs

```
<?php
// mettez ces fonctions quelque part dans votre application
function db_get_template ($tpl_name, &$tpl_source, &$smarty_obj)
{
    // requête BD pour récupérer le template
    // et remplir $tpl_source
    $sql = new SQL;
    $sql->query("select tpl_source
                  from my_table
                 where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_source = $sql->record['tpl_source'];
        return true;
    } else {
        return false;
    }
}

function db_get_timestamp($tpl_name, &$tpl_timestamp, &$smarty_obj)
{
    // requête BD pour remplir $tpl_timestamp
    $sql = new SQL;
    $sql->query("select tpl_timestamp
                  from my_table
                 where tpl_name='$tpl_name'");
    if ($sql->num_rows) {
        $tpl_timestamp = $sql->record['tpl_timestamp'];
        return true;
    } else {
        return false;
    }
}

function db_get_secure($tpl_name, &$smarty_obj)
```

```
{
    // on suppose que tous les templates sont svrs
    return true;
}

function db_get_trusted($tpl_name, &$smarty_obj)
{
    // pas utilisée pour les templates dans notre cas
}

// enregistre le nom de ressource "db"
$smarty->register_resource("db", array("db_get_template",
                                       "db_get_timestamp",
                                       "db_get_secure",
                                       "db_get_trusted"));

// utilise la ressource depuis le script PHP
$smarty->display("db:index.tpl");
?>
```

Le template Smarty :

```
{include file='db:/extras/navigation.tpl'}
```

Fonction de gestion de template par défaut

Vous pouvez spécifier une fonction qui sera utilisée pour récupérer le contenu d'un template dans le cas où le template ne peut pas être récupéré depuis sa ressource. Une utilisation possible est la création de templates à la volée.

Exemple 15.10. utilisation de la fonction de gestion de template par défaut

```
<?php
// mettez cette fonction quelque part dans votre application

function make_template ($resource_type, $resource_name, &$template_source, &$template_timestamp,
&$smarty_obj)
{
  if( $resource_type == 'file' ) {
    if ( ! is_readable ( $resource_name )) {
      // crée le fichier de template et renvoie le contenu
      $template_source = "Ceci est un nouveau template.";
      $template_timestamp = time();
      $smarty_obj->_write_file($resource_name,$template_source);
      return true;
    }
  } else {
    // pas un fichier
    return false;
  }
}

// règle la fonction par défaut
$smarty->default_template_handler_func = 'make_template';
?>
```

Chapitre 16. Etendre Smarty avec des plugins

Table des matières

La version 2.0 a introduit l'architecture de plugin qui est utilisée pour pratiquement toutes les fonctionnalités personnalisables de Smarty. Ceci comprend :

- les fonctions
- les modificateurs
- les fonctions de blocs
- les fonctions de compilation
- les filtres de pré-compilation
- les filtres de post-compilation
- les filtres de sorties
- les ressources
- les insertions

A part pour les ressources, la compatibilité avec les anciennes façons d'enregistrer les fonctions de gestion avec l'API register_ est conservée. Si vous n'avez pas utilisé cette API et que vous avez à la place directement modifié les variables de classes `$custom_funcs`, `$custom_mods` et d'autres, vous devez alors modifier vos scripts pour utiliser l'API ou convertir vos fonctionnalités personnalisées en plugins.

Comment fonctionnent les plugins

Les plugins sont toujours chargés à la demande. Seuls les modificateurs de variables, les ressources, etc invoqués dans les scripts de templates seront chargés. De plus, chaque plugin n'est chargé qu'une fois, et ce même si vous avez plusieurs instances de Smarty qui tournent dans la même requête.

Les filtres de post/pré-compilation et les filtres de sortie sont des cas un peu spéciaux. Comme ils ne sont pas mentionnés dans les templates, ils doivent être déclarés ou chargés explicitement via les fonctions de l'API avant que le template ne soit exécuté. L'ordre dans lequel les filtres multiples d'un même type sont exécutés dépend de l'ordre dans lequel ils sont enregistrés ou chargés.

Le répertoire de plugin peut être une chaîne de caractères contenant un chemin ou un tableau contenant de multiples chemins. Pour installer un plugin, placez-le simplement dans un de ces dossiers et Smarty l'utilisera automatiquement.

Conventions de nommage

Les fichiers et les fonctions de plugins doivent suivre une convention de nommage très spécifique pour être localisés par Smarty.

