Introduction

PclTemplate is an easy template management class.

The template can be any text based file, or string. The generated result will use the parsed template and a data structure (arrays) to fill the tokens.

PclTemplate support conditional block of information (if/endif) and also loops (list/endlist). A template can also include an other template.

Quick Start Guide

1 - Introduction

PclTemplate introduce different kind of tokens.
These tokens are :
- Simple tokens
- List/table/array tokens
- Conditional tokens
  - Include tokens

2 - Using Simple Tokens

The following is an example of a template file with simple tokens :

file 'model-simple.htm' :

```
<html>
 <head>
  <title>PclTemplate - Template Sample File - <!--(page_name)--></title>
 </head>
 <body>
  Hello <!--(token:user_name)--> !
 </body>
</html>
```

Notes :
- the keyword 'token' can be implicit. When delimiters are found, without any explicit type, the 'token' token type is implicitly understand.

The script to generate the page from the template will be :

file 'sample-simple.php' :

```
require_once('pcltemplate.class.php');

// ----- Create the template object
$v_template = new PclTemplate();
 // ----- Parse the template file
$v_template->parseFile('model-simple.htm');
 // ----- Prepare data
$v_att = array();
// ----- Set the values of the simple tokens
$v_att['page_name'] = 'First Generated Page';
$v_att['user_name'] = 'Vincent Blavet';
 // ----- Generate result in a string
$v_result = $v_template->generate($v_att, 'string');
 // ----- Display result
echo $v_result;
```

3 - Using a List Token Block

The following is an example of a template file with a list token block :

file 'model-list.htm' :

```
<html>
 <head>
```

```
   <title>PclTemplate - Template Sample File - <!--(page_name)--></title>
  </head>
  <body>
   Hello <!--(user_name)--> !     <br>       <!--(list:users)-->     <table border="1">       <tr>
    <td>First Name</td>
      <td>Last Name</td>
    </tr>       <!--(item)-->       <tr>       <td><!--(first_name)--></td>
      <td><!--(last_name)--></td>
    </tr>       <!--(enditem)-->     </table>     <!--(endlist)-->  </body>
</html>
```

The script to generate the page from the template will be :

file 'sample-list.php' :

```
require_once('pcltemplate.class.php');

// ----- Create the template object
$v_template = new PclTemplate();
   // ----- Parse the template file
$v_template->parseFile('model-list.htm');
  // ----- Prepare data
$v_att = array();
// ----- Set the values of the simple tokens
$v_att['page_name'] = 'First Generated Page';
$v_att['user_name'] = 'Vincent Blavet';
  // ----- Set the values of the list tokens
$v_att['users'][0]['first_name'] = 'Vincent';
$v_att['users'][0]['last_name'] = 'Blavet';
$v_att['users'][1]['first_name'] = 'Pierre';
$v_att['users'][1]['last_name'] = 'Martin';
// ----- Generate result in a string
$v_result = $v_template->generate($v_att, 'string');
  // ----- Display result
echo $v_result;
```

The result will be :

Hello Vincent Blavet !

First Name	Last Name
Vincent	Blavet
Pierre	Martin

There is some other options for managing a list with PclTemplate. Please look at

the advanced features.

4 - Using Conditional Blocks or Tokens

The following is an example of a template file with conditional blocks and simple conditional token :

file 'model-condition.htm' :

```
<html>
 <head>
  <title>PclTemplate - Template Sample File - <!--(page_name)--></title>
 </head>
 <body>
  Hello <!--(user_name)--> !  <br>  <!--(if:web_site)-->  His web site is : <!--(web_site)-->  <!--(endif)-->
   <br>
<!--(if:email_block)-->
   His email is :
<a href="mailto:<!--(email)-->"><!--(email)--></a>
    (alternative is
<!--(alternate)-->
)
<!--(endif)-->
   <br>
<!--(ifnot:not_here)-->
   Nothing to add !
<!--(endifnot)-->
 </body>
</html>
```

Notes :

- Inside a conditional block a token can be used several time.
- A conditional block allow you to have several values or other template blocks inside the block.
- A simple conditional token is in fact a conditional block with only a simple token inside with same name.

