Tutorial 1 : minimal example - simple variables replacements

The purpose of this tutorial is to show you the basic feature of odtPHP : simple variables replacement.

The PHP code

```php
<?php
require_once('../library/odf.php');

$odf = new odf("tutoriel1.odt");

$odf->setVars('titre', 'PHP');

$message = "PHP  est un langage de scripts libre ...";

$odf->setVars('message', $message);

$odf->exportAsAttachedFile();

?>
```

1 - After including the odtPHP library, we can create a new object with the constructor of the Odf class. This constructor takes one mandatory parameter : the path (relative or absolute) to the OpenOffice document template.

2 - Then, we can call the method setVars() from the object $odf in order to replace the tag {titre} from the template by the 'PHP' text. The first mandatory parameter of the method setVars() is the tag name in the template (without specifying the delimiters {}). The second mandatory parameter is the text content that will replace this tag within the final document.

3 - We call this method a second time in order to replace the tag {message} from template by the text 'PHP est un langage de scripts libre...'.

4 - Eventually, we call the method exportAsAttachedFile() in order to send the result directly to the client browser.

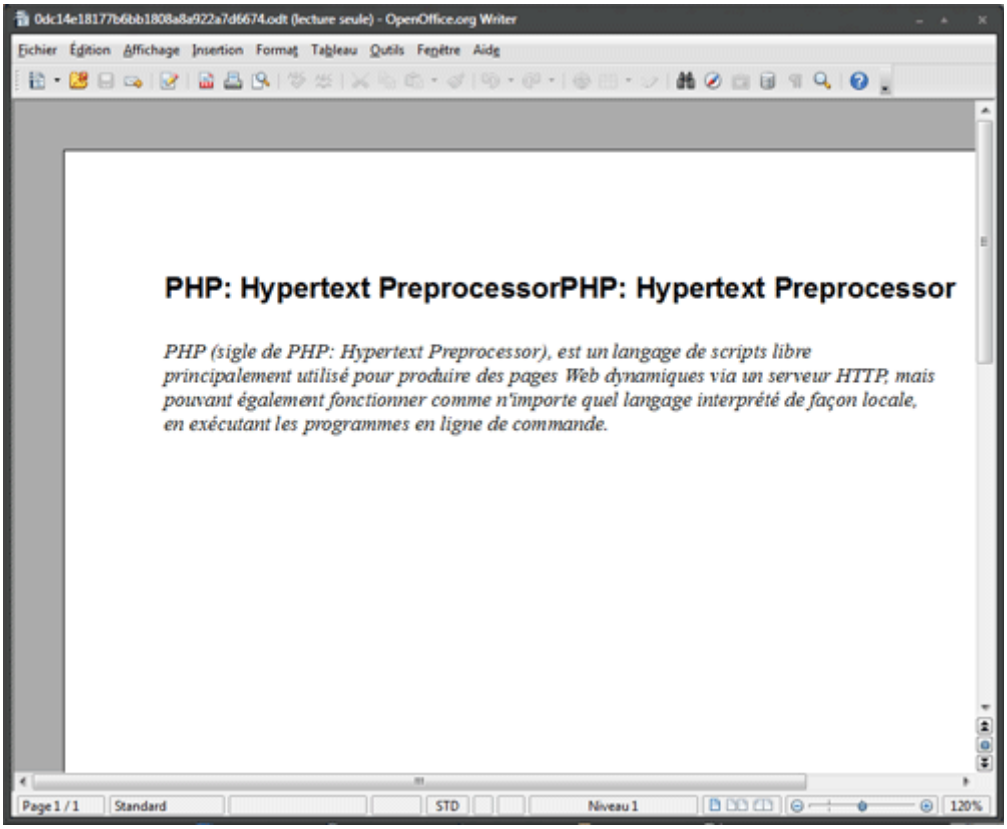
The OpenOffice model (tutoriel1.odt)

{titre}

{message}

To run this example, copy this template within an OpenOffice document named "tutoriel1.odt". In the template, tags are delimited by two braces {}.

The result



Note : the texts in the PHP code are shorter than the result in order to make it more readable. Styles (colors, layouts) of this result come from our template.

Tutorial 2: Insert an image in the document

The purpose of this tutorial is to show how to insert an image into the OpenOffice document generated with odtPHP. Adding an image to the generated document is substantially the same thing as the simple replacement of a tag by text content (tutorial 1). Note that for this example, you will have to indicate valid paths to existing images.