Les fichiers de **plugins** doivent être nommés de la façon suivante :

type.*nom*.php

- Où type est l'une des valeurs suivantes :
  - function
  - modifier
  - block
  - compiler
  - prefilter
  - postfilter
  - outputfilter
  - resource
  - insert

- Et nom doit être un identifiant valide : lettres, nombres et underscore seulement, voir les variables php [http://php.net/language.variables].
- Quelques exemples : function.html_select_date.php, resource.db.php, modifier.spacify.php.

Les fonctions de **plugins** dans les fichiers de plugins doivent être nommées de la façon suivante :

smarty_*type*, _*nom*()

- Les significations de type et de nom sont les mêmes que précédemment.
- Un exemple de nom de modificateur foo serait function smarty_modifier_foo().

Smarty donnera des messages d'erreurs appropriés si le fichier de plugin n'est pas trouvé ou si le fichier ou la fonction de plugin ne sont pas nommés correctement.

Ecrire des plugins

Les plugins peuvent être soit chargés automatiquement par Smarty depuis le système de fichier, soit être déclarés pendant l'exécution via une fonction register_* de l'API. Ils peuvent aussi être désalloués en utilisant une fonction unregister_* de l'API.

Pour les plugins qui ne sont pas enregistrés pendant l'exécution, le nom des fonctions n'ont pas à suivre la convention de nommage.

Si certaines fonctionnalités d'un plugin dépendent d'un autre plugin (comme c'est le cas de certains plugins accompagnant Smarty), alors la maniére appropriée de charger le plugin est la suivante :

```
<?php
require_once $smarty->_get_plugin_filepath('function', 'html_options');
?>
```

Une règle générale est que chaque objet Smarty est toujours passé au plugin en tant que dernier paramètre, sauf pour deux exceptions :

- les modificateurs ne sont pas passés du tout à l'objet Smarty
- les blocs récupèrent le paramètre *$repeat* passé après l'objet Smarty afin de conserver une compatibilité avec les anciennes versions de Smarty.

Les fonctions de templates

```
void smarty_function_name()($params, &$smarty);
array $params;
object &$smarty;
```

Tous les attributs passés aux fonctions de template à partir du template sont contenus dans le tableau associatif *$params*.

Le retour de la fonction sera substituée à la balise de fonction du template (fonction `{fetch}` par exemple). Sinon, la fonction peut simplement accomplir une autre tâche sans sortie (la fonction `{assign}` par exemple).

Si la fonction a besoin d'assigner des variables aux templates ou d'utiliser d'autres fonctionnalités fournies par Smarty, elle peut recevoir un objet *$smarty* pour celà.

Exemple 16.1. Fonction de plugin avec sortie

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :  function.eightball.php
 * Type :     fonction
 * Nom :      eightball
 * Rôle :     renvoie une phrase magique au hasard
 * -------------------------------------------------------------
 */
function smarty_function_eightball($params, &$smarty)
{
    $answers = array('Yes',
                     'No',
                     'No way',
                     'Outlook not so good',
                     'Ask again soon',
                     'Maybe in your reality');

    $result = array_rand($answers);
    return $answers[$result];
}
?>
```

peut être utilisée dans le template de la façon suivante :

```
Question: Will we ever have time travel?
Answer: {eightball}.
```

Exemple 16.2. Fonction de plugin sans sortie

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :  function.assign.php
 * Type :     fonction
 * Nom :      assign
 * Purpose :  assigne une valeur a une variable de template
 * -------------------------------------------------------------
 */
function smarty_function_assign($params, &$smarty)
{
    extract($params);

    if (empty($var)) {
        $smarty->trigger_error("assign: missing 'var' parameter");
        return;
    }

    if (!in_array('value', array_keys($params))) {
        $smarty->trigger_error("assign: missing 'value' parameter");
        return;
    }

    $smarty->assign($var, $value);
}
?>
```

Voir aussi : `register_function()` et `unregister_function()`.

Modificateurs

Les modificateurs sont de petites fonctions appliquées à une variable de template avant qu'elle ne soit affichée ou utilisée dans un autre contexte. Les modificateurs peuvent être chaînés entre eux.

```
mixed smarty_modifier_name($value, $param1);
mixed $value;
[mixed $param1, ...];
```

Le premier paramètre passé au modificateur est la valeur sur laquelle le modificateur est supposé opérer. Les autres paramétres peuvent être optionnels, dépendant de quel genre d'opération doit être effectué.

Le modificateur doit retourner [http://php.net/return] le résultat de son exécution.

