

# Initiation aux templates en PHP avec Smarty

par Eric POMMEREAU

Date de publication : 15 décembre 2006

Dernière mise à jour : 5 novembre 2008

Cet article a pour but d'initier le lecteur à l'utilisation des templates en PHP avec le moteur de template Smarty.

Une autre idée, sous-jacente, est de convaincre de l'intérêt d'utiliser un système de template dans les développements PHP.

Il ne s'agit en aucun cas ici de vous en présenter toutes les fonctionnalités, mais de vous donner un aperçu de ce que l'on peut faire avec Smarty et comment on peut le faire.

J'ai souhaité rendre Smarty le plus accessible possible en mettant l'accent sur les aspects pratiques (exemples, recettes de code et debugage), j'espère que vous aurez plaisir à me lire.

|   |    |
|---|----|
| I - Introduction.....                               | 3  |
| I-A - Remerciements.....                            | 3  |
| I-B - Présentation.....                             | 3  |
| I-C - Avantages et inconvénients.....               | 3  |
| I-C-1 - Inconvénients.....                          | 3  |
| I-C-2 - Avantages.....                              | 4  |
| I-D - Installer.....                                | 4  |
| I-E - Tester.....                                   | 4  |
| II - Retour sur les métas-structures.....           | 6  |
| II-A - Les tableaux.....                            | 6  |
| II-A-1 - Les tableaux indexés.....                  | 6  |
| II-A-2 - Les tableaux associatifs.....              | 7  |
| II-B - Les objets (syntaxe PHP5).....               | 7  |
| II-C - Les fichiers xml (avec SimpleXml).....       | 8  |
| III - Les différentes variables Smarty.....         | 8  |
| III-A - Les variables simples.....                  | 8  |
| III-B - Les tableaux.....                           | 9  |
| III-B-1 - Les tableaux indexés.....                 | 9  |
| III-B-2 - Les tableaux associatifs.....             | 10 |
| III-B-3 - La fonction append().....                 | 11 |
| III-C - Les objets (syntaxe PHP5).....              | 11 |
| III-D - Doc Smarty.....                             | 12 |
| III-E - Les variables internes.....                 | 13 |
| IV - Les fonctions natives.....                     | 13 |
| IV-A - Section.....                                 | 13 |
| IV-B - If, else, elseif.....                        | 14 |
| IV-C - Literal.....                                 | 15 |
| IV-D - Include.....                                 | 16 |
| V - Les fonctions utilisateur.....                  | 16 |
| V-A - Html_options.....                             | 17 |
| V-B - Cycle.....                                    | 17 |
| V-C - Mailto.....                                   | 18 |
| V-D - Liste des fonctions utilisateur.....          | 18 |
| VI - Les modificateurs de variable.....             | 18 |
| VI-A - Upper.....                                   | 18 |
| VI-B - Default.....                                 | 19 |
| VI-C - Truncate.....                                | 19 |
| VII - La console de débogage.....                   | 20 |
| VII-A - Appel dans le script PHP.....               | 20 |
| VII-B - Appel dans le template.....                 | 20 |
| VII-C - La console sans javascript.....             | 20 |
| VII-D - Paramètre dans l'url.....                   | 21 |
| VIII - Recettes de code.....                        | 21 |
| VIII-A - Extraction Mysql vers tableau HTML.....    | 21 |
| VIII-B - Utilisation d'une source xml.....          | 22 |
| VIII-C - Production d'un fichier XML (fil rss)..... | 24 |
| IX - Ressources.....                                | 25 |
| X - Conclusion.....                                 | 25 |

## I - Introduction

### I-A - Remerciements

Tout d'abord merci à **Yogui**, responsable de la partie PHP chez DEVELOPPEZ.COM, qui a tout de suite répondu présent quand je l'ai contacté pour la rédaction de cet article et qui a largement contribué à l'amélioration du document d'origine.


Merci également à **GrandFather** et **titoumimi** pour leurs contributions.

Merci à mes collègues: Sylvain JAMES, Jean-Luc MICHEL et Jean PEYROUX qui m'ont supporté dans la rédaction de cet article.

### I-B - Présentation

Smarty est un système (ou moteur) de templates (entendez par template modèle ou patron) utilisable avec PHP 4 ou PHP 5.

Vous trouverez le site officiel [ici](#)

 *Dans cet article les exemples sont présentés en PHP5*

### I-C - Avantages et inconvénients

L'intérêt principal de Smarty réside dans la séparation du contenu et de la forme.

Le concept d'un système de template en général et de Smarty en particulier est de réserver les tâches de production des données à PHP et de mettre le code de présentation (HTML en l'occurrence) dans des 'templates' ou modèles, un fichier qu l'on suffixera dans nos exemples par '.TPL'

L'analyse des avantages et inconvénients de Smarty que je fais ci-après est le fruit des quelques expériences que j'ai eu avec ce moteur de templates. Je sais que tout le monde ne sera pas forcément d'accord avec les arguments avancés. Je vous encourage donc à mettre Smarty en oeuvre et vous faire vous-même une opinion... l'important est d'essayer pour se faire une idée.

#### I-C-1 - Inconvénients

L'utilisation du système de template Smarty n'est pas à la portée de tous, souvent le développeur non chevronné sera rebuté par son utilisation. Par exemple, il est fréquent de ne pas trouver immédiatement la façon d'accéder à sa variable dans le template (même avec un peu d'expérience d'ailleurs)... ce qui peut être un peu irritant.

Même avec une certaine expérience en PHP "traditionnel" (production et présentation PHP mélangés), l'utilisation de Smarty n'est pas évidente, elle remet en cause nombre de choses dans la manière de développer.

L'apprentissage du langage de templates (ce que l'on retrouvera dans nos fichiers .TPL) est indispensable. Cela pourra aussi en refroidir plus d'un. Ainsi il faudra connaître les variables (et les innombrables façon d'y accéder), les fonctions, le debugage. C'est ce que je me propose de vous expliquer.

## I-C-2 - Avantages

Le premier avantage que je vois à l'utilisation de Smarty est le gain de temps à moyen/long terme. Si sa mise en place peut prendre un peu plus de temps, plus on avance dans un projet et plus Smarty apparaît comme une évidence.