The PHP code

```
<?php
require_once('../library/odf.php');

$odf = new odf("tutoriel2.odt");

$odf->setVars('titre','Anaska formation');

$message = "Anaska, leader Français de la formation informatique ...";

$odf->setVars('message', $message);

$odf->setImage('image', './images/anaska.jpg');

$odf->exportAsAttachedFile();
```

```
?>
```

1 - We create an object Odf by specifying the path to our template. Then, we call two times the method setVars() of the created object in order to replace tags {titre} and {message} by text content.

2 - The method setImage() allows us to add an image to the OpenOffice document. The first parameter of this method is the name of a tag somewhere in our template (without specifying the delimiters{}). The second parameter is the path (relative or absolute) to an image stored on the server. The tag will be replaced by the image in the document generated by odtPHP.

3 - When calling the method exportAsAttachedFile() , the library compress for us the desired images into the document and send the result to the client browser.

The OpenOffice model (tutoriel2.odt)

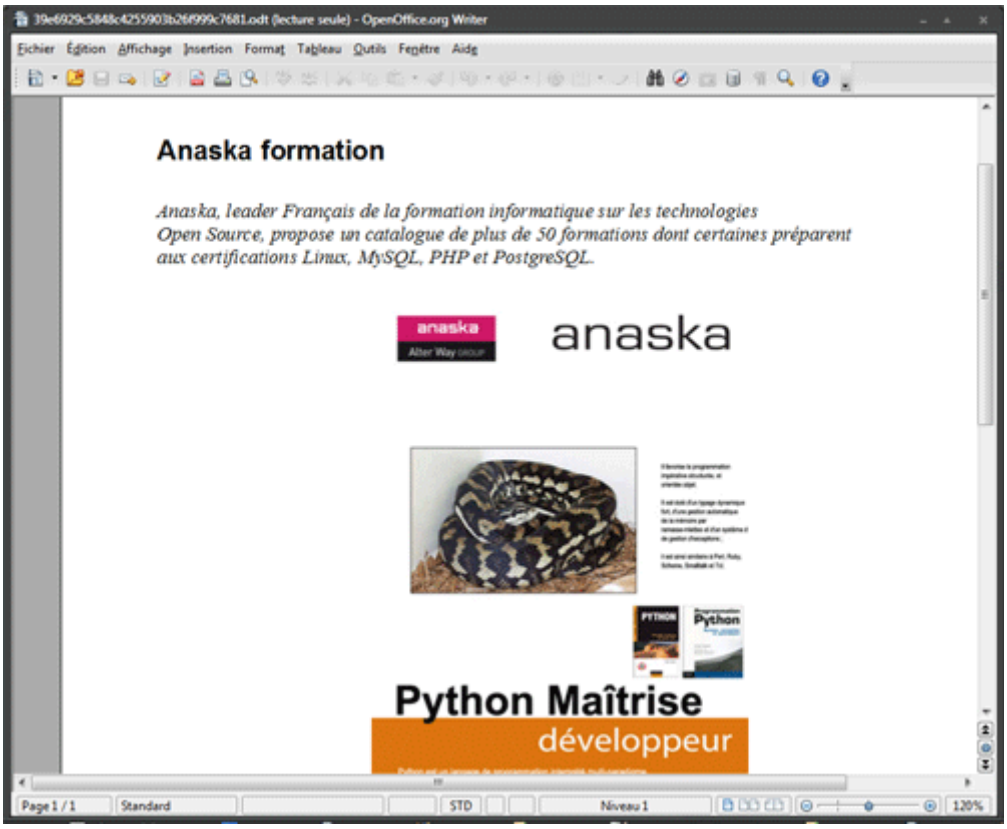
{titre}

{message}

{image}

To run this example, copy this template within an OpenOffice document named "tutoriel2.odt". In the template, tags are delimited by two braces {}.

The result



Note : the texts in the PHP code are shorter than the result in order to make it more readable. Styles (colors, layouts) of this result come from our template.

Tutorial 3 : Repeat a part of the document (Segment)

This tutorial purpose is to show how to repeat a part of your template with segments. We will see in this example how to display a list of articles in our final OpenOffice document.

