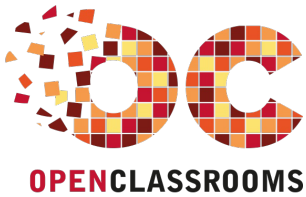
Utilisation d'un moteur de templates : Talus' TPL

Par Baptiste Clavié (Talus)
et vincent1870



www.openclassrooms.com

Dernière mise à jour le 2/11/2012

Sommaire

Utilisation d'un moteur de templates : Talus' TPL

Par  vincent1870 et  Baptiste Clavié (Talus)

Mise à jour : 02/11/2012
Difficulté : Facile Durée d'étude : 5 heures

Bonjour à tous.

Si vous venez ici, c'est par envie d'en savoir plus sur les moteurs de templates, et en particulier sur l'un d'entre eux : Talus' TPL. Comme vous l'avez sans doute remarqué, il existe déjà des tutos sur d'autres moteur de templates : Twig (voir le tuto) ou encore Smarty (voir le tuto). Ce tuto n'a pas pour but de leur faire de la concurrence mais de vous initier à un autre moteur existant. Il utilise une syntaxe de type XML.

Pré-requis :

- avoir lu les cours de M@teo21 en entier, toujours mieux pour avoir de bonnes bases ;
- avoir PHP 5.1 (ou plus) installé sur son serveur ;
- un petit plus : avoir lu le tutoriel de vyk12 sur la POO en PHP.

Sur ce, je vous souhaite une bonne lecture à la découverte de ce script.
Sommaire du tutoriel :

Téléchargement et découverte

Eh, attends : c'est quoi, un moteur de templates ?

Ah, j'allais oublier. Je vais donc brièvement vous présenter l'utilité d'un tel script.

Rappels / notions concernant un moteur de templates

En général, vous avez souvent un seul fichier par page, qui contient aussi bien le code d'affichage que les requêtes à MySQL notamment, comme ceci :

Code : PHP - Script de news basique

```
<h1>Les news du site</h1>

<?php
```

```php
// $pdo : instance de PDO, initialisée par vos soins
$q = $pdo->query("SELECT id, auteur, titre, DATE_FORMAT(date,
'%d/%m/%Y %Hh%i') AS date_formatee, contenu
FROM news
ORDER BY date DESC");
foreach($q as &$data){
    echo '
<div class="news">
<h2>' . $data['titre'] . '</h2>
<p>News postée le '.str_replace(' ', ' à ',
$data['date_formatee']).' par ' . $data['auteur'] . '</p>
<p>'.$data['contenu'].'</p>
</div>';
}

$q = null;
```

Un moteur de templates vous permet de séparer ce fichier en deux. L'un est chargé de l'affichage du code xHTML, l'autre d'effectuer les requêtes à MySQL, et d'appeler le second fichier.

Vous lirez souvent *"utiliser PHP comme moteur de templates"*. Cela signifie que le second fichier contient alors du code PHP, et qu'il est appelé par un `include`... et est constitué à 90% d'`echo`, et autres `if` / `foreach`.

Ce que nous allons voir ici consiste à remplacer le code PHP par un pseudo-langage, présentant les avantages suivants.

- Ce langage est assez simple. On retrouve l'équivalent d'expressions PHP. Cela rend ce langage idéal pour être utilisé par un graphiste, qui n'a ainsi pas à apprendre à utiliser PHP, mais juste ce langage, forcément plus simple.
- Nous en venons donc à un deuxième avantage : la présentation est totalement séparée de la logique du code, et de la récupération des données. On peut toucher au style du site sans toucher à la logique, et vice-versa.

Mais bien entendu, il y a des inconvénients à utiliser un tel système.

- Le script pourra être plus long à charger. Néanmoins, le moteur que je vais vous présenter reste très léger.
- Un autre langage que PHP doit évidemment être appris.

Notez enfin qu'un tel moteur s'interface très bien avec un modèle MVC (cet autre tutoriel pourra vous en apprendre plus).

Présentation et téléchargement de Talus' TPL

Talus'TPL est un moteur portant le nom de son concepteur, Talus. Il est disponible depuis le 2 février 2008, date de sortie de sa version 1.0.0. Il a depuis d'autres versions. La dernière est la 1.12. C'est avec celle-ci que nous allons travailler. Ce que je vous apprendrai ici sera valable avec toutes les versions après celle-ci, sauf qu'il est possible que quelques fonctionnalités récentes ne soient pas évoquées. J'essaierai de maintenir le tutoriel à jour autant que possible.