Le second avantage est la plus grande facilité de travailler à plusieurs. Prenez une équipe de développement avec des niveaux disparates (c'est le cas où je travaille). Ce découplage métier / présentation permet potentiellement à tout le monde de participer au développement bien plus facilement que dans un développement PHP plus traditionnel.

Un autre avantage qui m'est particulièrement cher (demandez donc à mes collègues !!!) est qu'avec Smarty au final le code produit est plus "propre" et "compréhensible" .. au final mieux organisé en particulier dans les gros projets. Bien sur il est possible de coder comme un cochon quand même !!!

Enfin, un système de cache permet d'accélérer considérablement la vitesse des traitements (la page n'est pas systématiquement recalculée). La décision d'utiliser la gestion de cache est à la charge du développeur.

Vous l'aurez compris, en ce qui me concerne, je suis convaincu que Smarty est un outil de PRODUCTIVITE.

## I-D - Installer

L'installation de Smarty nécessite peu de pré-requis:

- Un serveur web supportant PHP (apache ou IIS par exemple)
- Php version 4.0.6 ou version supérieure

Je vous encourage également à télécharger (ou consulter) la documentation de **Smarty** qui est très complète.

L'étape suivante est de télécharger les **sources de Smarty** et de les installer sur votre serveur (dans les exemples j'utilise le package **WAMP**). Attention l'archive contenant les sources de Smarty, téléchargeable sur ce site, est au format .tar.gz, procurez vous de quoi décompresser ce type d'archive (par exemple **7zip**).

Une fois l'archive de Smarty récupérée, vous constaterez la présence de plusieurs fichiers et répertoires:

En fait pour démarrer c'est simplement le répertoire **'libs'** qui nous intéresse. Il contient les fichiers suivants:



*Un conseil: placez les sources de Smarty dans un répertoire en ayant à l'esprit que vous pourrez utiliser ce système de templates dans plusieurs projets, à la racine de votre serveur web par exemple.*

## I-E - Tester

En premier lieu vous devez créer le fichier PHP qui aura pour objet de "piloter" Smarty (Inclusion de la librairie et instanciation de l'objet Smarty et affichage du template après compilation).

```
test.php
// Inclusion de la librairie
require_once ('../Smarty/Smarty.class.php');
// Instanciation d'un l'objet Smarty

$smarty = new Smarty();
// Affichage du template après compilation
```


test.php

```
$oSmarty->display('test.tpl');
```

L'étape suivante est de créer deux répertoires (dans les exemples, ces deux répertoires sont placés à la racine du répertoire contenant le fichier PHP):

- templates/
- templates\_c/

Ces deux répertoires contiendront respectivement les fichiers templates (.TPL) et les fichiers compilés.

 *La présence de ces deux répertoires est obligatoire pour l'utilisation de Smarty.*

Vous avez le choix pour l'emplacement des répertoires:

- **L'emplacement explicite:** Dans ce cas vous spécifiez vous-même l'emplacement de ces deux répertoires avec les deux propriétés de l'objet Smarty: "template\_dir" et "compile\_dir"
- **L'emplacement implicite:** il s'agit simplement de créer ces deux répertoires dans le répertoire du script PHP qui s'occupe des manipulations Smarty (c'est la méthode que nous retiendrons pour les exemples)

Pour terminer notre premier test, nous allons créer un fichier "test.tpl" dans le répertoire "templates/". C'est dans ce fichier que l'on retrouvera tout ce qui concerne l'affichage de notre page, à savoir le HTML (éventuellement JAVASCRIPT et STYLE) et les instructions propres à Smarty qui auront pour tâche d'afficher les données produites dans le fichier PHP. En voici le contenu:

test.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Test smarty</title>
</head>
<body>
  <h1>Test smarty</h1>
</body>
</html>
```

Pour la prévisualisation, lancez votre page "test.php". Vous devriez obtenir le résultat suivant:

### *Page de test*

Nous allons ensuite tester le passage d'une variable (produite dans le fichier PHP) au template Smarty, dans le but d'afficher une chaîne de caractères dans notre page HTML. Avant de nous lancer dans l'écriture du code sachez qu'il est nécessaire de respecter deux étapes:

- 1 Côté PHP: utilisation de la méthode assign() de l'objet Smarty pour affecter à la variable Smarty la valeur de la variable PHP
- 2 Côté TEMPLATE: Afficher la variable Smarty

test.php

```
<?php
// Inclure la librairie smarty
require_once('../smarty/Smarty.class.php');
```

**test.php**

```
// Instancier notre objet smarty
$smarty = new Smarty();

// Affecter la valeur "Bonjour le monde" à la variable SMARTY 'hello_world'
$smarty->assign('hello_world', 'Bonjour le monde');

// Provoque le rendu du template
$smarty->display('test.tpl');
?>
```

**test.tpl**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd"><html>
<head>
<title>Test smarty</title>
</head>
<body>
<h1>Test smarty</h1>
<!-- ici j'injecte la donnée qui vient de mon script PHP "{$hello_world}" -->
<h2 color="red">{$hello_world}</h2>
</body>
</html>
```


Revenons en à l'utilisation de la méthode **assign()** dans le fichier PHP. Cette méthode prend deux arguments:

- 1 Le **nom de la variable "cible"** que l'on utilisera dans le fichier template (c'est une chaîne de caractères)
- 2 La **variable ou la valeur PHP** que l'on souhaite associer au premier argument(variable ou valeur)

Voilà pour nos tests.

Maintenant nous allons faire un bref retour sur les métas-structures PHP (tableaux, objets, fichiers xml -avec simpleXML-) que nous utiliserons par la suite.

## II - Retour sur les métas-structures

 *La maîtrise des tableaux (initialisation et accès aux valeurs) me paraît un pré-requis indispensable pour utiliser Smarty dans de bonnes conditions.*

### II-A - Les tableaux

Nous allons revoir les deux principaux types de tableau que l'on utilise en PHP: les tableaux indexés et les tableaux associatifs (paire clé/valeur).