Exemple 16.3. Plugin modificateur simple

Ce plugin est un alias d'une fonction PHP. Il n'a aucun paramètre supplémentaires.

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :  modifier.capitalize.php
 * Type :     modificateur
 * Name :     capitalize
 * Rôle :     met une majuscule aux mots d'une phrase
 * -------------------------------------------------------------
 */
function smarty_modifier_capitalize($string)
{
    return ucwords($string);
}
?>
```

Exemple 16.4. Un plugin modificateur un peu plus complexe

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :  modifier.truncate.php
 * Type :     modificateur
 * Name :     truncate
 * Rôle :     Tronque une chaene a une certaine longueur si
 *            nécessaire, la coupe optionnellement au milieu
 *            d'un mot et ajoute la chaene $etc
 * -------------------------------------------------------------
 */
function smarty_modifier_truncate($string, $length = 80, $etc = '...',
                                  $break_words = false)
{
    if ($length == 0)
        return '';

    if (strlen($string) > $length) {
        $length -= strlen($etc);
        $fragment = substr($string, 0, $length+1);
        if ($break_words)
            $fragment = substr($fragment, 0, -1);
        else
            $fragment = preg_replace('/\s+(\S+)?$/', '', $fragment);
        return $fragment.$etc;
    } else
        return $string;
}
?>
```

Voir aussi : `register_modifier()` et `unregister_modifier()`.

Fonctions de blocs

```
void smarty_block_name()($params, $content, &$smarty);
array $params;
mixed $content;
object &$smarty;
```

Les fonctions de blocs sont des fonctions de la forme `{func} .. {/func}`. En d'autres mots, elles englobent des blocs de template et opèrent sur les contenus de ces blocs. Les fonctions de blocs ont la priorité sur les fonctions utilisateurs de même nom, ce qui signifie que vous ne pouvez avoir une fonction utilisateur `{func}` et une fonction de bloc `{func}..{/func}`.

- Par défaut, l'implémentation de votre fonction est appelée deux fois par Smarty : une fois pour la balise ouvrante et une autre fois pour la balise fermante (voir `$repeat` ci-dessous sur la façon de modifier ce comportement).
- Seule la balise ouvrante d'une fonction de bloc peut avoir des attributs. Tous les attributs passés par le template aux fonctions de templates sont contenus dans le tableau associatif *$params*. Votre fonction a aussi accès aux attributs de la balise ouvrante quand c'est la balise fermante qui est exécutée.
- La valeur de la variable *$content* est différente selon si votre fonction est appelée pour la balise ouvrante ou la balise fermante. Si c'est pour la balise ouvrante, elle sera à `NULL` et si c'est la balise fermante, elle sera égale au contenu du bloc de template. Notez que le bloc de template aura déjà été exécuté par Smarty, vous recevrez donc la sortie du template et non sa source.
- Le paramètre *$repeat* est passé par référence à la fonction d'implémentation et fournit la possibilité de contrôler le nombre d'affichage du bloc. Par défaut, *$repeat* vaut `TRUE` lors du premier appel à la fonction de bloc (le bloc

d'ouverture du tag) et FALSE lors de tous les autres appels à la fonction de bloc (le bloc de fermeture du tag). Chaque fois que la fonction d'implémentation retourne avec le paramètre *$repeat* vallant TRUE, le contenu situé {func}...{/func} est évalué et la fonction d'implémentation est appelé une nouvelle fois avec le nouveau bloc de contenu en tant que paramètre *$content*.

Si vous imbriqué des fonctions de bloc, il est possible de connaître la fonction de bloc parente grâce à la variable $smarty->_tag_stack. Faîtes un var_dump() [http://php.net/var_dump] dessus et la structure devrait apparaître.

Exemple 16.5. Fonction de bloc

```
<?php
/*
* Smarty plugin
* -------------------------------------------------------------
* Fichier :  block.translate.php
* Type :     bloc
* Nom :      translate
* Rôle :     traduire un bloc de texte
* -------------------------------------------------------------
*/
function smarty_block_translate($params, $content, &$smarty, &$repeat)
{
   // n'affiche que lors de la balise fermante
   if(!$repeat){
      if (isset($content)) {
         $lang = $params['lang'];
         // effectuer une bonne traduction ici avec $content
         return $translation;
      }
   }
}
?>
```

Voir aussi : register_block() et unregister_block().