The script to generate the page from the template will be :

file 'sample-condition.php' :

```
require_once('pcltemplate.class.php');
```

```
// ----- Create the template object
$v_template = new PclTemplate();
    // ----- Parse the template file
$v_template->parseFile('model-condition.htm');
  // ----- Prepare data
$v_att = array();
// ----- Set the values of the simple tokens
$v_att['page_name'] = 'First Generated Page';
$v_att['user_name'] = 'Vincent Blavet';
  // ----- Set the values of the simple token condition
$v_att['web_site'] = 'www.phpconcept.net';    // ----- Set the values of the bloc condition
$v_att['email_block']['email'] = 'vincent@phpconcept.net';
$v_att['email_block']['alternate'] = 'support@phpconcept.net';

// ----- Generate result in a string
$v_result = $v_template->generate($v_att, 'string');
    // ----- Display result
echo $v_result;
```

The result will be :

Hello Vincent Blavet !
His web site is : www.phpconcept.net
His email is vincent@phpconcept.net (alternative is support@phpconcept.n

5 - Including files or templates

The following is an example of a template file which also the inclusion of an other template :

file 'model-inclusion.htm' :

```
<html>
  <head>
    <title>PclTemplate - Template Sample File - <!--(page_name)--></title>
  </head>
  <body>
    Hello <!--(first_name)--> <!--(last_name)--> !
    <br>
<!--(include:footer)-->
```

```
</body>
</html>
```

file 'model-footer.htm' :

```
<hr>
<!--(token:copyright)--> - <!--(token:author)-->
<br>
```

The script to generate the page from the template will be :

file 'sample-inclusion.php' :

```
require_once('pcltemplate.class.php');

// ----- Create the template object
$v_template = new PclTemplate();
  // ----- Parse the template file
if ($v_template->parseFile('model-inclusion.htm') != PCL_TEMPLATE_ERR_NO_ERROR) {
  echo "Error parsing file :<br>";
  echo nl2br($v_template->errorInfo());
  exit;
}
    // ----- Prepare data
$v_att = array();
// ----- Set the values of the simple tokens
$v_att['page_name'] = 'Sample Inclusion';
$v_att['last_name'] = 'Blavet';
$v_att['first_name'] = 'Vincent';

// ----- Include token  // The token is not a single value but an array with :  // - The filename to include ('filename').  // - The values of the tokens for the included file ('values').
$v_att['footer']['filename'] = 'model-footer.htm';
$v_att['footer']['values']['copyright'] = 'Copyright PhpConcept';
$v_att['footer']['values']['author'] = 'vblavet';
  // ----- Generate result in a string
$v_result = $v_template->generate($v_att, 'string');
  // ----- Display result
echo $v_result;
```

The result will be :

Hello Vincent Blavet !

Copyright PhpConcept - vblavet

Understanding PclTemplate Syntax

1 - Overview

PclTemplate generate a text based result from two sources.The first source is a template which describe the expected format of the result. The second source is the data used to generate a meaningfull result. The general advantage of a template mechanism is to generate multiple results with the same format, but with different content, or to be able to change the output format without changing the content source.

The template format of PclTemplate is based on a text based source (a file or a string) which contain blocks and keywords (tokens) that will be replaced by the meaningfull content. PclTemplate introduce different type of blocks and tokens, each of it having a different behaviour.

Template source is made of series of nested static text and **token blocks**. The generic syntax of the template is :

<static text 1>
[Start Delimiter]start_token_keyword:value[Stop Delimiter]
<mixed static text and token blocks>
[Start Delimiter]end_token_keyword[Stop Delimiter]
<static text 2>

"*static text*" can be any kind of text, as soon as it does not create confusion with "Start Delimiter" and "Stop Delimiter".
"*Start Delimiter*" and "*Stop Delimiter*" : Are a text patterns that will

permit a parsing of the template to identify the token blocks.
"*start_token_keyword*" : Is a reserved keyword that identify the begining of a token block. Each toekn block behave differently.
"*end_token_keyword*" : Is a reserved keyword that identify the end of a started token block. A token block must be closed by the valid corresponding reserved keyword (generally same name with 'end' prepend to it).
"*value*" : A uniq identifier, relative to the current block type, that identify the data source that will be used to generate the right content for this block.

An exemple of a template with a single token block :

```
<html>
<body>
  Hello !  <br>  <!--(if:condition_data)-->  The condition is here !  <!--(endif)-->
</body>
</html>
```

In this sample, the token block will be described if the content identified by "condition_data" is present in the data source.