#### II-A-1 - Les tableaux indexés

L'utilisation la plus simple, à chaque valeur ajoutée au tableau est attribuée un index que l'on utilisera pour accéder à la valeur.

```
$aMonTableauIndexe = array('Sylvain', 'Jean-Luc', 'Jean', 'Eric');

echo $aMonTableauIndexe[2]; // affiche 'Jean'
```

## II-A-2 - Les tableaux associatifs

Pour ce qui est des tableau associatifs, la syntaxe est quelque peu différente puisque l'on doit associer à chaque clé une valeur.

```
$aMonTableauAsscoiatif = array(
    'Jean-Luc' => 'Michel',
    'Jean' => 'Peyroux',
    'Eric' => 'Pommereau',
    'Sylvain' => 'James'
);

echo $aMonTableauAsscoiatif['Jean']; // Affiche 'Peyroux'
```

## II-B - Les objets (syntaxe PHP5)

Voici un petit exemple d'une classe représentant une personne.

Nous allons voir, dans un premier temps, la définition de la classe, puis son utilisation.

A la fin du script j'utilise la fonction `print_r($objet)` pour afficher le contenu de l'objet.

```
class personne
{
    public $nom = "";
    public $prenom = "";
    public $aInfos = array();

    public function __construct($sNom, $sPrenom) {
        $this->nom = $sNom;
        $this->prenom = $sPrenom;
    }

    public function add_info($sInfo) {
        array_push($this->aInfos, $sInfo);
    }
}

$oPersonne = new personne('POMMEREAU', 'Eric');
$oPersonne->add_info('Fixe: 01.53.71.29.14');
$oPersonne->add_info('Adresse: 122, rue du Château des Rentiers 75013 PARIS');
$oPersonne->add_info('Date de naissance: 23/11/1973');

printf(
    '<pre>%s</pre>',
    print_r($oPersonne, 1)
);

/* AFFICHE *****
personne Object
(
    [nom] => POMMEREAU
    [prenom] => Eric
    [aInfos] => Array
        (
            [0] => Fixe: 01.53.71.29.14
            [1] => Adresse: 122, rue du Château des Rentiers 75013 PARIS
            [2] => Date de naissance: 23/11/1973
        )
)
*/
```

## II-C - Les fichiers xml (avec SimpleXml)

La librairie SimpleXML porte très bien son nom. Grâce à SimpleXML vous allez charger un fichier XML (provenant d'une chaîne de caractères ou d'un fichier) et accéder à ses valeurs en deux coups de cuillères à pot. Dans notre exemple nous allons charger un fichier XML et montrer comment accéder à une valeur de ce fichier.

### personnes.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<personnes>
  <personne>
    <nom>POMMEREAU</nom>
    <prenom>ERIC</prenom>
  </personne>
  <personne>
    <nom>JAMES</nom>
    <prenom>Sylvain</prenom>
  </personne>
  <personne>
    <nom>MICHEL</nom>
    <prenom>Jean-Luc</prenom>
  </personne>
</personnes>
```

Maintenant le fichier PHP que l'on utilise pour accéder à la valeur du fichier XML par l'intermédiaire de la librairie SimpleXML

### Le fichier PHP

```
<?php
    $oFichierXml = simplexml_load_file('personnes.xml');

    // Lecture et affichage du nom *****
    echo $oFichierXml->personne[0]->nom; // Affiche: 'POMMEREAU'
?>
```

## III - Les différentes variables Smarty

Nous allons voir dans ce chapitre comment passer des variables PHP au moteur Smarty et ensuite comment les utiliser dans les fichiers de templates. Le mécanisme d'affectation des variables n'est pas si simple... comme nous allons le voir.

### III-A - Les variables simples

On commence ici en douceur puisque l'on reprend le même principe que le test que l'on a fait (dans le chapitre 'tester'). Dans le fichier PHP, nous allons créer 2 variables (un entier et une chaîne de caractères). Ensuite il nous faut les "recenser" dans l'objet Smarty avec la méthode assign('variable\_cible\_smarty', \$variablePHP). Enfin on provoque l'affichage avec la méthode display('nom\_du\_template.tpl') de notre objet Smarty.

### Fichier exemple.php

```
// Instancier notre objet smarty
$oSmarty = new Smarty();

// Fixer les chemins de template (optionnel)
$oSmarty->template_dir = '../templates';
$oSmarty->compile_dir = '../templates_c';

// 1. Affectation des variables
$sune_chaine = "C'est génial smarty";
$un_entier = 33;

// 2. Recensement dans smarty
$oSmarty->assign('smarty_une_chaine', $sune_chaine);
```

#### Fichier exemple.php

```
$oSmarty->assign('smarty_un_entier', $un_entier);

// 3. Affichage du template après passage de l'objet
$oSmarty->display('exemple.tpl');
```

**!** La méthode `assign()` de la classe `Smarty` recense une variable PHP (second argument) et l'associe à une chaîne de caractères qui sera une variable Smarty dans le fichier de template. On peut être dérouté (croyez moi je l'ai été aussi!!) et faire des erreurs en pensant que le premier argument doit être une variable. Donc n'oubliez pas !!! Le premier argument d'`assign` est une chaîne de caractères.

#### exemple.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Test smarty</title>
</head>
<body>
<h1>Test smarty</h1>
<ul>
<li>une chaine: <span style="color:red;">{$smarty_une_chaine}</span></li>
<li>un entier: <span style="color:red;">{$smarty_un_entier}</span></li>
</ul>
</body>
</html>
```

**i** Pour les développeurs qui utilisent PHP4 on utilisera avantageusement la méthode `assign_by_ref()` qui évite la "recopie" des variables.

## III-B - Les tableaux

### III-B-1 - Les tableaux indexés

Si l'usage des variables simple est relativement simple, l'utilisation des tableaux s'avère un peu plus délicate... mais rassurez vous, rien d'insurmontable.