Fonctions de compilation

Les fonctions de compilation sont appelées durant la compilation du template. Elles sont utiles pour injecter du code PHP ou du contenu "statique variant avec le temps" (bandeau de pub par ex.). Si une fonction de compilation et une fonction personnalisée ont le même nom, la fonction de compilation a priorité.

```
mixed smarty_compiler_name()($tag_arg, &$smarty);
string $tag_arg;
object &$smarty;
```

Les fonctions de compilation ont deux paramètres : une chaîne contenant la balise - en gros, tout, depuis le nom de la fonction jusqu'au délimiteur de fin - et l'objet Smarty. Elles sont censées retourner le code PHP qui doit être injecté dans le template compilé.

Exemple 16.6. Fonction de compilation simple

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :   compiler.tplheader.php
 * Type :      compilation
 * Nom :       tplheader
```

```
 * Rôle :      Renvoie l'en-tête contenant le nom du fichier
 *             source et le temps de compilation.
 * -------------------------------------------------------------
 */
function smarty_compiler_tplheader($tag_arg, &amp;$smarty)
{
    return "\necho '" . $smarty-&gt;_current_file . " compiled at " . date('Y-m-d H:M'). "';";
}
?>
```

Cette fonction peut-être appelée depuis le template comme suivant :

```
{* cette fonction n'est executée que lors de la compilation *}
{tplheader}
```

Le code PHP résultant dans les templates compilés ressemblerait à ça :

```
<?php
echo 'index.tpl compiled at 2002-02-20 20:02';
?>
```

Voir aussi : `register_compiler_function()` et `unregister_compiler_function()`.

filtres de pré-compilation/filtres de post-compilation

Les filtres de pré-compilation et les filtres de post-compilation ont des concepts très proches. Ils différent dans leur exécution, plus précisément dans le moment où ils sont exécutés.

```
string smarty_prefilter_name()($source, &$smarty);
string $source;
object &$smarty;
```

Les filtres de pré-compilation sont utilisés pour transformer la source d'un template juste avant la compilation. Le premier paramètre passé à la fonction de filtre de pré-compilation est la source du template, éventuellement modifiée par d'autres filtres de pré-compilations. Le plugin est supposé retourner la source modifiée. Notez que cette source n'est sauvegardée nulle part, elle est seulement utilisé pour la compilation.

```
string smarty_postfilter_name()($compiled, &$smarty);
string $compiled;
object &$smarty;
```

Les filtres de post-compilation sont utilisés pour modifier la sortie du template (le code PHP) juste après que la compilation a été faîte mais juste avant que le template ne soit sauvegardé sur le système de fichiers. Le premier paramètre passé à la fonction de filtre de post-compilation est le code du template compilé, éventuellement déja modifié par d'autres filtres de post-compilations. Le plugin est censé retourner la version modifiée du code.

Exemple 16.7. Plugin de filtre de post-compilation

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :  prefilter.pre01.php
 * Type :     filtre de pré-compilation
 * Nom :      pre01
 * Rôle :     Passe les balises HTML en minuscules.
 * -------------------------------------------------------------
 */
 function smarty_prefilter_pre01($source, &$smarty)
 {
```

```
    return preg_replace('!<(\w+)[^>]+>!e', 'strtolower("$1")', $source);
 }
?>
```

Exemple 16.8. Plugin de filtre de post-compilation

```
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :  postfilter.post01.php
 * Type:      filtre de post-compilation
 * Nom :      post01
 * Rôle :     Renvoie du code qui liste toutes les variables
 *            du template.
 * -------------------------------------------------------------
 */
 function smarty_postfilter_post01($compiled, &$smarty)
 {
     $compiled = "<pre>\n<?php print_r(\$this->get_template_vars()); ?>\n</pre>" . $compiled;
     return $compiled;
 }
?>
```

Voir aussi `register_prefilter()`, `unregister_prefilter()` `register_postfilter()` et `unregister_postfilter()`.

Filtres de sortie

Les plugins de filtres de sortie opèrent sur la sortie du template, après que le template a été chargé et exécuté, mais avant que la sortie ne soit affichée.

```
string smarty_outputfilter_name()($template_output, &$smarty);
string $template_output;
object &$smarty;
```

Le premier paramètre passé à la fonction du filtre de sortie est la sortie du template qui doit être modifiée et le second paramètre est l'instance de Smarty appelant le plugin. Le plugin est supposé faire un traitement et en retourner le résultat.