The PHP code

```
<?php
require_once('../library/odf.php');

$odf = new odf("tutoriel3.odt");

$odf->setVars('titre', 'Quelques articles de l\'encyclopédie Wikip
édia');

$message = "La force de cette encyclopédie en ligne réside...";

$odf->setVars('message', $message);

$listeArticles = array(
    array(    'titre' => 'PHP',
              'texte' => 'PHP est un langage de scripts (...)',
    ),
    array(    'titre' => 'MySQL',
              'texte' => 'MySQL est un système de gestion de base de
données (...)',
    ),
    array(    'titre' => 'Apache',
              'texte' => 'Apache HTTP Server, souvent appelé Apache
(...)',
    ),
)

$article = $odf->setSegment('articles');
foreach($listeArticles AS $element) {
    $article->titreArticle($element['titre']);
    $article->texteArticle($element['texte']);
    $article->merge();
}
$odf->mergeSegment($article);

$odf->exportAsAttachedFile();

?>
```

1 - We create an Odf object by specifying the path to our template. Then, we call two times the method setVars() of the created object in order to replace {titre} and {message} tags by text content.

2 - For the second step, we initialize in an array a data set that we want to fully display. The array provides a list of articles consisting of a title and a text.

3 – We initialize the segment "articles" with a call to the method setSegment() which takes as argument the name of the segment and returns an object of type Segment.

4 - Then, We can loop on our data array. At each loop, we replace the tags {titreArticle} and {texteArticle} of segment by content of the current line of the array. For that, we directly access to the methods titreArticle() and texteArticle() of the object $article. You can also perform this replacement in a classic way by calling the method setVar() of the object $article.

5 - Eventually, when we have added all our desired data, we can add the segment to the document by calling the method mergeSegment() of the object $odf which takes as parameter our segment $article.

The OpenOffice model (tutoriel3.odt)

{titre}

{message}

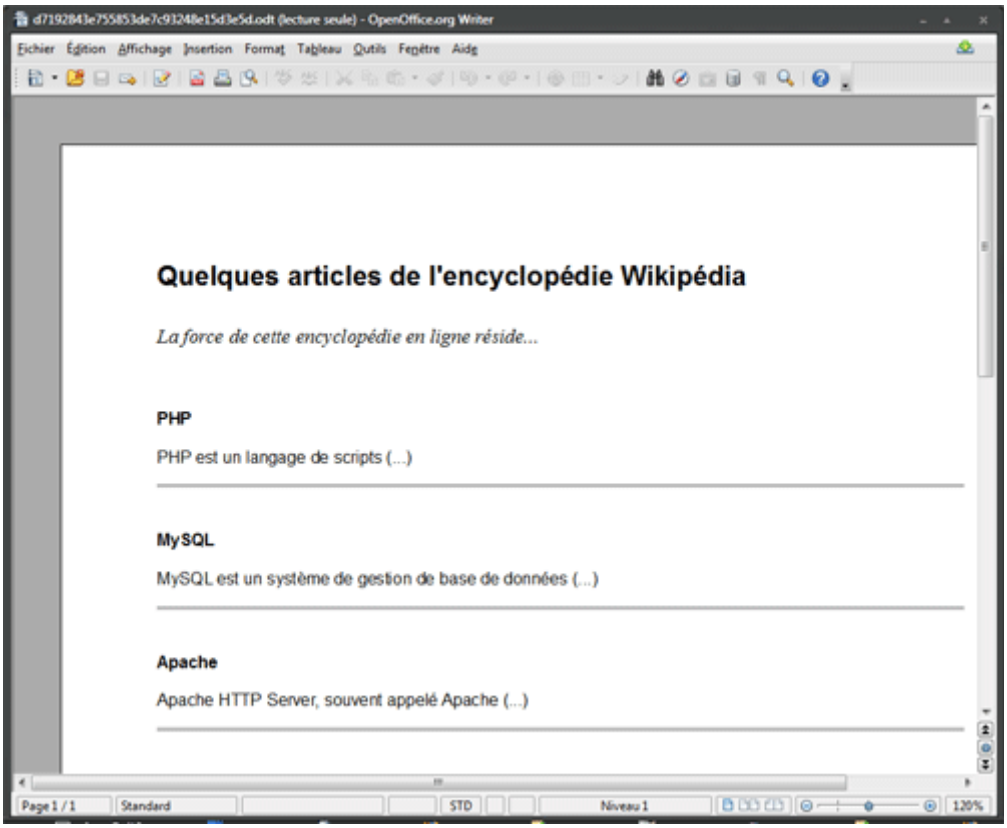
[!-- BEGIN articles --]
{titreArticle}

{texteArticle}

[!-- END articles --]

To run this example, copy this template within an OpenOffice document named "tutoriel3.odt". In the template, segments are delimited by a beginning and ending tag : [!-- BEGIN nomSegment --] [!-- END nomSegment --].

The result



Note : the texts in the PHP code are shorter than the result in order to make it more readable. Styles (colors, layouts) of this result come from our template.