A noter que le principe pour le référencement des variables Smarty est exactement le même (ce sera le cas quelque soit le type de variable utilisé). C'est dans le fichier de template que l'on modifiera la syntaxe pour accéder à la variable souhaitée

#### tabIndexe.php

```
// Inclure la librairie smarty
require_once('../smarty/Smarty.class.php');
```

```
// Instancier notre objet smarty
$oSmarty = new Smarty();
```

```
// 1. Création et affectation d'un tableau indexé
$mon_tableau_indexé = array("Jean-Luc", "Jean", "Sylvain", "Eric");
```

```
// 2. Recensement dans smarty
$oSmarty->assign("smarty_mon_tableau_indexé", $mon_tableau_indexé);
```

```
// 3. Affichage du template après passage de l'objet
$oSmarty->display('tabIndexe.tpl');
```

#### tabIndexe.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

## tabIndexe.tpl

```

<title>Test smarty</title>
</head>
<body>
<h1>Test smarty</h1>
<ul>
<li>Une valeur du tableau indexé: {$smarty_mon_tableau_indexe[0]}</li>
<!-- Jean-Luc -->
<li>Une valeur du tableau indexé: {$smarty_mon_tableau_indexe[1]}</li>
<!-- Jean -->
<li>Une valeur du tableau indexé: {$smarty_mon_tableau_indexe[2]}</li>
<!-- Sylvain -->
<li>Une valeur du tableau indexé: {$smarty_mon_tableau_indexe[3]}</li>
<!-- Eric -->
</ul>
</body>
</html>
    
```

## III-B-2 - Les tableaux associatifs

Même principe que ce que l'on a vu précédemment, l'accès aux valeurs est un peu différent de la méthode utilisée pour les tableaux indexés un exemple:

## tabAssoc.php

```

// Inclure la librairie smarty
require_once ('../smarty/Smarty.class.php');

// Instancier l'objet smarty
$smarty = new Smarty();

// 1. Création et affectation d'un tableau associatif
$monTableauAssocatif = array(
    "Eric" => "Pommereau",
    "Jl" => "Michel",
    "Jean" => "Peyroux"
);

// 2. Recensement dans smarty
$smarty->assign("smarty_mon_tableau_assoc", $monTableauAssocatif);

// 3. Affichage du template après passage de l'objet
$smarty->display("tabAssoc.tpl");
    
```

## tabAssoc.tpl

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Test smarty</title>
</head>
<body>
<h1>Test smarty</h1>
<ul>
<li>Une valeur du tableau associatif: {$smarty_mon_tableau_assoc.Eric}</li>
<!-- Pommereau -->
<li>Une valeur du tableau associatif: {$smarty_mon_tableau_assoc.Jl}</li>
<!-- Michel -->
<li>Une valeur du tableau associatif: {$smarty_mon_tableau_assoc.Jean}</li>
<!-- Peyroux -->
</ul>
</body>
</html>
    
```

### III-B-3 - La fonction append()

Une alternative à l'utilisation d'un tableau intermédiaire (comme je l'ai fait dans les deux exemples précédents): l'utilisation de la méthode append().

Une autre utilité cette méthode est la possibilité d'ajouter à la même variable Smarty du contenu à plusieurs moments du script, ce qui est exclu avec la méthode assign().

#### exemple 1 (tableau indexé)

```
// selection de 5 enregistrements (à partir de 0)
$query = "SELECT PSN_FIRST_NAME, PSN_LAST_NAME, PSN_PHONE
FROM person ORDER BY PSN_LAST_NAME LIMIT 0,5";

$smarty_rs = mysql_query($query, $mysql_ressource) or die(mysql_error());

while ($aRow = mysql_fetch_array($smarty_rs)) {
    $smarty->append('smarty_table_mysql', $aRow);
}

/*
...
INSTRUCTIONS
...
*/

// selection de 5 enregistrements (à partir de 5)
$query = "SELECT PSN_FIRST_NAME, PSN_LAST_NAME, PSN_PHONE
FROM person ORDER BY PSN_LAST_NAME LIMIT 5,5";

$smarty_rs = mysql_query($query, $mysql_ressource) or die(mysql_error());

while ($aRow = mysql_fetch_array($smarty_rs)) {
    $smarty->append('smarty_table_mysql', $aRow);
}
```

Attention cependant, utiliser la méthode append() ne permet pas le debugage côté PHP (par exemple avec la fonction print\_r()) puisqu'il n'y a pas de variable intermédiaire dans le script PHP.

### III-C - Les objets (syntaxe PHP5)

Reprenons l'exemple (définition de la classe et utilisation de l'objet) vu au dessus, et

#### exempleClass.php

```
// Inclure la librairie smarty
require_once('../smarty/Smarty.class.php');

// Définition de la classe
class personne
{
    public $nom = "";
    public $prenom = "";
    public $aInfos = array();

    public function __construct($sNom, $sPrenom) {
        $this->nom = $sNom;
        $this->prenom = $sPrenom;
    }

    public function add_info($sInfo) {
        array_push($this->aInfos, $sInfo);
    }
}

// Instancier notre objet smarty
```

**exempleClass.php**

```

$Smarty = new Smarty();

// 1. Création et affectation d'un objet
$Personne = new personne("POMMEREAU", "Eric");
$Personne->add_info("mobile: 06.11.75.86.xx");
$Personne->add_info("Adresse: 4, villa des Marronniers 91580 ETRECHY");
$Personne->add_info("Date de naissance: 23/11/1973");

// 2. Recensement dans smarty
$Smarty->assign("smarty_objet_personne", $Personne);

// 3. Affichage du template après passage de l'objet
$Smarty->display("exempleClass.tpl");
    
```

Maintenant voyons comment "récupérer nos petits" dans notre fichier de template:

**exempleClass.tpl**

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Test smarty</title>
</head>
<body>
<h1>Test smarty</h1>
<ul>
<li>Propriété "nom" de l'objet: {$smarty_objet_personne->nom}</li>
<!-- Affiche "Pommereau" -->

<li>Propriété "aInfos" (première valeur du tableau) de l'objet:
{$smarty_objet_personne->aInfos[0]}
</li>
<!-- Affiche "mobile: 01.213.65.32" -->
</ul>
</body>
</html>
    
```

## III-D - Doc Smarty

Vous trouverez dans la liste suivante, provenant tout droit de la **documentation Smarty**, qui expose les diverses façon d'accéder à des variables dans les templates Smarty:

```

{$foo}          <-- affiche une variable simple (qui n'est pas un tableau ou un objet)
{$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]}     <-- syntaxe uniquement valide dans une section de boucle, voir {section}
    
```