Commencez donc par télécharger le script sur son forum officiel, rubrique **Releases**. Prenez la version la plus récente (je rappelle qu'au moment de l'écriture, la version la plus récente était la 1.12). Dans le premier message du sujet, vous trouverez les liens pour télécharger le script en **.tgz** ou en **.zip**. Pensez à revenir régulièrement pour vous tenir à jour.

Ensuite, décompressez l'archive et mettez-la sur votre site avec votre logiciel FTP favori. Vous pouvez mettre les fichiers à la racine ou bien dans un sous-dossier. Notez le dossier "lib" à la racine de l'archive : c'est celui-ci qui nous intéresse. Libre à vous de le renommer comme bon vous semble (comme **includes**, **libs**, etc.).

Aucune installation n'est nécessaire, il suffira d'inclure le script PHP principal contenu dans l'archive (soit, par défaut, **/lib/Talus_TPL/Engine.php**) dans vos pages PHP. Les autres fichiers .php contenus dans le répertoire **lib/Talus_TPL/** seront inclus automatiquement suivant les besoins du moteur... et vous retrouverez, à la racine de l'archive, des fichiers d'informations tels que le *changelog* ou encore la licence du moteur.

Notions de base

Nous allons donc commencer par voir comment utiliser les fonctions principales de Talus' TPL. La première chose à faire est **d'instancier un objet de type `Talus_TPL`**. Ce vocabulaire est celui de la Programmation Orientée Objet (ou POO pour les intimes). Je vous expliquerai rapidement tout ce qui est utile ici à ce propos.

Création de l'objet

Voici le code *minimal* permettant de créer l'objet `Talus_TPL` évoqué plus haut :

Code : PHP

```php
<?php $tpl = new Talus_TPL_Engine($dossier_templates,
$dossier_cache); ?>
```

En POO, l'opérateur `new` est indispensable pour créer une instance de l'objet. Pour définir les deux dossiers, j'ai personnellement recours à la fonction dirname. Souvent, on se demande par rapport à quoi les chemins doivent être définis (dans un include situé dans un fichier déjà inclus lui-même, par exemple). Ce bout de code affiche le chemin complet du fichier :

Code : PHP

```php
<?php echo dirname(__FILE__); ?>
```

`__FILE__` est une constante contenant automatiquement le nom du fichier courant. Alors, pour inclure un fichier situé dans un dossier parent, je fais :

Code : PHP

```php
<?php include dirname(__FILE__).'/../fichier.php'; ?>
```

Notez que depuis la version 5.3 de PHP, il est possible de simplement utiliser la constante `__DIR__`.

Tout ça pour dire que je vais employer la méthode suivante pour définir les deux dossiers. Voici donc le code final :

Code : PHP

```php
<?php $tpl = new Talus_TPL_Engine(dirname(__FILE__).'/templates/',
dirname(__FILE__).'/cache/'); ?>
```

Je me place donc dans mon cas personnel où le fichier se trouve à la racine du site, mes templates dans un dossier **/templates/** et mon dossier de cache dans **/cache/**. Adaptez bien sûr à votre cas.

Ça marche pas ton code, tu m'as bien eu.

Ah, en effet. Mais vous devriez savoir pourquoi.

...

...

Si vous avez trouvé, bravo. Il suffisait en effet de penser à inclure le script contenant la classe PHP, **lib/Talus_TPL/Engine.php**.

Création des deux fichiers

Si vous avez bien tout compris, chaque fichier PHP va se trouver scindé en deux : la partie logique du code (en PHP toujours), et le fichier de template (utilisant la syntaxe de Talus' TPL). Avec Talus' TPL, le fichier de template porte l'extension **.html**. Sa syntaxe est de type XML. Pour plus d'infos, ce tutoriel de Tangui fait un point complet sur le XML. La syntaxe des fichiers de template ainsi que les fonctions PHP seront toutes détaillées par la suite.