Exemple 16.9. Plugin de filtre de sortie

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :  outputfilter.protect_email.php
 * Type :     filtre de sortie
 * Nom :      protect_email
 * Rôle:     Convertie les @ en %40 pour protéger des
 *           robots spammers.
 * -------------------------------------------------------------
 */
function smarty_outputfilter_protect_email($output, &$smarty)
{
     return preg_replace('!(\S+)@([a-zA-Z0-9\.\-]+\.([a-zA-Z]{2,3}|[0-9]{1,3}))!',
                         '$1%40$2', $output);
}
?>
```

Voir aussi `register_outputfilter()` et `unregister_outputfilter()`.

Ressources

Les plugins ressources sont un moyen générique de fournir des sources de templates ou des composants de scripts PHP à Smarty. Quelques exemples de ressources : bases de données, LDAP, mémoire partagée, sockets, etc.

Il y au total quatre fonctions qui ont besoin d'être enregistrées pour chaque type de ressource. Chaque fonction reçoit le nom de la ressource demandée comme premier paramètre et l'objet Smarty comme dernier paramètre. Les autres paramètres dépendent de la fonction.

```
bool smarty_resource_name_source()($rsrc_name, &$source, &$smarty);
string $rsrc_name;
string &$source;
object &$smarty;
bool smarty_resource_name_timestamp()($rsrc_name, &$timestamp, &$smarty);
string $rsrc_name;
int &$timestamp;
object &$smarty;
bool smarty_resource_name_secure()($rsrc_name, &$smarty);
string $rsrc_name;
object &$smarty;
bool smarty_resource_name_trusted()($rsrc_name, &$smarty);
string $rsrc_name;
object &$smarty;
```

- La première fonction est supposée récupérer la ressource. Son second paramètre est une variable passée par référence où le résultat doit être stocké. La fonction est supposée retourner `TRUE` si elle réussit à récupérer la ressource et `FALSE` sinon.
- La seconde fonction est supposée récupérer la date de dernière modification de la ressource demandée (comme un timestamp UNIX). Le second paramètre est une variable passée par référence dans laquelle la date doit être stockée. La fonction est supposée renvoyer `TRUE` si elle réussit à récupérer la date et `FALSE` sinon.
- La troisième fonction est supposée retourner `TRUE` ou `FALSE` selon si la ressource demandée est sûre ou non. La fonction est utilisée seulement pour les ressources templates mais doit tout de même être définie.
- La quatrième fonction est supposée retourner `TRUE` ou `FALSE` selon si l'on peut faire confiance ou non à la ressource demandée. Cette fonction est utilisée seulement pour les composants de scripts PHP demandés par les balises `{include_php}` ou `{insert}` ayant un attribut *src*. Quoiqu'il en soit, elle doit être définie pour les ressources templates.

Exemple 16.10. resource plugin

```
<?php
/*
* Smarty plugin
* -------------------------------------------------------------
* Fichier :  resource.db.php
* Type :     ressource
* Nom :      db
* Rôle :     Récupére des templates depuis une base de données
* -------------------------------------------------------------
*/
function smarty_resource_db_source($tpl_name, &$tpl_source, &$smarty)
{
   // fait des requêtes BD pour récupérer votre template
   // et remplir $tpl_source
   $sql = new SQL;
```

```
   $sql->query("select tpl_source
   from my_table
   where tpl_name='$tpl_name'");
   if ($sql->num_rows) {
      $tpl_source = $sql->record['tpl_source'];
      return true;
   } else {
      return false;
   }
}

function smarty_resource_db_timestamp($tpl_name, &$tpl_timestamp, &$smarty)
{
   // fait des requêtes BD pour remplir $tpl_timestamp
   $sql = new SQL;
   $sql->query("select tpl_timestamp
   from my_table
   where tpl_name='$tpl_name'");
   if ($sql->num_rows) {
      $tpl_timestamp = $sql->record['tpl_timestamp'];
      return true;
   } else {
      return false;
   }
}

function smarty_resource_db_secure($tpl_name, &$smarty)
{
   // suppose que tous les templates sont svrs
   return true;
}

function smarty_resource_db_trusted($tpl_name, &$smarty)
{
   // inutilisée pour les templates
}
?>
```

Voir aussi : register_resource() et unregister_resource().

Insertions

Les plugins d'insertion sont utilisés pour implémenter les fonctions qui sont appelées par les balises `{insert}` dans les templates.