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
    
```

## III-E - Les variables internes

Smarty offre, en plus du mécanisme "manuel" d'affectation de variable, la possibilité d'accéder à certaines variables réservées accessible par l'intermédiaire de la variable "smarty". Il s'agit en fait de l'enrobage des variables PHP (\$\_SERVER ou \$\_ENV par exemple). Cette fois encore je reprends la doc Smarty pour afficher quelque unes de ces variables

 *Pour accéder aux valeurs de ces variables internes vous n'avez rien à faire côté PHP, tout est directement accessible dans le fichier de template. L'accès se fait par la variable "\$smarty" qui est automatiquement créée.*

### Quelques variables internes de Smarty

```
{* 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) *}
{$smarty.post.page}


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

{* affiche la valeur du cookie "utilisateur" *}
{$smarty.cookies.utilisateur}

{* affiche la variable serveur "SERVER_NAME" *}
{$smarty.server.SERVER_NAME}
```

## IV - Les fonctions natives

La compréhension des fonctions Smarty est indispensable, en effet c'est grâce aux fonctions que vous allez pouvoir répéter une portion de code, insérer du code en fonction de certaines conditions, insérer du javascript dans vos templates et bien d'autres choses encore.

 *Les fonctions s'utilisent entre accolage ex: {section}. Comme en html avec les balises, toute déclaration de fonction s'accompagne d'une balise de "fermeture". Pour la fonction {section} on utilisera par exemple {/section}.*

### IV-A - Section

Cette fonction essentielle est peut-être la première qui m'a fait prendre conscience du potentiel de Smarty.

Section vous permet d'afficher une portion de code HTML autant de fois qu'il y a d'éléments dans le conteneur qui lui a été indiqué. C'est en fait strictement la même chose qu'une boucle, gardez toujours ça à l'esprit. Pour que tout cela fonctionne, il faut donc avoir passé à Smarty un conteneur de donnée (tableau ou objet) et lui se charge du reste.

Vous devez spécifier au moins deux arguments: LOOP qui est la variable Smarty (tableau ou objet assigné en PHP) et un attribut NAME dont la valeur représente le nom de la SECTION. La valeur de NAME est comme un index que nous utiliserons dans la fonction pour accéder à l'élément courant, aller un exemple:

```
section.php
// Inclure la librairie smarty
require_once('../smarty/Smarty.class.php');

// Instancier l'objet smarty
$oSmarty = new Smarty();

$aMonTableau = array("Jean", "Jean-Luc", "Sylvain", "Eric");

// 2. Recensement dans smarty de la variable $aMonTableau
```

**section.php**

```

$Smarty->assign('smarty_tableau', $aMonTableau);

// 3. Affichage du template après passage de l'objet
$Smarty->display('section.tpl');
    
```

Jusqu'ici, rien de particulier, la nouveauté se trouve dans le template:

**section.tpl**

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>La fonction section</title>
</head>
<body>
<h2>La fonction section</h2>
<ul>
<!--Bloc répété autant de fois qu'il y aura d'éléments dans '$smarty_tableau'-->
{section name=idx_smarty_tableau loop=$smarty_tableau}
<li>{$smarty_tableau[idx_smarty_tableau]}</li>
{/section}
</ul>
</body>
</html>
    
```

## IV-B - If, else, elseif

Tout comme en PHP, Smarty offre dans ses templates la possibilité d'afficher du contenu en fonction de certaines conditions définies par le développeur (ou le graphiste) dans le fichier de template. Prenons l'exemple de la gestion d'un profil "administrateur" où certains éléments ne doivent pas être affichés dans certains cas. On pourra tester l'existence ou la valeur d'une variable pour déterminer que la portion de code concernée doit ou ne doit pas s'afficher

On déclarera la fonction "IF" de cette façon {if ma\_condition}mon code...{/if}. Dans le cas du {IF}, l'argument (ma\_condition) est la condition que l'on souhaite évaluer (égalité, différence, négation...). Un exemple:

**if.php**

```

// Inclure la librairie smarty
require_once('../smarty/Smarty.class.php');

// Instancier notre objet smarty
$Smarty = new Smarty();

$userIsAdmin = 'true';

// 2. Recensement dans smarty
$Smarty->assign('smarty_userIsAdmin', $userIsAdmin);

// 3. Affichage du template après passage de l'objet
$Smarty->display('if.tpl');
    
```

Ici encore rien de particulier côté PHP, c'est la partie TEMPLATE qui est concernée, dans cet exemple, on teste la valeur d'une variable Smarty concernant les droits d'un utilisateur.

**if.tpl**

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Fonctions IF/ELSE/ELSEIF</title>
</head>
<body>
    
```

**if.tpl**

```

<h2>Les fonctions IF/ELSE/ELSEIF</h2>

{if $smarty_userIsAdmin == 'true'}
  <div>Salut administrateur</div>
{else}
  <div>Salut utilisateur lambda !!</div>
{/if}

</body>
</html>
    
```

## IV-C - Literal

Literal vous permet de désactiver un temps l'interprétation Smarty. Cela permet notamment l'insertion de scripts JAVASCRIPT qui génèrent habituellement des erreurs (entre autre à cause des accolades).

En effet le fichier de template suivant génère une erreur:

**sansLiteral.tpl**

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title></title>
</head>
<body>
  <script language="JavaScript" type="text/javascript">
    function test () {
      alert('Bonjour le monde !!');
    }
  </script>
</body>
</html>
    
```

**Fatal error: Smarty error: [in index.tpl line 10]: syntax error: unrecognized tag: alert('Bonjour le monde !!'); (Smarty\_Compiler.class.php, line 439) in C:\Program Files\wamp\www\demos\smarty\lib\_smarty\Smarty.class.php on line 1095**

En utilisant la fonction {LITERAL}, c'est à dire en entourant le script javascript ou la C.S.S. concerné, cela ne posera plus de problème:

**avecLiteral.tpl**

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Literal</title>
</head>
<body>
  {LITERAL}
  <script language="JavaScript" type="text/javascript">
    function test () {
      alert('Bonjour le monde !!');
    }
  </script>
  {/LITERAL}
</body>
</html>
    