Tutorial 4 : Imbricate several segments

The purpose of this tutorial is to show how to imbricate several segments within others segments. Imbricating segments is a very simple task with odtPHP.

The PHP code

```
<?php
require_once('../library/odf.php');

$odf = new odf("tutoriel4.odt");

$odf->setVars('titre','Articles disponibles :');

$categorie = $odf->setSegment('categories');
for ($j = 1; $j <= 2; $j++) {
    $categorie->setVar('TitreCategorie', 'Catégorie ' . $j);
    for ($i = 1; $i <= 3; $i++) {
        $categorie->articles->titreArticle('Article ' . $i);
        $categorie->articles->date(date('d/m/Y'));
        $categorie->articles->merge();
    }
    for ($i = 1; $i <= 4; $i++) {
        $categorie->commentaires->texteCommentaire('Commentaire '
. $i);
        $categorie->commentaires->merge();
    }
    $categorie->merge();
}
$odf->mergeSegment($categorie);

$odf->exportAsAttachedFile();

?>
```

1 - We create an Odf object by specifying the path to our template. Then, we call the method setVars() of the created object in order to replace the tag {titre} by text content.

2 – We initialize the segment "categories" with a call to the method setSegment() which takes for argument the name of the segment and returns an object of type Segment.

3 – For this example, we want to add to the generated document two categories. So, we loop a first time from 1 to 2. At each loop, we indicate a title to our current category with a call to the method setVar().

4 – In the loop, each child segment is accessible through the property of the same name of the father segment. For instance, the segment "articles" is a child of segment "categories". So, we can access to it directely with $categorie->articles. Thus, we can replace variables of this child segment with the method that has the same name of the tag to replace. For instance : $categorie->articles->titreArticle(). Alternatively, we can also call the method setVar() which will perform the same replacement.

5 – For each child and for the segment "categories", we call the method merge() in order to merge data for each loop. Only one call to mergeSegment() is required at the end to add all our segments to the generated document.

The OpenOffice model (tutoriel4.odt)

{titre}

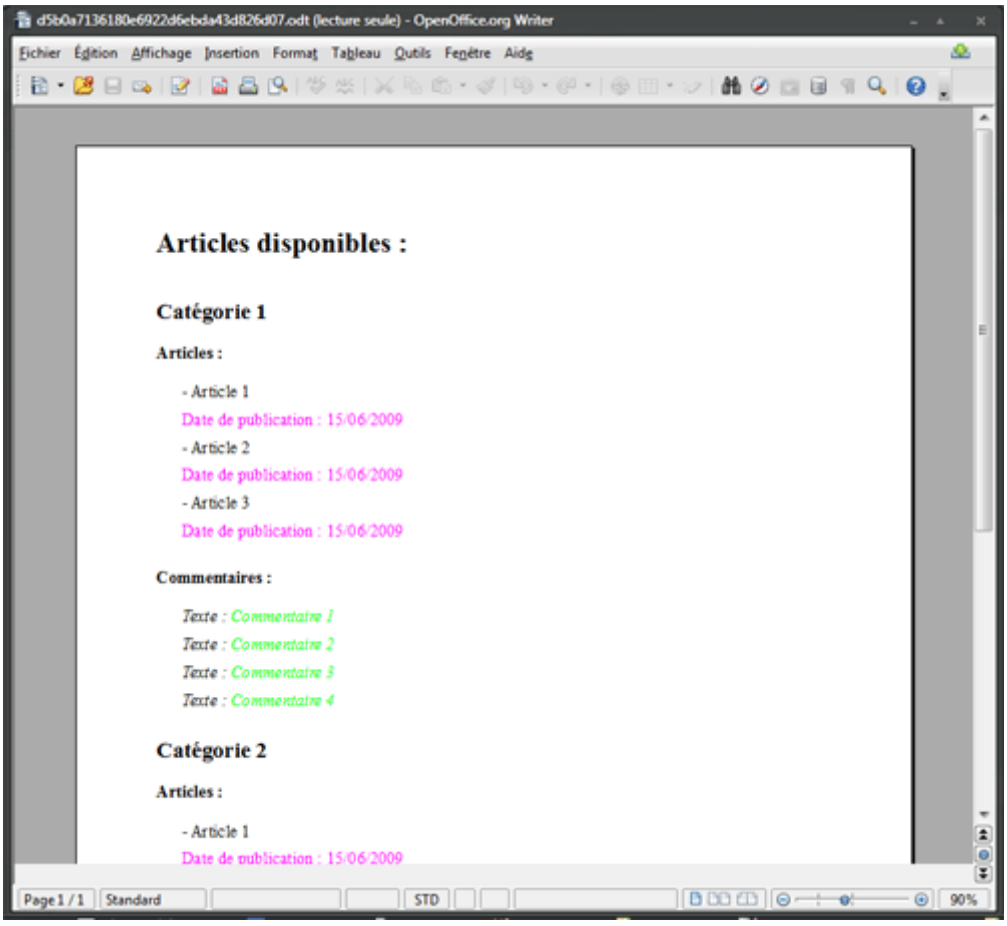
[!-- BEGIN categories --]
{TitreCategorie}