2 - Token Blocks Types

2.1 - Basic Token Block

The most basic token block is not really a block. It is a simple token that will be replaced by a value.

```
<html>
<body>
  Hello !  <br>  <!--(if:condition_data)-->  The condition is here !  <!--(endif)-->
</body>
</html>
```

2.1 - IF Token Block

The IF Token Block, will allow to use a block if a condition is met. When the source data contains a field with name "condition_data", then the block will be used.

```
<html>
<body>
  Hello ! <br> <!--(if:condition_data)--> The condition is here ! <!--(endif)-->
</body>
</html>
```

2.1 - IFNOT Token Block

The IFNOT Token Block, will allow to use a block if a condition is not met. When the source data does not contain any field with name "condition_data", then the block will be used.

```
<html>
<body>
  Hello ! <br> <!--(ifnot:condition_data)--> The condition is not here !
<!--(endifnot)-->
 </body>
</html>
```

2.1 - LIST Token Block

The LIST Token Block will allow to use a list or an array of data from the data source. Inside the LIST Token Block a "ITEM" Token Block must be found and will be repeated for each row of the array.

```
<html>
<body>
  Hello ! <br> <!--(list:users)--> Header<br> <!--(item)--> Line : <!--(column
_1)-->
, <!--(
column_2)-->
<br>
  <!--(enditem)-->
   Footer<br>
<!--(endlist)-->
```

```
 </body>
</html>
```

2.1 - ITEM Token Block

The ITEM Token Block can only be used directly inside a LIST Token Block. The ITEM Token Block will be repeated for each item of the array of the source data.

```
<html>
<body>
  Hello ! <br> <!--(list:users)--> Header<br> <!--(item)--> Line : <!--(column
_1)--> , <!--(
column_2)-->
<br>
  <!--(enditem)-->
   Footer<br>
<!--(endlist)-->
 </body>
</html>
```

The "item" token block can only be used inside a "list" token block.

2.1 - IFEMPTY Token Block

The IFEMPTY Token Block is a special block used to select a block when an array (associated to a LIST Token Block) is present but empty.

```
<html>
<body>
  Hello ! <br> <!--(list:users)--> Header<br> <!--(ifempty)--> <i>No element
in the list</i><br>
 <!--(endifempty)-->

<!--(item)-->
   Line :
<!--(column_1)-->
,
<!--(column_2)-->
```

```
<br>
  <!--(enditem)-->
    Footer<br>
<!--(endlist)-->
 </body>
</html>
```

The "ifempty" token block can only be used inside a "list" token block.

2.1 - IFNOTEMPTY Token Block

The IFNOTEMPTY Token Block is a special block used to select a block when an array (associated to a LIST Token Block) is present and not empty.

```
<html>
<body>
   Hello !  <br>  <!--(list:users)-->  Header<br>  <!--(ifempty)-->  <i>No element in the list</i><br>
  <!--(endifempty)-->

<!--(item)-->
    Line :
<!--(column_1)-->
,
<!--(column_2)-->
(*see note)<br>
  <!--(enditem)-->

<!--(ifnotempty)-->
      <i>(*)Note : This is displayed only when there is at least one row</i><br>
  <!--(endifnotempty)-->
    Footer<br>
<!--(endlist)-->
 </body>
</html>
```

The "ifnotempty" token block can only be used inside a "list" token block.

2.1 - INCLUDE Token

The INCLUDE Token is used to include a template inside a template.

```
<html>
 <head>
  <title>PclTemplate - Template Sample File - <!--(page_name)--></title>
 </head>
 <body>
  Hello <!--(first_name)--> <!--(last_name)--> !
  <br>
<!--(include:footer)-->
</body>
</html>
```

The INCLUDE token can be used to include any text file with no template tokens.

Advanced Features

1 - Using string rather than file

The template can be a string rather than a file.

See the sample bellow :

```
file 'sample-string.php' :
require_once('pcltemplate.class.php');

// ----- Create the template object
$v_template = new PclTemplate();

// ----- Prepare a template string
$v_str = '
  <document name="<!--(doc_name)-->" size="<!--(doc_size)-->">
    <city name="<!--(city)-->"/>
```

```
  </document>
';

// ----- Parse the template file
$v_template->parseString($v_str);

// ----- Prepare data
$v_att = array();
$v_att['doc_name'] = 'document.txt';
$v_att['doc_size'] = '12';
$v_att['city'] = 'Paris';

// ----- Generate result    $v_result = $v_template->generate($v_att, 'string');
   // ----- Display result
echo '<pre>';
echo htmlentities($v_result);
echo '</pre>';
```

2 - Changing the delimiters

The delimiters can be dynamically modified.