Sachez que dans le fichier de template, vous aurez en gros deux types de structures :

- du code xHTML qui sera conservé ;
- du code utilisant la syntaxe particulière de Talus' TPL, qui sera converti en PHP lors de l'affichage, de façon transparente. Une fois la conversion effectuée, elle restera en cache (d'où l'utilité du dossier cache !), et le fichier résultant sera exécuté tel quel.

Tout code PHP dans le template sera affiché tel quel, et ne sera donc pas exécuté !

Affecter un template à un script, et afficher le template

Dans cette partie, nous allons voir quelque chose de primordial : comment afficher un fichier de template. En effet, il ne suffit pas de faire un include. Vous devez passer par une méthode dédiée.

La méthode `parse` vous permet de... parser le template. Comprenez que lorsque vous faites cela, toute la syntaxe spécifique de Talus' TPL est convertie en PHP, qui est enregistré (pour éviter de le retraduire à chaque fois). Ce code PHP est ensuite exécuté.

Le fichier PHP est modifié à chaque fois que votre template change, de façon automatique. Ce fichier intermédiaire est stocké dans le dossier de cache défini à l'instanciation de Talus_TPL_Engine. Vous pouvez forcer la compilation à se reproduire en spécifiant le second argument de `parse()` à `false`.

Lorsque vous appelez la méthode `parse()` de votre objet Talus_TPL_Engine (que j'instancie toujours dans une variable `$tpl`, question d'habitude), vous devez spécifier le nom du fichier de template à parser. Le chemin est - par rapport au chemin des fichiers de templates - indiqué lors de la création de `$tpl` (reportez-vous ci-dessus si ce n'est pas clair). Voici donc un schéma à retenir :

Code : PHP

```php
<?php
// Création de l'objet de template
$tpl = new Talus_TPL_Engine(dirname(__FILE__).'/templates/',
dirname(__FILE__).'/cache/');

/* Diverses opérations que nous allons voir par la suite */

// Parsage !
$tpl->parse('template.html');
```

Nous sommes maintenant prêts à voir ces fameuses instructions. Vous pourrez alors apprécier la puissance de ce script, et juger de ses différences par rapport à d'autres systèmes.

Utiliser des variables

Affecter et afficher une variable

Une des opérations les plus simples à faire avec votre nouveau moteur de templates est de pouvoir afficher des variables. Comme toute opération (ou presque), elle se passera en deux temps :

- vous devrez commencer par ajouter un morceau de code côté PHP pour **assigner la variable au moteur** ;
- ensuite, il faudra l'afficher depuis votre template.

Côté PHP

Vous disposez d'une fonction au nom simple à retenir pour assigner des variables : `Talus_TPL::set()`. Elle peut s'utiliser de

deux manières :

Code : PHP

```php
<?php
// Première méthode : on assigne 'abcd' à VAR
$tpl->set('VAR', 'abcd');

// Variante : on assigne 'abcd' à VAR par l'intermédiaire d'une
variable PHP
$texte = 'abcd';
$tpl->set('VAR', $texte);

// Seconde méthode : on assigne des variables multiples via un
array
$tpl->set(array('VAR1'=>'abcd', 'VAR2'=>'efgh', 'AUTRE_VAR'=>12.5));
```

Quelques remarques donc sur ce code.
La fonction `Talus_TPL::set()` prend deux arguments : le nom de la variable qui sera utilisée côté template, et sa valeur. Les variables suivent les mêmes règles de nommage que pour PHP. Vous pouvez ensuite assigner une valeur de n'importe quel type, donc aussi bien un array qu'une chaîne de caractères qu'un entier... sauf un objet. Les variables côté PHP et côté template ne doivent pas nécessairement porter le même nom.

L'assignation avec `Talus_TPL::set()` permet aussi d'affecter plusieurs variables en une seule fois. La fonction reçoit un array, avec dans les clés le nom de la variable côté template et dans les valeurs... sa valeur.

Côté template

L'affichage d'une variable est très simple :

Code : XML

```
{VAR}
```

affichera le contenu de la variable `VAR`. Vous pouvez aussi de la même manière accéder aux clés d'un array :

Code : XML

```
{VAR[1]['cle1']}
```

... ainsi qu'aux propriétés d'un objet (pour peu qu'elle soit publique, ou que la méthode __get() soit correctement implémentée) !

Code : XML

```
{VAR->attribute}
```

Pour des raisons techniques (limitation des expressions régulières), il est pour le moment impossible d'utiliser le symbole ']' dans la clé d'un array.