Articles :
[!-- BEGIN articles --]
- {titreArticle}
Date de publication : {date}[!-- END articles --]

Commentaires :
[!-- BEGIN commentaires --]
Texte : {texteCommentaire}[!-- END commentaires --]
[!-- END categories --]

To run this example, copy this template within an OpenOffice document named "tutoriel4.odt". In the template, we can imbricate segments easily. The segment "categories" contains a segment "articles" and another segment "commentaires".

The result



Styles (colors, layouts) of this result come from our template.

Tutorial 5 : Insert images within a segment

The purpose of this tutorial is to show how to insert an image in a segment. This task is significantly

the same thing as adding an image to the root of the document. Instead of calling the method setImage() on the object $odf, we call it on the object of type Segment.

The PHP code

```
<?php

require_once('../library/odf.php');

$odf = new odf("tutoriel5.odt");

$odf->setVars('titre', 'Quelques articles de l\'encyclopédie Wikip
édia');

$message = "La force de cette encyclopédie en ligne réside...";

$odf->setVars('message', $message);

$listeArticles = array(
    array(    'titre' => 'PHP',
              'texte' => 'PHP, est un langage de scripts (...)',
              'image' => './images/php.gif'
    ),
    array(    'titre' => 'MySQL',
              'texte' => 'MySQL est un système de gestion de base de
données (...)',
              'image' => './images/mysql.gif'
    ),
    array(    'titre' => 'Apache',
              'texte' => 'Apache HTTP Server, souvent appelé Apache
(...)',
              'image' => './images/apache.gif'
    )
);

$article = $odf->setSegment('articles');
foreach($listeArticles AS $element) {
    $article->titreArticle($element['titre']);
    $article->texteArticle($element['texte']);
    $article->setImage('image', $element['image']);
    $article->merge();
}
$odf->mergeSegment($article);

$odf->exportAsAttachedFile();

?>
```

1 - We create an Odf object by specifying the path to our template. Then, we call two times the method setVars() of the created object in order to replace {titre} and {message} tags by text content.

2 - For the second step, we initialize in an array a data set that we want to fully display. The array provides a list of articles consisting of a title, a text and an image.

3 – We initialize the segment "articles" with a call to the method setSegment() which takes as argument the name of the segment and returns an object of type Segment.

4 - Then, We can loop on our data array. At each loop, we replace the tags {titreArticle} and {texteArticle} of segment by content of the current line of the array. For that, we directly access to the methods titreArticle() and texteArticle() of the object $article. You can also perform this replacement in a classic way by calling the method setVar() of the object $article. In the same way, we call the method setImage() in order to replace the tag image by an image of our choice. When we have made these replacements, we merge the segment by calling the method merge() of the object $article.

5 - Eventually, when we have added all our desired data, we can add the segment to the document by calling the method mergeSegment() of the object $odf which takes as parameter our segment $article.

The OpenOffice model (tutoriel5.odt)

{titre}

{message}

[!-- BEGIN articles --]
{titreArticle}

{texteArticle}
{image}

[!-- END articles --]

To run this example, copy this template within an OpenOffice document named "tutoriel35.odt". In the template, an image is represented by a classic tag.

Tutorial : Repeat the line of a table

The purpose of this tutorial is to show how to repeat a line of table with odtPHP segment. We can do that by the same way as handling ordinary segments, except that the tags in our template must be prefixed by "row.".