```

## IV-D - Include

Cette fonction permet d'appeler dans un template un autre template, dans la même idée qu'un include() PHP. Cela permet de découpler encore plus la présentation. Nous allons voir dans l'exemple suivant le cas d'une zone principale et d'un "HEADER".

### include.php

```
require_once('../smarty/Smarty.class.php');

// Instancier l'objet smarty
$smarty = new Smarty();

$smarty->assign('header_text', "Ceci est l'entête de ma page !!!");
$smarty->assign('main_text', "Ceci est la zone principale de ma page !!!");

// 3. Affichage du template après passage de l'objet
$smarty->display('include.tpl');
```

Le fichier de template que l'on va inclure:

### header.tpl

```
<div id="header" style="background-color:teal;">
  <h1>{$header_text}</h1>
</div>
```

Voyons maintenant le template qui prend en charge l'inclusion (ici du template header.tpl)

### include.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Exemple d'include</title>
</head>
<body>
  {include file="header.tpl"}
  <div id="main" style="background-color:#6699cc;">
    <h1>{$main_text}</h1>
  </div>
</body>
</html>
```



*Dans le template inclu on peut utiliser sans problème les variables Smarty.*

## V - Les fonctions utilisateur

Pourquoi fonctions utilisateur ? Eh bien c'est une des caractéristiques et des forces de Smarty. Les fonctions utilisateurs ont le même principe de fonctionnement que les fonctions natives, un mot clé, des arguments, le tout entre accolades.

Quelle différence alors ? Ces fonctions ne font pas partie du coeur de Smarty mais utilisées comme PLUGIN. D'ailleurs vous verrez dans les sources de Smarty un répertoire /plugin. Si vous regardez bien vous retrouvez un fichier par fonction utilisateur.

## V-A - Html\_options

La fonction HTML\_OPTIONS provoque l'affichage d'une liste d'options (à l'intérieur d'un élément HTML "SELECT"). Il faut pour cela lui indiquer une liste de valeurs qui seront affichées et une liste d'identifiants associés à ces valeurs... vite un exemple !!!

### html\_options.php

```
$oSmarty = new Smarty();

// Liste des textes pour les "<option>"
$aNoms = array("Sylvain", "Jean", "Jean-Luc", "Eric");
// Liste des valeurs pour les "<option>"
$aIndex = array(1,2,3,4);
// L'élément sélectionné est le 3
$iSelected = 3;

$oSmarty->assign('smarty_liste_noms', $aNoms);
$oSmarty->assign('smarty_liste_index', $aIndex);
$oSmarty->assign('smarty_liste_selected', $iSelected);

$oSmarty->display('html_options.tpl');
```

### html\_options.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>exemple HTMLOPTIONS</title>
</head>
<body>
<form name="mon_formulaire">
<select name="gars_dev">
{html_options values=$smarty_liste_index selected=$smarty_liste_selected output=$smarty_liste_noms}
</select>
</form>
</body>
</html>
```

### Résultat

## V-B - Cycle

La fonction CYCLE permet l'affichage en alternance de 1 à n valeurs. Dans la pratique, on l'utilisera avantageusement pour l'alternat de couleurs dans un tableau affichant un ensemble de données. C'est ce que nous allons voir tout de suite dans l'exemple

### cycle.php

```
require_once('../lib_smarty/Smarty.class.php');
  


```
$oSmarty = new Smarty();

$aNoms = array("Sylvain", "Jean", "Jean-Luc", "Eric");

$oSmarty->assign('smarty_liste_noms', $aNoms);

$oSmarty->display('cycle.tpl');
```


```

### cycle.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

### cycle.tpl

```
<title>Exemple fonction cycle</title>
</head>
<body>
  {section name=idx loop=$smarty_liste_noms}
  <div style="background-color:{cycle values="red,blue"};">{$smarty_liste_noms[idx]}</div>
  {/section}
</body>
</html>
```

## V-C - Mailto

Comme son nom l'indique cette fonction vous permet d'ajouter rapidement un lien mailto (lien vers un ou plusieurs mails). L'utilisation de mailto permet notamment l'encodage en javascript / hexadécimal des adresses mails qui sont alors bien moins facilement repérables.

Un exemple d' utilisation en encodage javascript:

### encode.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Exemple fonction encode</title>
</head>
<body>
  {mailto address="eric190@free.fr" encode="javascript"}
</body>
</html>
```

## V-D - Liste des fonctions utilisateur

```
capture
config_load
foreach,foreachelse
include
include_php
insert
if,elseif,else
ldelim,rdelim
literal
php
section, sectionelse
strip
```

## VI - Les modificateurs de variable

Les modificateurs de variable permettent de modifier le contenu des variables passées à SMARTY. Grâce aux modificateurs le développeur pourra intervenir sur le contenu, l'apparence des variables. Les modificateurs sont nombreux et relativement simples à mettre en oeuvre. Le modificateur de variable s'applique comme un filtre en mettant un "pipe" après la variable

### VI-A - Upper

UPPER permet de mettre le contenu de la variable en capitales.

```
// Inclure la librairie smarty
```

```

require_once('../smarty/Smarty.class.php');

// Instancier notre objet smarty
$smarty = new Smarty();

$smarty->assign("smarty_phrase", "Les chaussettes de l'archi-duchesse");

// 3. Affichage du template après passage de l'objet
$smarty->display('exemple.tpl');
    
```

### Modificateur UPPER

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Exemple modificateur UPPER</title>
</head>
<body>
<p>{$smarty_phrase|upper}</p>
</body>
</html>
    
```

### Résultat

LES CHAUSSETTES DE L'ARCHI-DUCHESSSE

## VI-B - Default

Ce modificateur permet de donner à une variable Smarty une valeur par défaut. Par exemple pour insérer un espace dans un tableaux quand la donnée est vide.

### Modificateur DEFAULT

```


<table>
<tr>
<td>{$smarty_nom|default:"&nbsp;"}</td>
<td>{$smarty_prenom|default:"&nbsp;"}</td>
</tr>
</table>
    
```

## VI-C - Truncate

Truncate entraîne la sésure d'une chaîne de caractères au nombre de caractères souhaité

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Modificateur TRUCATE</title>
</head>
<body>
<p>{$smarty_phrase|truncate:30}</p>
</body>
</html>
    
```

 **Par défaut, le découpage ne se fait normalement pas sur un mot mais su la fin du mot, issu de la doc Smarty: "Par défaut, truncate tentera de couper la chaîne à la fin d'un mot". Cette caractéristique est facilement "débrayable".**

## VII - La console de débogage

La console de débogage permet au développeur notamment de voir les templates appelés, les variables Smarty et leurs contenu ainsi que les variables de configuration. C'est un formidable outil que je vous encourage vivement à utiliser.