Échapper une variable

Et si je veux afficher `{VAR}` dans mon fichier **.html**, sans que cela soit interprété ?

Pour que VAR ne soit pas affichée, utilisez cela :

Code : XML

```
{\VAR}
```

Il s'affichera alors :

Citation : Rendu

{VAR}

On dit qu'on ***échappe*** la variable.

Utiliser la variable sans l'afficher

Pendant que j'y suis, une autre chose à connaître est la manière d'obtenir la variable sans l'afficher. Cela servira quand nous verrons les conditions et autres fonctionnalités de la syntaxe Talus' TPL.

Code : XML

```
{$VAR}
```

L'instruction renvoie la valeur de la variable, **mais sans l'afficher**. Un peu comme si vous faisiez de la manière suivante :

Code : PHP

```
<?php
$VAR;
```

Constantes

Enfin pour finir ce chapitre, voyons l'utilisation des constantes. Vous n'avez pas à les assigner à nouveau, elles sont accessibles depuis le fichier de template sans code additionnel côté PHP.

Code : XML

```
{__CONSTANTE__}
```

Il suffit donc juste de l'entourer de deux tirets bas __. Pour obtenir leur valeur, même topo que précédemment :

Code : XML

```
{__$CONSTANTE__}
```

A propos de constantes, vous pouvez également utiliser des constantes de classes comme suite :

Code : XML

```
{__Classe::CONST__}
{__$Classe::CONST__}
```

Utiliser les filtres

Un ajout intéressant de Talus' TPL par rapport à d'autres moteurs de templates est l'utilisation de filtres. Ils fonctionnent de la même manière que dans Django. Leur rôle est d'appliquer une transformation à la variable. Ils peuvent s'utiliser indifféremment à l'affichage ou à la récupération d'une variable. Pour les utiliser, ajoutez-les à la suite de la variable, séparés par des barres verticales |. Ils sont cumulatifs.

Code : XML

```
/* Affiche le contenu d'une variable en minuscules */
{VAR|minimize}
```

Code : XML

```
/*
 * Echappe les caractères HTML et convertit les sauts de ligne en
<br /> ;
 */
{VAR|protect|nl2br}
```

donnera à la compilation quelque chose comme :

Code : PHP

```
<?php echo nl2br(htmlspecialchars($tpl->vars['VAR'])); ?>
```

J'ai pris cet exemple car ici l'ordre a de l'importance. Essayez d'inverser les filtres si vous ne me croyez pas. 

Les filtres peuvent aussi accepter des *arguments*. Pour cela, après le nom du filtre, séparez vos arguments par le symbole deux-points.

Code : XML

```
/* Protège la variable VAR, la coupe à 50 caractères maximum, et
interprete les sauts de ligne. */
{VAR|protect|cut:50|nl2br}
```

De même, vous pouvez aussi ajouter des filtres automatiques sur vos variables (lors de l'assignation via `set`, vos variables seront donc automatiquement transformées par le filtre renseigné), grâce à la méthode `Talus_TPL::autoFilters()` :

Code : PHP

```
<?php
$tpl->autoFilters('protect'); // protect sera appliqué sur chaque
variable déclarées dans l'objet $tpl
```

Je finirai sur une petite liste des filtres disponibles de base, sachant que cette liste peut évoluer. Vous pouvez de plus facilement ajouter les vôtres. Notez que certains filtres peuvent avoir des paramètres supplémentaires, mais je vous laisser explorer à votre discrétion pour les découvrir !

Nom du filtre	Usage
floor	Arrondit le nombre à l'entier inférieur.
ceil	Arrondit le nombre à l'entier supérieur.
protect	Échappe les caractères HTML du contenu de la variable.
safe	Désactive une application de protect sur le contenu de la variable. Utile si, par exemple, vous avez mis le filtre automatique "protect".
maximize	Met le contenu de la variable en majuscules.
minimize	Met le contenu de la variable en minuscules.
ucfirst	Met la première lettre du contenu de la variable en majuscule.
lcfirst	Met la première lettre du contenu de la variable en minuscule.
invertCase	Inverse la casse dans le contenu de la variable.
convertCase	Convertit la casse du contenu de la variable
nl2br	Convertit les sauts de ligne en dans le contenu de la variable.
paragraphy	Transforme le contenu d'une variable en série de paragraphes ou sauts de ligne suivant le contexte.
slugify	Convertit le contenu de la variable en *slug* (type celui qu'on trouve sur le site du zéro, dans la barre d'adresse, pour des sujets, news ou tutoriels).

