

Was ist TPLE ?

Zunächst ein Abkürzung für Templateengine und damit das Synonym für unsere eigene Templateengine.

Sie basiert auf der Erkenntnis, das die Templateengine Smarty, die wir für die 1 er Serie eingesetzt haben, eigentlich völliger Unsinn ist.

Unsinn aus folgenden Gründen:

- Smarty verwendet eine eigene Scriptsprache, die man erst einmal lernen muss.
- Smarty compiliert diese Scripte und macht dann ein PHP Script daraus
- Smarty ist für den DB Betrieb nur sehr umständlich zu nutzen
- Smarty verbraucht viel RAM und Zeit

PHP selbst wurde als Templateengine entwickelt und verfügt über eine riesige Menge an Funktionen.

PHP steht auf dem Lehrplan eines jeden Webdesigners.

Der Smartysyntax müsste zusätzlich erlernt werden.

Warum aus einer Scriptsprache durch Compilierung ein PHP Script machen, wenn es auch direkt geht.

Warum muss man das umständliche Handling im DB – Betrieb akzeptieren ?

Warum sollte man RAM – und Zeitnachteile in Kauf nehmen ?

Warum sollte man nicht die unglaublichen Möglichkeiten des kompletten PHP Umfanges nutzen können ?

Und so sieht ein Template mit TPLE aus:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <meta http-equiv="cache-control" content="no-cache" />
  <meta http-equiv="expires" content="3600" />
  <meta name="revisit-after" content="2 days" />
  <meta name="robots" content="index,follow" />

  <meta name="publisher" content="Jan Czarnowski" />
```

```
<meta name="copyright" content="Jan Czarnowski" />
<meta name="author" content="PowerSite" />
<meta name="distribution" content="global" />
<meta name="description" content="Diese Seite wurde mit PowerSite erstellt." />
<meta name="keywords" content="PowerSite,PowerCMS,CMS,Smarty" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<?php $this->show('xajaxheader');?>
<?php $this->show('c_meta_name');?>
<link rel="stylesheet" type="text/css" media="all" href="css/layout2setup.css" />
<link rel="stylesheet" type="text/css" media="all" href="css/layout2text.css" />
<link rel="icon" type="image/x-icon" href="layoutimages/favicon.ico" />
<link rel="alternate" type="application/rss+xml" title="RSS - News" href="mininewsrss.php"/>
<title><?php $this->show('c_titel');?></title>
</head>

<body>
<!-- Main Page Container -->
<div class="page-container">

<!-- A. HEADER -->
<div class="header">

<!-- A.1 HEADER TOP -->
<div class="header-top">
<div class="round-border-topleft"></div><div class="round-border-topright"></div>

<!-- Sitelogo and sitename -->
<a class="sitelogo" href="#" title="Home"></a>
<div class="sitename">
<h1><a href="<?php echo ROOT_URL;?>/index.php?<?php echo QUERY_VAR?>=home"
title="Go to Start page">PowerCMS • 2.0</a></h1>

</div>
```

```
<!-- Navigation Level 1 -->
<div class="nav1">
  <ul>
    <li><a href="#" title="Unsere Links">Links</a></li>
    <li><a href="#" title="Sitemap">SiteMap</a></li>
    <li><a href="#" title="Impressum">Impressum</a></li>
  </ul>
</div>
</div>
```

```
<!-- A.3 HEADER BOTTOM -->
<div class="header-bottom">
  <div class="nav2">
    <?php $this->display(TEMPLATE_DIR.'menu_horizontal_ebene1.tpl.php');?>
  </div>
</div>
```

```
<!-- A.4 HEADER BREADCRUMBS -->
```

```
<!-- Breadcrumbs -->
<div class="header-breadcrumbs">
```

```
<?php $this->display(TEMPLATE_DIR.'breadcrumbs.tpl.php'); ?>
  <!-- Search form -->
  <div class="searchform">
    <?php global $piajax; $piajax-
    >piajax_form_remote(array('url'=>'suchen.php','update'=>#suchdiv','id'=>'suchform'));?>
    <fieldset>
      <input class="field" type="text" name="pisearch" value="Suchbegriffe ..."
      onblur="if(this.value=='') this.value='Suchbegriffe ...';" onfocus="if(this.value=='Suchbegriffe ...')
      this.value=";" />
      <input type="hidden" name="action" value="showresults" />
      <input type="hidden" name="ajax" value="1" />
      <input type="submit" value="Los" name="button" class="button" />
```