By default PclTemplate use the following "HTML compatible" delimiters : "<!--(" and ")-->". the idea of these default delimiters are to show the template tokens as HTML comments, and then be compatible with WYSIWYG HTML editors.

To modify the delimiters use the following methods :

file 'sample-delimiters.php' :

```
require_once('pcltemplate.class.php');

// ----- Create the template object
$v_template = new PclTemplate();

// ----- Prepare a template string
```

```
$v_str = '
  <document name="--[doc_name]--" size="--[doc_size]--">
    <city name="--[city]--"/>
  </document>
';

// ----- Change the delimiters
$v_template->changeDelimiters('--[',']--');

// ----- Parse the template file
$v_template->parseString($v_str);

// ----- Prepare data
$v_att = array();
$v_att['doc_name'] = 'document.txt';
$v_att['doc_size'] = '12';
$v_att['city'] = 'Paris';

// ----- Generate result    $v_result = $v_template->generate($v_att, 'string');
    // ----- Display result
echo '<pre>';
echo htmlentities($v_result);
echo '</pre>';
```

3 - Implicit condition

PclTemplate use implicit condition. When a simple token is defined in the template, but not set in the data array, the generated result, will just ignore the token.

In the same way, if a token block is defined in the template, but not present in the data array, the generated result will skip the block. This is a implicit way to do an 'if' condition.

4 - Nested blocks

Starting with version 0.4 PclTemplate support nested blocks. For example a 'list' block support 'if' or 'ifnot' conditions inside it. Or a 'if' or 'ifnot' condition block support 'list' blocks inside it.

Error Management

PclTemplate returns error messages :

- PCL_TEMPLATE_ERR_NO_ERROR(1) :
  No error during operation.
- PCL_TEMPLATE_ERR_GENERIC(0) :
  An undetailled error occurs during operation. The error string might give more informations.
- PCL_TEMPLATE_ERR_SYNTAX (-1) :
  A syntax error while parsing a template occur. The error string gives the template filename and the line.
- PCL_TEMPLATE_ERR_READ_OPEN_FAIL (-2) :
  Unable to open a file in read mode.
- PCL_TEMPLATE_ERR_WRITE_OPEN_FAIL (-3) :
  Unable to open a file in write mode.
- PCL_TEMPLATE_ERR_INVALID_PARAMETER (-4) :
  Invalid parameter used in a method call.

PclTemplate store error messages and codes in the object. The method errorInfo() returns a string with all the errors separated by a newline.

```
// ----- Error management while parsing file
if ($v_template->parseFile('template_file.htm') != PCL_TEMPLATE_ERR_NO_ERROR) {
  die("Error parsing file :<br>".nl2br($v_template->errorInfo());
}    // ----- Error management while generatign result  $v_result = $v_template->generate($v_att, 'string');
if ($v_result === 0) {    die("Error generating file :<br>".nl2br($v_template->errorInfo()));
```

```
}
```

Notice the use of '===' and not '==' to check the result value of generate()

FAQ - Frequently Asked Questions

Nothing yet ;-)

Limitations & Prerequisits

PclTemplate is storing all the template file in memory, so PclTemplate might have limitation for very large template files.

When using included templates, the syntax is checked only when generating the template. Error reports are then very limited.

License & Copyrights

PclTemplate is released by Vincent Blavet (vincent@phpconcept.net) under GNU LGPL License. So you can use it at no cost.

HOWEVER, if you release a script, an application, a library or any kind of code including PclTemplate, YOU MUST :
- Release your work under GNU/LGPL license (free software),
- You must indicate in the documentation (or a readme file), that your work include PclTemplate script, and make a reference to the author and the web

site http://www.phpconcept.net

I will also appreciate that you send me an email (vincent@phpconcept.net), just to be aware that someone is using PclTemplate somewhere in the world.

For more information on GNU/LGPL licensing : http://www.gnu.org

Warnings

This application and the associated files are non commercial, non professional work.
It should not have unexpected results. However if any damage is caused by this software the author can not be responsible.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

The use of this software is at the risk of the user.