Notez qu'il existe un sujet sur Talus' Works, qui recense les filtres que les utilisateurs ont pu créer ; n'hésitez pas à consulter cette liste, et même à proposer de nouveaux filtres ; peut-être seront-ils intégrés à une future version de Talus' TPL ?

Employer des conditions

Vous vous demandez sans doute à ce stade comment effectuer des tests simples sur les variables. Bien sûr que Talus' TPL vous l'autorise, et nous allons le voir maintenant.

Il existe trois balises de noms identiques à ceux en PHP, à employer côté template. Notez qu'il n'y a aucune manipulation spéciale à effectuer du côté de PHP. Ces balises sont :

- `if` ;
- `elseif` ;
- `else`.

Il y a un argument à passer aux balises `if` et `elseif` : la condition. En pratique, procédez comme suit (je pars du principe que j'ai créé une variable `VAR` dans le code PHP) :

Code : XML

```
<if condition="{$VAR} == 3">
 {\VAR} vaut 3
 <elseif condition="{$VAR} == 4" />
 {\VAR} vaut 4
 <elseif condition="{$VAR} == 5" />
 {\VAR} vaut 5
 /* et ainsi de suite */
```

```
 <else />
 {\VAR} a une autre valeur
</if> /* on n'oublie pas de fermer le if */
```

Simple, non ? Notez au passage l'utilisation de l'échappement pour afficher le nom de la variable. N'oubliez pas non plus le signe dollar $ devant la variable (on ne veut pas l'afficher mais *récupérer sa valeur*).

Deux petits raccourcis pour les fainéants.

- Vous pouvez abréger l'attribut `condition` en `cond`. Le code devient donc :

Code : XML

```
<if cond="{$VAR} == 3">
 {\VAR} vaut 3
 <elseif cond="{$VAR} == 4" />
 {\VAR} vaut 4
 <elseif cond="{$VAR} == 5" />
 {\VAR} vaut 5
 <else />
 {\VAR} a une autre valeur
</if>
```

- La balise **<elseif>** peut se raccourcir en **<elif>** :

Code : XML

```
<if cond="{$VAR} == 3">
 {\VAR} vaut 3
 <elif condition="{$VAR} == 4" />
 {\VAR} vaut 4
 <elif condition="{$VAR} == 5" />
 {\VAR} vaut 5
 <else />
 {\VAR} a une autre valeur
</if>
```

Les deux raccourcis sont bien évidemment cumulatifs ! 

Et pour faire des boucles ?

Justement ça tombe bien j'y viens.  Les boucles sont très utilisées, que ce soit pour parcourir des array, ou récupérer un résultat provenant d'une base de données contenant plusieurs lignes.

Parcourir un array avec foreach

Vous êtes sans doute habitués à vous balader dans des array avec la structure `foreach` en PHP. Eh bien bonne nouvelle, vous pouvez faire la même chose avec Talus' TPL du côté des templates. Toutefois, La syntaxe est un peu différente de celle utilisée en PHP. Vous utiliserez la balise **<foreach>**, mais en lui fournissant un seul argument : l'array à parcourir.

Code : XML

```
<foreach array="{$ARRAY}">
 /* Opérations sur l'array */
</foreach>
```

Une fois entre les deux balises du **<foreach>**, vous pouvez accéder soit à la clé, soit à la valeur contenue dans l'array. À chaque itération, ces données changent bien évidemment, comme en PHP. Vous devez donc concrètement spécifier si vous

voulez la clé ou la valeur, et de quel array (cela vous permet d'imbriquer des `foreach` ; cependant, vous verrons ce point un peu plus tard).

Code : XML

```
<foreach ary="{$ARRAY}">
 La clé est {ARRAY.key} et sa valeur {ARRAY.value}
 <if cond="{$ARRAY.val} == 1">Je dirais même que sa valeur est
1</if>
</foreach>
```

Notez ici d'autres raccourcis pour fainéants, comme ceux évoqués lors des conditions : l'attribut "array" du foreach peut se condenser en "ary", et "value" en "val" !