```
string smarty_insert_name()($params, &$smarty);
array $params;
object &$smarty;
```

Le premier paramètre passé à la fonction est une tableau associatif d'attributs.

La fonction d'insertion est supposée retourner le résultat qui sera substitué à la balise `{insert}` dans le template.

Exemple 16.11. Plugin d'insertion

```
<?php
/*
 * Smarty plugin
 * -------------------------------------------------------------
 * Fichier :  insert.time.php
 * Type :     temps
 * Nom :      time
 * Rôle :     Insert la date/heure courante conformément
 *            au format
```

```
 * ------------------------------------------------------------
 */
function smarty_insert_time($params, &$smarty)
{
    if (empty($params['format'])) {
        $smarty->trigger_error("insert time: missing 'format' parameter");
        return;
    }

    return strftime($params['format']);
}
?>
```

Partie IV. Appendices

Table des matières

Chapitre 17. Diagnostic des erreurs

Table des matières

Erreurs Smarty/PHP

Smarty peut identifier de nombreuses erreurs comme des attributs de balises manquants ou de noms de variables malformés. Dans ce cas-là, vous verrez apparaître une erreur semblable à :

Exemple 17.1. erreurs Smarty

```
Warning: Smarty: [in index.tpl line 4]: syntax error: unknown tag - '%blah'
in /path/to/smarty/Smarty.class.php on line 1041

Fatal error: Smarty: [in index.tpl line 28]: syntax error: missing section name
in /path/to/smarty/Smarty.class.php on line 1041
```

Smarty vous indique le nom du template, le numéro de la ligne et l'erreur. Après cela, vous pouvez connaître le numéro de ligne où il y a eu erreur dans la définition de la classe Smarty.

Il y a certaines erreurs que Smarty ne peut pas détecter, comme les balises fermantes manquantes. Ce type d'erreurs est la plupart du temps repéré dans la phase de compilation PHP du template compilé.

Exemple 17.2. Erreur d'analyse PHP

```
Parse error: parse error in /path/to/smarty/templates_c/index.tpl.php on line 75
```

Quand vous rencontrez une erreur d'analyse PHP, le numéro de la ligne indiqué est celui du fichier PHP compilé et non du template. Vous pouvez alors regarder le template et détecter l'erreur. Voici quelques erreurs fréquentes : balises fermantes pour `{if}{/if}` ou `{section}{/section}` manquantes, ou syntaxe logique incorrecte dans une instruction `{if}`. Si vous ne trouvez pas l'erreur, vous devrez alors ouvrir le fichier PHP compilé et aller à la ligne correspondante pour trouver d'où vient l'erreur.

Exemple 17.3. Autres erreurs communes

```
Warning: Smarty error: unable to read resource: "index.tpl" in...
ou
Warning: Smarty error: unable to read resource: "site.conf" in...
```

- Le dossier *$template_dir* est incorrect, n'existe pas ou le fichier the file `index.tpl` n'est pas dans le dossier

`templates/`.

- Une fonction `{config_load}` est dans un template (ou `config_load()` a été appelé) et soit `$config_dir` est incohérent, n'existe pas, ou `site.conf` n'est pas dans le dossier.

```
Fatal error: Smarty error: the $compile_dir 'templates_c' does not exist,
             or is not a directory...
```

- Soit le dossier *$compile_dir* n'est pas correctement défini, le dossier n'existe pas, ou `templates_c` est un fichier et non un dossier.

```
Fatal error: Smarty error: unable to write to $compile_dir '....
```

- Le dossier *$compile_dir* n'est pas accessible en écriture par le serveur web. Voir le bas de la page sur l'installation de Smarty pour les permissions.

```
Fatal error: Smarty error: the $cache_dir 'cache' does not exist,
or is not a directory. in /..
```

- Cela signifie que *$caching* est activé et soit le dossier *$cache_dir* n'est pas correctement défini, le dossier n'existe pas, ou `cache` est un fichier et non un dossier.

```
Fatal error: Smarty error: unable to write to $cache_dir '/...
```

- Cela signifie que *$caching* est activé et le dossier *$cache_dir* n'est pas accessible en écriture par le serveur web. Voir le bas de la page sur l'installation de Smarty pour les permissions.

Voir aussi le débogage, *$error_reporting* et `trigger_error()`.

Chapitre 18. Trucs et astuces

Table des matières

Gestion des variables non-assignées

Peut-être voudrez-vous des fois afficher une valeur par défaut pour une variable qui n'a pas été assignée, comme pour afficher ` ` afin que les couleurs de fond des tableaux fonctionnent. Beaucoup utiliseraient une instruction `{if}` pour gérer celà, mais il existe un moyen plus facile dans Smarty : l'utilisation du modificateur de variable `default`.

> **NOTE:** Les erreurs «de variable indéfinie» seront affichés si la fonction PHP `error_reporting()` [http://php.net/error_reporting] vaut `E_ALL` et qu'une variable n'a pas été assignée à Smarty.

Exemple 18.1. Afficher   quand une variable est vide