The PHP code

```
<?php
require_once('../library/odf.php');

$odf = new odf("tutoriel6.odt");

$odf->setVars('titre', 'Quelques articles de l\'encyclopédie Wikip
édia');

$message = "La force de cette encyclopédie en ligne réside...";

$odf->setVars('message', $message);

$listeArticles = array(
```

```
    array(    'titre' => 'PHP',
              'texte' => 'PHP est un langage de scripts (...)',
    ),
    array(    'titre' => 'MySQL',
              'texte' => 'MySQL est un système de gestion de base de
données (...)',
    ),
    array(    'titre' => 'Apache',
              'texte' => 'Apache HTTP Server, souvent appelé Apache
(...)',
    ),
);

$article = $odf->setSegment('articles');
foreach($listeArticles AS $element) {
    $article->titreArticle($element['titre']);
    $article->texteArticle($element['texte']);
    $article->merge();
}
$odf->mergeSegment($article);

$odf->exportAsAttachedFile();

?>
```

1 - We create an Odf object by specifying the path to our template. Then, we call two times the method setVars() of the created object in order to replace {titre} and {message} tags by text content.

2 - For the second step, we initialize in an array a data set that we want to fully display. The array provides a list of articles consisting of a title and a text.

3 – We initialize the segment "articles" with a call to the method setSegment() which takes as argument the name of the segment (without the prefix "row.") and returns an object of type Segment.

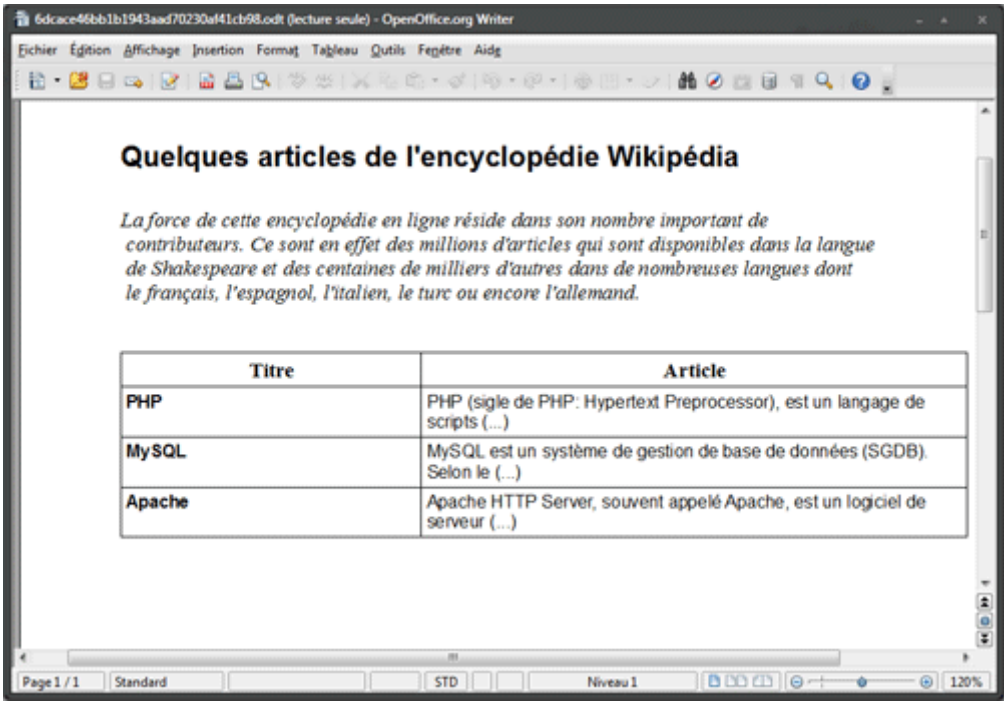
4 - Then, We can loop on our data array. At each loop, we replace the tags {titreArticle} and {texteArticle} of segment by content of the current line of the array. For that, we directly access to the methods titreArticle() and texteArticle() of the object $article. You can also perform this replacement in a classic way by calling the method setVar() of the object $article.

5 - Eventually, when we have added all our desired data, we can add the segment to the document by calling the method mergeSegment() of the object $odf which takes as parameter our segment $article.

The OpenOffice model (tutoriel6.odt)

To run this example, copy this template within an OpenOffice document named "tutoriel6.odt". This template requires to create a table under OpenOffice. Nmaes of segments within a line of array must be prefixed by "row." in order to be handled correctly.

The result



Note : the texts in the PHP code are shorter than the result in order to make it more readable. Styles (colors, layouts) of this result come from our template.

Tutorial 7 : Change the library behaviour

The purpose of this tutorial is to show how to change the behaviour of odtPHP. To do that, you just have to give to the constructor of the Odf class a second optional parameter.