Dans son utilisation "de base", l'appel à la console se fait dans le script PHP, ce qui provoque au rechargement de page l'apparition d'une fenêtre "popup" (attention aux bloqueurs de popup). L'affichage de la console est donc effectué en javascript.

### VII-A - Appel dans le script PHP

Première possibilité, l'appel dans le script PHP qui implémente Smarty. Dans ce cas de figure, on fixe une propriété debugging à true.

```
console.php
// Inclure la librairie smarty
require_once ('../smarty/Smarty.class.php');

// Instancier notre objet smarty
$smarty = new Smarty();

$smarty->debugging = true;

$smarty->assign("smarty_phrase", "Les chaussettes de l'archi-duchesse");

// 3. Affichage du template après passage de l'objet
$smarty->display('exemple.tpl');
```

### VII-B - Appel dans le template

Une autre façon d'appeler la console consiste à insérer une fonction {DEBUG} dans le template

```
consoleInTemplate.tpl
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Exemple d'utilisation de la console de débogage</title>
</head>
<body>
<p>{$smarty_phrase|truncate:30}</p>
{debug}
</body>
</html>
```

### VII-C - La console sans javascript

Une fonction très intéressante: l'affichage de la console directement dans le fichier de sortie sans passer par javascript, il s'agit d'un paramètre (OUTPUT) à ajouter dans la fonction {debug}:

```
consoleEnHtml.tpl
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>La console sans javascript</title>
```

#### consoleEnHtml.tpl

```

</head>
<body>
  <p>{$smarty_phrase|truncate:30}</p>
  {debug output="html"}
</body>
</html>
    
```

#### Affichage de la console sans passer par JAVASCRIPT

Comme vous le voyez la console n'est plus lancée dans une fenêtre popup mais bien intégrée à la page web générée

### VII-D - Paramètre dans l'url

Dernière fonctionnalité que nous allons voir, c'est un must !!! la console est 'appelable' en passant un paramètre dans l'url qui invoque le script php. Pour se faire il faut permettre à Smarty de le faire en fixant la propriété 'debugging\_ctrl' à 'URL' comme ceci:

#### consoleUrl.php

```

$smarty = new Smarty();

$smarty->debugging_ctrl= 'URL';
    
```

#### Appel de la console en utilisant un paramètre dans l'URL

Voyez dans la barre d'adresse, j'ai passé le paramètre 'SMARTY\_DEBUG' (peu importe qu'il y ait une valeur associée), la propriété Smarty "debugging\_ctrl" étant à true, la console s'est affichée.

### VIII - Recettes de code

L'objet de cette partie du tutoriel est de vous présenter quelques cas concrets de ce que l'on peut faire et comment on peut le faire avec SMARTY.

#### VIII-A - Extraction Mysql vers tableau HTML

Voilà un exemple que l'on retrouve dans la plupart des applications, partir d'un ensemble de données (une table Mysql dans notre exemple) pour afficher un tableau

Un screenshot de ma table visualisée avec phpMyAdmin (base training, table person)

La partie PHP ne présente rien d'exceptionnel. L'idée est d'empiler chacune des "lignes" récupérées (avec la fonction `mysql_fetch_array`) pour réutiliser le tableau ainsi constitué dans le template Smarty.

#### fromMysqlToTable.php

```

require_once('../lib_smarty/Smarty.class.php');

$smarty = new Smarty();

mysql_ressource = mysql_connect('localhost', 'root', '');

mysql_select_db("training", $mysql_ressource);

$query = "SELECT PSN_FIRST_NAME, PSN_LAST_NAME, PSN_PHONE FROM person ORDER BY PSN_LAST_NAME";

mysql_rs = mysql_query($query, $mysql_ressource) or die(mysql_error());
    
```

**fromMysqlToTable.php**

```

$aRecordSet = array();

while ($aRow = mysql_fetch_array($mysql_rs)) {
    array_push($aRecordSet, $aRow);
}

$smarty->assign('smarty_table_mysql', $aRecordSet);

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

**mysqlToTable.tpl**

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title></title>
</head>
<body>
<table width="450px">
<tr style="background-color:teal;color:white;">
<th>NOM</th>
<th>PRENOM</th>
<th>TELEPHONE</th>
</tr>
<!-- Pour chaque élément du tableau $smarty_table_mysql -->
<section name=idx loop=$smarty_table_mysql>
<tr style="background-color:{cycle values="#ffffcc, #cccccc"};">
<td>{$smarty_table_mysql[idx].PSN_LAST_NAME}</td>
<td>{$smarty_table_mysql[idx].PSN_FIRST_NAME}</td>
<td>{$smarty_table_mysql[idx].PSN_PHONE}</td>
</tr>
</section>
</table>
</body>
</html>
    
```

*Résultat: un tableau HTML avec alternat de couleur*

## VIII-B - Utilisation d'une source xml

Nous allons voir comment, à partir d'un fichier XML (source fichier ou bien chaîne de caractères) on peut mettre ces données dans notre template.

**personnes.xml**

```

<?xml version="1.0"?>
<personnes>
<personne>
<nom>PEYROUX</nom>
<prenom>Jean</prenom>
<tels>
<tel>06.76.89.78.56</tel>
</tels>
</personne>
<personne>
<nom>MICHEL</nom>
<prenom>Jean-Luc</prenom>
<tels>
<tel>01.87.54.23.34</tel>
<tel>06.12.21.34.54</tel>
<tel>09.09.98.89.09</tel>
</tels>
</personne>
</personnes>
    
```

## personnes.xml

```

<nom>POMMEREAU</nom>
<prenom>Eric</prenom>
<tels>
  <tel>01.78.98.87.87</tel>
  <tel>06.12.34.32.34</tel>
</tels>
</personne>
</personnes>
    
```

Comme nous allons le voir, la taille du code PHP est particulièrement faible.