Vous pouvez donc soit **afficher** les valeurs des variables (`{ARRAY.key}` et `{ARRAY.val}`), soit **obtenir** leurs valeurs pour effectuer des tests dessus (soient `{$ARRAY.key}` et `{$ARRAY.val}`).

Renommer un array lors de l'itération à travers le <foreach>

Vous avez aussi la possibilité de renommer un array ; cela vous permet d'imbriquer des array, éventuellement identiques. Si vous souhaitez procéder à ce genre de manipulation (surtout si vous voulez itérer à travers la valeur de l'array souhaité), l'attribut `as` est obligatoire. Ceci étant dit, procédez donc comme suit pour utiliser cette fonctionnalité :

Code : XML

```
<foreach ary="{ARRAY}">
    <foreach ary="{$ARRAY.val}" as="{$ARY}">
        La clé est {ARY.key} est sa valeur {ARY.value}
        <if cond="{$ARY.value} == 1">Même que sa valeur est 1</if>
    </foreach>
</foreach>
```

Attributs spéciaux

Des attributs sont automatiquement affectées à la boucle, sans que vous n'ayez rien à faire. Il s'agit de :

- `{$ARRAY.is_first}` : booléen indiquant si c'est la première occurrence du bloc ;
- `{$ARRAY.is_last}` : booléen indiquant si c'est la dernière occurrence du bloc ;
- `{$ARRAY.size}` : entier indiquant le nombre d'occurrences de votre array ;
- `{$ARRAY.current}` : entier indiquant à quelle occurence nous sommes actuellement.

Vous les récupérez de la manière classique :

Code : XML

```
<foreach ary="{$ARRAY}">
    Ma valeur est {ARRAY.val} ({ARRAY.cur} occurence / {ARRAY.size}
occurence(s))
<foreachelse />
    Aucun membre.
</foreach>
```

Notez, enfin, que vous pouvez voir l'alternative à un **<foreach>** (dans le cas où le tableau à parcourir est vide, par exemple)

par la balise **<foreachelse />**.

Un autre raccourci s'est glissé dans le code. Sauriez-vous le retrouver ? 

Maintenant, les inclusions !

Vous pouvez inclure un template depuis un autre fichier de template. Il vous suffit d'utiliser la balise **<include>**, en envoyant comme argument le chemin vers le template à inclure. Vous pouvez spécifier si vous ne désirez l'inclure qu'une fois (à savoir que s'il a déjà été inclus, il ne pourra pas l'être à nouveau plus tard ; c'est l'équivalent de `include_once` en PHP), en envoyant un boolean (`true` ou `false`). Vous pouvez aussi demander un "require", en changeant le nom de la balise en... **<require>**.

Code : XML

```
<include tpl="menu.html" /> /* strictement équivalent à <include
tpl="menu.html" once="false" /> */
<include tpl="erreur.html" once="true">
<require tpl="fichier_important.html">
```

Concernant l'argument "tpl", il y a plusieurs remarques à savoir...

- les fichier doivent se terminer par **.html** ;
- vous pouvez, à la place d'un fichier, mettre une variable en paramètre : **<include** `tpl="{$MA_VAR}"` **>**.

Enfin, une fonctionnalité intéressante : vous pouvez spécifier des variables spéciales à passer au fichier inclus. Celles-ci seront donc accessibles dans le template enfant (et seulement le template enfant !), ainsi que tous les fichiers qu'il inclut. Vous avez juste à ajouter, après le nom du template à ajouter, une *query string* (un peu comme quand vous transférez des variables de type GET) : **<include** `tpl="mon_template.html?var1=ma_valeur&var2=autre_valeur"` **>**. Les variables `{var1}` et `{var2}` seront donc accessibles dans le template enfant.

Simple, clair et efficace. 

Mise en œuvre : un système de news

Reprenons l'exemple usé jusqu'à la corde du système de news. Je vais vous donner un code PHP « banal », et à vous de l'adapter à ce moteur de template.