The PHP code

```
<?php
require_once('../library/odf.php');

$config = array(
    'ZIP_PROXY' => 'PhpZipProxy', // Make sure you have Zip extens
ion loaded
    'DELIMITER_LEFT' => '#', // Yan can also change delimiters
    'DELIMITER_RIGHT' => '#'
);

$odf = new odf("tutoriel7.odt", $config);

$odf->setVars('titre', 'PHP: Hypertext PreprocessorPHP: Hypertext
Preprocessor');

$message = "PHP est un langage de scripts libre ...";

$odf->setVars('message', $message);
$odf->exportAsAttachedFile();

?>
```

Change the behaviour of odtPHP is very easy. you just have to give a configuration array for the second parameter of the Odf class constructor. The keys of this array are configuration names to modify : ZIP_PROXY, DELIMITER_LEFT and DELIMITER_RIGHT.

ZIP_PROXY is used to modify the proxy to use for handling Zip compression/decompression within odtPHP. By default, 'PclZip' is used. It consists of an interface with the PclZip library. You can also use 'PhpZipProxy'. In this case, odtPHP will directly use Zip extension from PHP. Note that you will have to activate this extension in your PHP configuration. Moreover, note that this extension is bugged since PHP 5.2.7.

DELIMITER_LEFT and DELIMITER_RIGHT are used to personalize delimiters of tags for the template. By default, these delimiters are braces : {}.

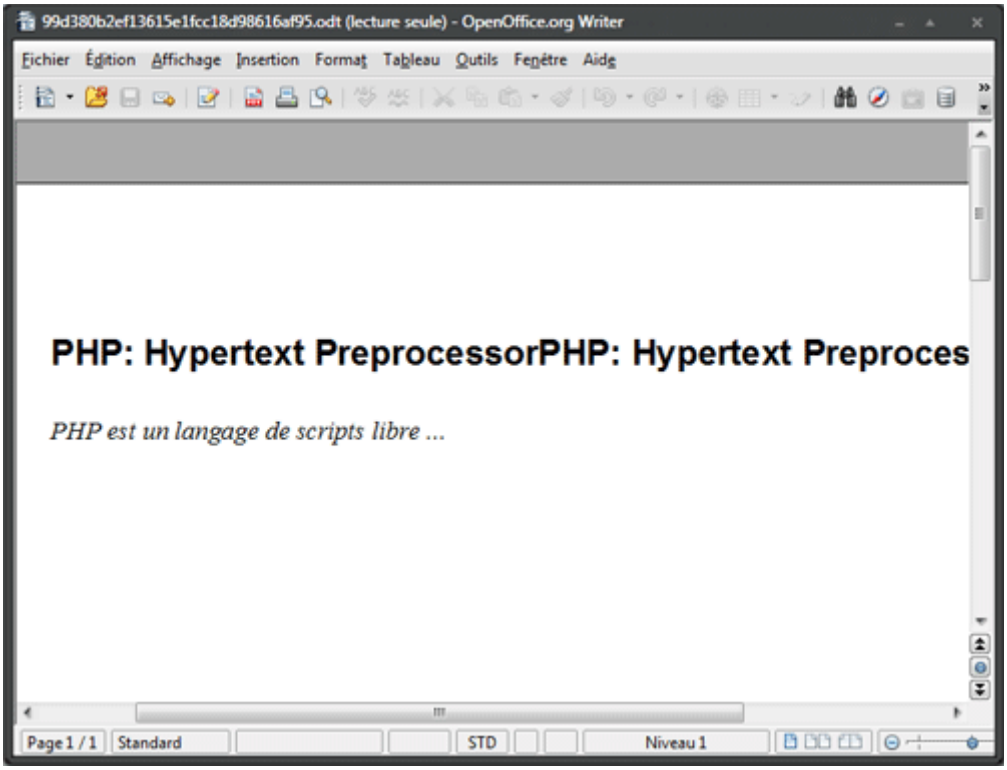
The OpenOffice model (tutoriel7.odt)