## personnes.php

```

require_once ('../lib_smarty/Smarty.class.php');

$smarty = new Smarty();

$xmlFile = simplexml_load_file("personnes.xml");

$smarty->assign('smarty_xml_file', $xmlFile);

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

Le fichier de template est un peu plus complexe puisque nous utilisons la fonction foreach (un peu différente de {SECTION}), cette fonction est de plus imbriquée pour obtenir les téléphones de chacun:

## personnes.tpl

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Utilisation d'une source XML</title>
</head>
<body>
  <table width="450px">
    <tr style="background-color:teal;color:white;">
      <th>NOM</th>
      <th>PRENOM</th>
      <th>TELEPHONES</th>
    </tr>
    <!-- pour chaque item personne -->
    {foreach from=$smarty_xml_file item=personne}
      <tr style="background-color:{cycle values="#ffffcc, #cccccc"};">
        <td>{$personne->nom}</td>
        <td>{$personne->prenom}</td>
        <td>
          <table>
            <!-- pour chaque item telephone -->
            {foreach from=$personne->tels->tel item=telephone}
              <tr>
                <td>
                  {$telephone}
                </td>
              </tr>
            {/foreach}
          </table>
        </td>
      </tr>
    {/foreach}
  </table>
</body>
</html>
    
```

## VIII-C - Production d'un fichier XML (fil rss)

Nous avons vu comment utiliser avec Smarty une source XML.

Nous allons maintenant voir comment produire du XML. Plus particulièrement dans cet exemple nous allons produire un fil RSS 2.0.

Pour réaliser ce fil RSS, nous allons utiliser une source mysql dont voici un aperçu:

Pour la suite même logique que ce que nous avons pu voir précédemment. Le fichier PHP chargé de la récupération des données et de la "ventilation" vers Smarty.

```
rss.php
<?php

require_once('../lib_smarty/Smarty.class.php');

$smarty = new Smarty();

// Le tableau conteneur des éléments RSS
$rssItems = array();

// Se connecter à mysql
if (! $connexion = mysql_connect('localhost', 'root', ''))
    die("Impossible de se connecter");

// Choisir la base de données
if (! mysql_select_db('demos'))
    die("Erreur de selection de la base de données");

// Construire la requête
$sql = "SELECT `title`, `description`, `link` FROM rss" ;

// Exécuter la requête et récupérer le jeu d'enregistrements
if (! $mysqlRes = mysql_query($sql))
    die("Erreur: la requête n'a pu être exécutée");

// Pour chaque ligne du jeu d'enregistrement
while ($row = mysql_fetch_array($mysqlRes, MYSQL_ASSOC)) {
    // Ajouter dans le conteneur
    array_push($rssItems, $row);
}

// Passer la
$smarty->assign('smarty_rssItems', $rssItems);

header("Content-Type: text/xml");

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

Enfin, le fichier de template est un fichier XML standard RSS 2.0 **Standard RSS** avec une partie "dynamique" reproduite autant de fois qu'il y a d'éléments dans la base de donnée.

```
rss.tpl
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="0.91">
  <channel>
    <title>Exemple de production de fil RSS avec SMARTY</title>
    <link>http://localhost/demos/init_smarty/demoRSS/</link>
```

## rss.tpl

```
<description>La démo suivant vous montre comment créer facilement un flux RSS avec
SMARTY.</description>
<language>fr</language>
<copyright>Eric POMMEREAU</copyright>
<webMaster>eric-pommereau@developpez.com</webMaster>

<image>
  <title>Smarty</title>
  <url>http://localhost/demos/init_smarty/demoRSS/smarty_icon.gif</url>
  <link>http://smarty.php.net/</link>
  <width>88</width>
  <height>31</height>
</image>

{section name=itemIndex loop=$smarty_RssItems}
  <item>
    <title>{$smarty_RssItems[itemIndex].title}</title>
    <description>{$smarty_RssItems[itemIndex].description}</description>
    <link>{$smarty_RssItems[itemIndex].link}</link>
  </item>
{/section}

</channel>
</rss>
```





Le Fil RSS tel qu'il apparaît dans le navigateur FIREFOX

## IX - Ressources

### Smarty

- Le site de Smarty :  [Smarty](#)
- Documentation :  [La doc Smarty](#)
- Les ressources (article, plugins et autres applications utilisant Smarty) :  [ressources Smarty](#)
- Un seul livre (pour le moment...), il est en anglais :  [Smarty : PHP Template Programming and Applications](#)

### Tutoriels

- Articles concernant les gabarits sur  [developpez.com](#)
- Tutoriel :  [Comparatif des systèmes de template pour PHP](#)
- Tutoriel :  [Découverte des principaux moteurs de template en PHP](#)
- Tutoriel :  [Conception avancée d'une galerie d'images générée à la volée](#)

### Forums

- Forums :  [Général sur developpez.com](#)
- Forums :  [Partie PHP sur developpez.com](#)
- Forums :  [Partie bibliothèques PHP sur developpez.com](#)

## X - Conclusion



*Un petit conseil pour la route: si vous ne vous êtes jamais servi de Smarty ne reprenez pas un projet "du type traditionnel" existant ... ça peut vite tourner au cauchemar. Je préconise plutôt un démarrage en douceur sur un projet tout neuf.*

Voilà... c'est terminé. J'espère que vous avez eu plaisir à me lire, si j'ai su susciter quelques vocation quand à l'utilisation de Smarty dans vos projets... alors mon objectif est atteint !!!

Pour ma part je suis convaincu que Smarty est un bon système (peut-être pas le meilleur...) pour créer, maintenir et faire évoluer des applications en PHP. N'hésitez pas à me contacter si vous avez des questions, des suggestion ou pour signaler une erreur.

Contact: [eric.pommereau@caramail.com](mailto:eric.pommereau@caramail.com)