```
{* la méthode pas adaptée *}
{if $title eq ''}
    
{else}
   {$title}
{/if}


{* la bonne méthode *}
{$title|default:' '}
```

Voir aussi `default` et la gestion des variables par défaut.

Gestion des variables par défaut

Si une variable est utilisée fréquemment dans vos templates, lui appliquer le modificateur `default` peut être un peu fastidieux. Vous pouvez remédier à celà en lui assignant une valeur par défaut avec la fonction `{assign}`.

Exemple 18.2. Assigner une valeur par défaut à une variable de template

```
{* faites cela quelque part en haut de votre template *}
{assign var='title' value=$title|default:'no title'}

{* si $title est vide, il contiendra alors la valeur "no title" *}
{$title}
```

Voir aussi `default` et la gestion des variables non-assignées.

Passage du titre à un template d'en-tête

Quand la majorité de vos templates utilisent les mêmes en-tête et pied-de-page, il est d'usage de les mettre dans leurs propres templates et de les inclure (`{include}`). Mais comment faire si l'en-tête doit avoir un titre différent, selon la page d'où on vient ? Vous pouvez passer le titre à l'en-tête en tant qu' attribut quand il est inclus.

Exemple 18.3. Passer le titre au template d'en-tête

`mainpage.tpl` - Lorsque la page principal est construite, le titre «Man Page» est passé au `header.tpl` et sera utilisé en tant que titre.

```
{include file='header.tpl' title='Main Page'}
{* le corps du template va ici *}
{include file='footer.tpl'}
```

`archives.tpl` - Lorsque la page principal est construite, le titre sera «Archives». Notez que dans cet exemple, nous utilisons une variable du fichier `archives_page.conf` au lieu d'une variable classique.

```
{config_load file='archive_page.conf'}

{include file='header.tpl' title=#archivePageTitle#}
{* corps du template ici *}
{include file='footer.tpl'}
```

`header.tpl` - Notez que «Smarty News» est affiché si la variable `$title` n'est pas définie, en utilisant le modificateur de variable par `default`.

```
<html>
 <head>
  <title>{$title|default:'Smarty News'}</title>
 </head>
 <body>
```

`footer.tpl`

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

Dates

De façon générale, essayez de toujours passer les dates à Smarty sous forme de timestamp [http://php.net/time]. Cela permet aux designers de templates d'utiliser `date_format` pour avoir un contrôle total sur le formatage des dates et de comparer facilement les dates entre elles.

Exemple 18.4. Utilisation de date_format

```
{$startDate|date_format}
```

Affichera :

```
Jan 4, 2009
```

```
{$startDate|date_format:"%Y/%m/%d"}
```

Affichera :

```
2009/01/04
```

Les dates peuvent être comparées dans le template en utilisant les timestamps, comme ceci :

```
{if $date1 < $date2}
    ...
{/if}
```

En utilisant la fonction `{html_select_date}` dans un template, le programmeur veut en général convertir le résultat d'un formulaire en un timestamp. Voici une fonction qui devrait vous être utile.

Exemple 18.5. Conversion des éléments date d'un formulaire en timestamp

```
<?php

// celà suppose que vos éléments de formulaire soient nommés
// startDate_Day, startDate_Month, startDate_Year

$startDate = makeTimeStamp($startDate_Year, $startDate_Month, $startDate_Day);

function makeTimeStamp($year='', $month='', $day='')
{
   if(empty($year)) {
       $year = strftime('%Y');
   }
   if(empty($month)) {
       $month = strftime('%m');
   }
   if(empty($day)) {
       $day = strftime('%d');
   }

   return mktime(0, 0, 0, $month, $day, $year);
}
?>
```

Voir aussi `{html_select_date}`, `{html_select_time}`, `date_format` et *$smarty.now*,

WAP/WML

Les templates WAP/WML nécessitent un en-tête Content-Type [http://php.net/header] qui doit être passé avec le template. Le moyen le plus facile de faire celà est d'écrire une fonction utilisateur qui écrit l'en-tête. Si vous utilisez le cache, celà ne fonctionnera pas. Nous utiliserons donc une balise d'insertion ({insert}) (rappelez-vous que les balises d'insertion ne sont pas mises en cache !). Assurez-vous qu'aucune sortie rien n'est transmise au navigateur avant l'appel du template, sans quoi la modification de l'en-tête échouera.

Exemple 18.6. Utilisation d'{insert} pour écrire un en-tête Content-Type WML