#titre#

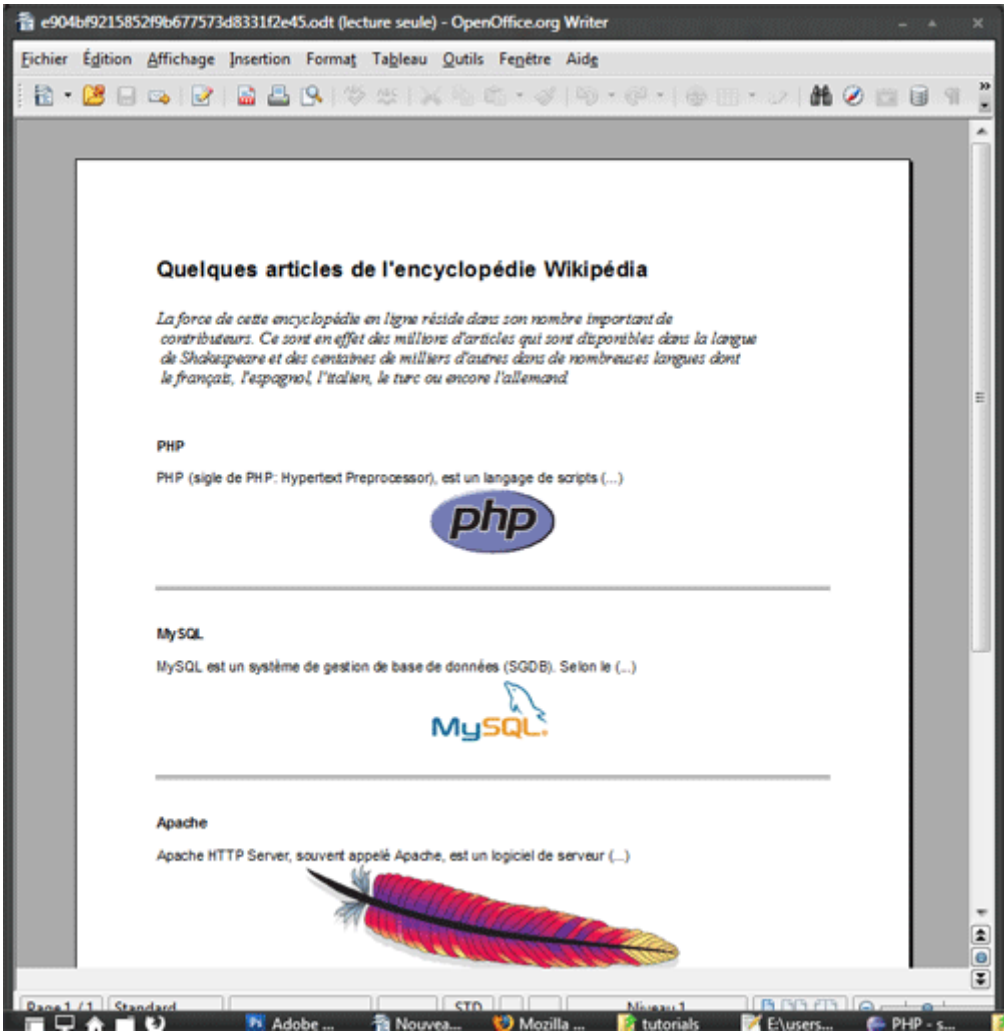
#message#

Here, we have choosen to use the character # as delimiter of tags.

The result



The result



Note : the texts in the PHP code are shorter than the result in order to make it more readable. Styles (colors, layouts) of this result come from our template.

Tutorial 8 : Advanced options of the method setVars()

The purpose of this tutorial is to show optional parameters of the method setVars().

The PHP code

```
<?php
$odf->setVars($key, $value, $encode, $charset);
?>
```

The method setVars() requires just two parameters : the name $key of a tag within the template and a text $value. However, two optional arguments can be provided.

$encode is a boolean. If this parameter is equal to true, special HTML characters (<, >, ", &) will be encoded in entities. By default, this parameter equals to true. If you set it to false, you will be able to insert raw XML data.

$charset is name of a characters set. By default, this parameter equals to 'ISO-8859'. Data are so automatically encoded in UTF-8 before adding them in the XML document. If you set this parameter to 'UTF-8', then data will not be encoded a second time in UTF-8. This can be useful to provide to the template data that are already encoded in UTF-8.

Tutorial 9 : Retrieve the generated document

Ce tutoriel a pour objectif de vous montrer les deux méthodes possibles pour récupérer le résultat d'un traitement avec odtPHP.

The purpose of this tutorial is to show two possible methods to retrieve the result of a process with odtPHP.

The PHP code

```
<?php
$odf->exportAsAttachedFile();

$odf->saveToDisk('fichier.odt');
?>
```

The method exportAsAttachedFile() allows you to send the result directly to the client browser. Be careful, before calling this method, make sure that HTTP headers have not been already sent.

La méthode saveToDisk() enregistre le résultat sur le serveur. Elle prend en paramètre un chemin de fichier valide. Notez que si vous appelez saveToDisk() sans paramètre, odtPHP effectuera simplement une sauvegarde interne du document en cours de génération.

The method saveToDisk() saves the result on the server. It takes as parameter a valid path to the file. Note that if you call saveToDisk() without parameter, odtPHP will just perform an internal saving of the current document.