Voici le code pour créer une table `news` avec quelques éléments déjà insérés :

Code : SQL

```
CREATE TABLE `news` (
`id` SMALLINT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`titre` VARCHAR( 255 ) NOT NULL ,
`auteur` VARCHAR( 255 ) NOT NULL ,
`date` DATETIME NOT NULL ,
`contenu` TEXT NOT NULL
) ENGINE = MYISAM ;

INSERT INTO `news` (
`id` ,
`titre` ,
`auteur` ,
`date` ,
`contenu`
)
VALUES (
NULL , 'Une première news', 'vincent1870', '2007-12-30 18:38:02',
'Bienvenue à tous sur ce beau site !<br /> <br /> Bon surf ! ;)'
```

```
), (
NULL , 'Et une deuxième', 'Arthur', '2007-12-11 18:38:44', 'Hello
!<br /> What''s happening ?'
);
```

Copiez et collez ce code dans phpMyAdmin, onglet **SQL**, pour créer votre table.
Et voici le code PHP. Je ne le commenterai pas ; si vous avez du mal à le comprendre, relisez les cours de M@teo21.

Code : PHP

```
<?php
try {
  $pdo = new PDO('mysql:host=localhost;dbname=tests', 'root', '');
} catch (PDOException $e) {
  die('Erreur : ' . $e->getMessage());
}
?>

<h1>Les news du site</h1>

<?php
$q = $pdo->query("SELECT id, auteur, titre, DATE_FORMAT(date,
'%d/%m/%Y %Hh%i') AS date_formatee, contenu
FROM news
ORDER BY date DESC");
foreach($q as &$data){
    echo '
<div class="news">
<h2>' . $data['titre'] . '</h2>
<p>News postée le ' . str_replace(' ', ' à ',
$data['date_formatee']) . ' par ' . $data['auteur'] . '</p>
<p>' . $data['contenu'] . '</p>
</div>';
}

$q = null;

//On ferme la connexion à MySQL
$pdo = null;
```

Bon, maintenant au boulot ! Vous devez me transformer ce script pour y intégrer Talus' TPL. Vous devrez donc créer un second fichier, contenant le template. Bonne chance !

...

Bien ! Voici maintenant le corrigé, pour comparer nos travaux. Je commence par **news.php** :

Code : PHP

```
<?php
//On se connecte à MySQL
try {
  $pdo = new PDO('mysql:host=localhost;dbname=tests', 'root', '');
} catch (PDOException $e) {
  die('Erreur : ' . $e->getMessage());
}

//On inclut et démarre Talus' TPL
require 'lib/Talus_TPL/Engine.php';
$tpl = new Talus_TPL_Engine('./', 'cache/');

//On stocke les news dans un array
$news = array();
$q = $pdo->query("
```

```
$q = $pdo->query('
SELECT id, auteur, titre, DATE_FORMAT(date, '%d/%m/%Y \à %Hh%i') AS
date_formatee, contenu
FROM news
ORDER BY date DESC");

$data = $q->fetchAll();

//On ferme la connexion à MySQL
$q = null;
$pdo = null;

//On assigne la liste des news
$tpl->set('NEWS', $news);

//On affiche le résultat
$tpl->parse('news.html');
```

Le code se comprend facilement si vous avez lu tout ce qui précède. Je considère que le template est dans le même dossier, et que vous disposez d'un dossier **cache/** à la racine. Et voici le template **news.html** :

Code : XML

```
<h1>Les news du site</h1>

<foreach ary="{$NEWS}">
    <div class="news">
        <h2>{NEWS.val['titre']}</h2>
        <p>News postée le {NEWS.val['date_formatee']} par
{NEWS.val['auteur']}</p>
        <p>{NEWS.val['contenu']}</p>
    </div>
</foreach>
```

Je finirai par vous laisser quelques liens relatifs à Talus' TPL...

- Le site contenant les productions de Talus, Talus' Works.
- La documentation officielle, plus complète que ce tuto, et plus vite mise à jour, mais moins explicative (*une doc n'est pas un tuto*).
- Le projet GitHub : suivez l'avancement de Talus' TPL, au jour le jour.
- Un bug à rapporter ? Une suggestion à proposer ? Visitez le *bug tracker* ! 

J'espère que ce tutoriel vous a été profitable. Les commentaires sont là, mais je rappelle juste qu'ils ne doivent pas servir de support pour ce script. Les liens ci-dessus sont là pour ça !

Je remercie vincent1870 pour son travail originel sur ce tutoriel, ainsi que vyk12 et lui-même pour leurs dernières relectures.