```
<?php

// assurez-vous que Apache est configuré pour les extensions .wml !
// mettez cette fonction quelque part dans votre applications
// ou dans Smarty.addons.php
function insert_header()
{
   // cette fonction attend un argument $content
   if (empty($params['content'])) {
       return;
   }
   header($params['content']);
   return;
}

?>
```

votre template Smarty *doit* commencer avec la balise d'insertion :

```
{insert name=header content="Content-Type: text/vnd.wap.wml"}

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">

<!-- begin new wml deck -->
<wml>
 <!-- begin first card -->
 <card>
  <do type="accept">
   <go href="#two"/>
  </do>
  <p>
   Welcome to WAP with Smarty!
   Press OK to continue...
  </p>
 </card>
 <!-- begin second card -->
 <card id="two">
  <p>
   Pretty easy isn't it?
  </p>
 </card>
</wml>
```

Templates composants

Traditionnellemnt, la programmation avec des templates dans les applications se déroule de la façon suivante : d'abord vous récupérez vos variables dans l'application PHP (peut-être avec des requêtes en base de données), puis vous instanciez votre objet Smarty, `assign()` les variables et `display()` le template. Disons par exemple que nous avons un téléscripteur dans notre template. Nous récupérerions les données dans notre application, puis les assignerions ensuite pour les afficher. Mais ne serait-ce pas mieux de pouvoir ajouter ce téléscripteur à n'importe quelle application en incluant directement le template sans avoir à se soucier de la récupération des données ?

Vous pouvez réaliser celà en écrivant un plugin personnalisé pour récupérer le contenu et l'assigner à une variable du template.

Exemple 18.7. Template composant

`function.load_ticker.php` - Efface le fichier du répertoire des *$plugins*

```
<?php
```

```
// notre fonction pour récupérer les données
function fetch_ticker($symbol,&amp;$ticker_name,&amp;$ticker_price)
{
   // du traitement qui récupère $ticker_name
   // depuis la ressource ticker
   return $ticker_info;
}

function smarty_function_load_ticker($params, &$smarty)
{
   // appel de la fonction
   $ticker_info = fetch_ticker($params['symbol']);

   // assignation de la variable de template
   $smarty->assign($params['assign'], $ticker_info);
}
?>
```

index.tpl

```
{load_ticker symbol='SMARTY' assign="ticker"}

Stock Name: {$ticker.name} Stock Price: {$ticker.price}
```

Voir aussi {include_php}, {include} et {php}.

Dissimuler les adresses email

Vous-êtes vous déjà demandé pourquoi vos adresses emails sont sur autant de listes de diffusion de spam ? Une façon pour les spammers de récupérer les adresses est de parcourir les pages Web. Voici une façon de remédier à ce problème : mettre votre adresse email dans du Javascript brouillé au milieu de votre source HTML, sans que celà ne gêne l'affichage sur le navigateur Web. Cela est fait grâce au plugin {mailto}.

Exemple 18.8. Exemple de dissimulation d'une adresse email

```
<div id="contact">Envoyer une demande à
{mailto address=$EmailAddress encode='javascript' subject='Bonjour'}
```

> **Note technique:** Cette méthode n'est pas infaillible. Un spammer peut programmer son collecteur d'email pour passer outre cette astuce, mais c'est cependant peu probable.

Voir aussi escape et {mailto}.

Chapitre 19. Ressources

La page Web de Smarty se trouve à l'adresse suivante : http://smarty.php.net/

- Vous pouvez souscrire à la mailing liste en envoyant un email à `smarty-general-subscribe@lists.php.net`. Les archives de la mailing list se trouvent à l'adresse suivante : ici [http:/ / marc.theaimsgroup.com/ ?l=smarty-general&r=1&w=2]
- Les forums sur http://www.phpinsider.com/smarty-forum/
- Le wiki sur http://smarty.incutio.com/
- Le chat sur irc.freenode.net#smarty [http://smarty.incutio.com/]
- La FAQ ici [http://smarty.incutio.com/?page=SmartyFrequentlyAskedQuestions] et ici [http://smarty.php.net/faq.php]

Chapitre 20. BUGS

Vérifiez le fichier de BUGS fourni avec la dernière version de Smarty ou consultez le site Web.