```

</fieldset>
</form>
  </div>
</div>
</div>

<!-- END COPY here -->

<!-- B. MAIN -->
<div class="main">

  <!-- B.1 MAIN NAVIGATION -->
  <div class="main-navigation">

    <!-- Navigation Level 3 -->
    <div class="round-border-topright"></div>
<?php if (count($this->VARS['v_menu'])>1): ?>
  <h1 class="first">Sub-Navigation</h1>

<!-- Navigation with grid style -->
<?php $this->display(TEMPLATE_DIR.'menu_vertical_ebene_2_3.tpl.php');?>
<div id="werbung">
  <h1>News</h1>
  <?php $this->plugin('modernnews',array('detailseite'=>'details','sortierung'=>'kategorie'));?>
  <hr /><br />
<?php $this->show('c_block1');?>
</div>
<?php else:?>
  <h1 class="first">News</h1>
  <?php $this->plugin('modernnews',array('detailseite'=>'details','sortierung'=>'kategorie'));?>
<div>
  <hr /><br />
<?php $this->show('c_block1');?>
</div>

```

```
<?php endif;?>
</div>
  <!-- B.1 MAIN CONTENT -->
  <div class="main-content">
    <div id="suchdiv"></div>
<div id="allgemein"></div>
<div id="main">
<?php $this->show('c_inhalt');?>
</div>
<p>
  Bearbeitung dieser Seite am: <?php echo date($this->returnText('FORMAT_DATUM'),$this-
>VARIS['c_modified_date']);?><br />
  </p>
  <?php $this->display(TEMPLATE_DIR.'nextprev.tpl.php');?>
</div>
</div>
<!-- C. FOOTER AREA -->
  <div class="footer">
    <p>Copyright &copy; 2009 - <?php $this->printText('HINWEIS5');?>-</p>
    <p class="credits"><?php $this->printText('HINWEIS3');?> <a href="http://1234.info/"
title="Designer Homepage">1234.info</a> | <?php $this->printText('HINWEIS2');?> <a
href="http://powercms.org" title="PowerCMS">Darren</a> | <?php $this-
>printText('HINWEIS4');?> <a href="http://.powercms.org" title="PowerCMS">PowerCMS</a> |
  <a href="http://validator.w3.org/check?uri=referer" title="Validate XHTML
code">XHTML 1.0</a></p>
  </div>
</div>
</body>
</html>
```

Wer Smarty kennt wird auch erkennen das die praktischen Unterschiede im Einsatz nur sehr gering sind.

Aber – man kann buchstäblich alles machen, was PHP anbietet, das geht bei Smarty nicht unbedingt oder wieder nur sehr umständlich.

Die Templates können mit TPLE auf einfachste Art und Weise mit unglaublicher Flexibilität versehen werden, denn unter TPLE ist buchstäblich alles PHP, ob die Inhalte nun aus einer DB – Tabelle oder aus einer Datei kommen – das spielt keine Rolle.

Inhalte die aus der DB kommen und somit als Variable vorliegen können einfach und direkt verarbeitet werden – ein ganz anderes Verhalten als bei Smarty.

Weil Smarty den Kompilierungszeitpunkt mit dem Zeitpunkt der letzten Bearbeitung vergleichen muss, ist ein DB Betrieb mit Smarty denkbar umständlich, mit TPLE jedoch extrem einfach.

Und – last but not least – Smarty Plugins sind sehr leicht auf TLE umzuarbeiten.

Wer TPLE im Vergleich zu Smarty kennt, der weiss was hier gemeint ist.

Noch nie war es so einfach mit einer Templateengine zu arbeiten.

Die Templateengine TPLE für Serie 2 hat einige Abweichungen zu der von PowerSite eingesetzten Version.

Es liegen folgende Funktionen vor:

1. public function __construct(\$pdir,\$tdir,\$sprache,\$ldir,\$ajax=null)
2. function bs(\$name,\$var,\$antwort="",\$class="")
3. function loadTransFile()
4. function printText(\$txt_id,\$spezial="")
5. function returnText(\$txt_id,\$spezial="")
6. function show(\$var)
7. function assign(\$key, \$val=null,\$compile=false)
8. function display(\$tpl_file,\$script="")
9. function vardisplay(\$code)
10. function fetch(\$tpl_file,\$script="")
11. function varfetch(\$code)
12. function doublevarfetch(\$code)
13. function plugin(\$func_name)

Mit diesen Funktionen lässt sich alles machen was für PowerCMS notwendig ist.

public function __construct(\$pdir,\$tdir,\$sprache,\$ldir,\$ajax=null)

Das ist der Konstruktor der TPLE Klasse und somit keine eigentliche Funktion.

Der Einsatz bei PowerCMS erfolgt für das Frontend in der include.php wie folgt:

```
$tpl = new TPLE(PLUGINS_DIR,TEMPLATE_DIR,$sprache,ROOT_PATH.LANG_DIR,$piajax);
```

\$pdir enthält den Pfad zu den Plugins.

\$tdir enthält den Pfad zu den Templates

\$sprache ist die aktuelle Sprache

\$ldir ist der Pfad zu den Internationalisierungsdateien für Templates

\$ajax enthält das Objekt zur Klasse Piajax

function bs(\$name,\$var,\$antwort="",\$class="")

Diese Funktion gibt ein Selectfeld zurück deren Basis Arrays mit numerischen Indexe sind.

\$name ist der Name des Selectfeldes.

\$var ist ein numerisches Array mit den nicht sichtbaren Variablen

\$antwort ist ein numerisches Array mit den sichtbaren Variablen

\$class kann eine CSS Klasse enthalten.

function loadTransFile()

Das ist normal eine interne Funktion die vom Konstruktor aufgerufen wird.

Sie verarbeitet die aktuelle Internationalisierung für Templates bzw. Stellt die Datei zur Verfügung.

function printText(\$txt_id,\$spezial="")

Gibt im Rahmen der Internationalisierung den Text der aktuellen Sprache gemäss \$txt_id aus.

Wenn für \$txt_id kein extra Wert in der Auflösung vorhanden ist, wird der Wert von \$txt_id ausgegeben.

\$spezial dient den Fällen, wo die Struktur der Sprachdatei eine Stufe zusätzlich in der Auflösung hat (wird normal nicht benötigt).

function returnText(\$txt_id,\$spezial="")

Funktioniert wie printText jedoch führt keine Ausgabe sondern eine Rückgabe aus.

function show(\$var)

Gibt den Wert einer TPLE – Variable aus, die mit \$var bezeichnet ist.

Gibt es \$var nicht wird " ausgegeben.

function assign(\$key, \$val=null,\$compile=false)

Weist TPLE einen Key (\$key) mit dem Inhalt \$val zu.

Ist \$compile = true dann wird \$val mit der Funktion varfetch vor der Zuweisung verarbeitet.

So z.B. im Einsatz in der index.php (Frontend):

```
$tocompile=array('metadata','inhalt','block1','block2','block3','block4');
```

.....

```
foreach ($r as $key=>$c)
    $tpl->assign('c_'.$key,$c,in_array($key,$stocompile));
```

Das bedeutet hier in der Praxis das die Datenteile metadata,inhalt, block1 bis block4 TPLE – Teile enthalten können, die zudem zusätzlich noch ein mal einen durch einen TPLE Aufruf darin einen weiteren TPLE Anteil erzeugen können.

Da das Template bereits mit diesen vorverarbeiteten Teilen versehen ist bevor es selbst einmal gefetcht wird ist die Existenz aller Inhalte sichergestellt.

function display(\$tpl_file,\$script=)

Führt ein require_once durch mit \$tpl_file.

\$script wird nur in der Entwicklungszeit benötigt um einen Hinweistext auszugeben, wenn \$tpl_file nicht vorhanden ist.

function vardisplay(\$code)

Führt den \$code direkt aus und bringt ihn zur Anzeige.

function fetch(\$tpl_file,\$script=)

Öffnet \$tpl_file, verarbeitet sie und gibt den erzeugten Wert zurück.

Ist \$tpl_file nicht vorhanden, wird ein Fehlertext zurück gegeben.

function varfetch(\$code)

Wie vardisplay jedoch erfolgt eine Rückgabe.

function doublevarfetch(\$code)

Wie varfetch jedoch erfolgt ein doppelter Durchlauf. Das Ergebnis aus dem ersten Durchlauf wird erneut verarbeitet.

Macht Sinn, wenn aus dem ersten Durchlauf weiterer TPLE – Kode entsteht der dann aber ja noch nicht verarbeitet ist, sondern nur als Kode vorliegt.

function plugin(\$func_name)

Mit dieser Funktion werden Plugins aufgerufen.

\$func_name enthält den Namen des Plugins.

Parameter werden als zweiter Parameter und als Array eingesetzt.

Beispiel:

```
<?php
```

```
$this->plugin('modernnews',array('detailseite'=>'details','sortierung'=>'kategorie','show'=>'1','template'=>TEMPLA  
TE_DIR.'modernnews/showdetail.tpl-php'));
```

```
?>
```

Die Plugins sind ganz normale PHP Funktionen, die Plugins Ordner erwartet werden (Siehe Konstruktor).

Die internen Variablen

Es gibt nur wenige Variable in TPLE:

```
var $PLUGIN_DIR;  
Var $TEMPLATE_DIR;  
var $VARS;  
var $xml;  
var $langpfad;  
var $pia;
```

PLUGIN_DIR

Enthält den über den Konstruktor gelieferten Wert mit dem Pfad zum Pluginverzeichnis.

TEMPLATE_DIR

Enthält den über den Konstruktor gelieferten Wert mit dem Pfad zum Templateverzeichnis.

VARS

Ist die Sammelvariable für die an TPLE zugewiesenen Variable und Inhalten dazu.

Die Werte werden mit `$tpl->VARS['name']` oder intern mit `$this->VARS['name']` angesprochen.

xml

Nimmt die Inhalte der Internationalisierungsdaten über `simplexml` auf.

langpfad

Enthält den Pfad zu Internationalisierungsdatei, der Wert wird ber den Konstruktor geliefert.

pia

Enthalt null oder das Objekt zu PiAjax. Ansparche dann in Templates `$this->pia->function`.

Grundsätzlicher Aufbau eines Plugins

```
if (!defined('INSIDE_POWER_CMS')) die ('DIRECT ACCESS FORBIDDEN');
```

```
function name($params)
{ global $db,$config,$tpl,$func;
//Aktionen
}
```

INSIDE_POWER_CMS wird vorher einmal definiert und zwar vor der Nutzung eines Plugins.

Liegt sie nicht vor wird eine Ausführung abgebrochen.

Es wird also ein Direktaufruf damit unterbunden.

Die globalen Variablen werden nach Bedarf eingebunden.

Wird der Datenbankzugang benötigt wird \$db notwendig sein.

Wird die config benötigt muss man sie einbinden.

Werden andere Dinge aus der aktiven TPLE Klasse benötigt, muss man \$tpl haben.

Wird RemoveXSS benötigt, die in \$func enthalten ist, muss man auch diese so einbinden.

Parameter werden normal als Array übergeben.

Man kann aber auch eine Reihe von Einzelwerten definieren.

Alles ist normales PHP.

Wird eine Funktion in Zusammenhang mit Ajax benutzt muss man darauf achten, das ein Plugin dann keine Ausgabe erzeugt sondern eine Rückgabe.

Die Funktion wird als File in den Plugins – Ordner unter seinem Namen abgelegt – hier also name.php

Wozu eine Templateengine ?

Eine PHP Anwendung besteht im Prinzip aus den Teilen welche die reine Anwendung steuern , Daten sammeln und solchen welche etwas zur Ausgabe bringen.

Wenn man nun die gesammelten Daten einer Templateengine zuweist und dem Designer die Inhalte und die dahinter steckende Logik der Daten bekannt sind, kann er (der Designer) unabhängig von dem steuernden Teil die Ausgabe so gestalten wie er möchte.

Würde man den Ausgabeteil direkt in den Steuerteil setzen, also damit vermischen, wäre eine Trennung schwerlich möglich.

Ein Designer müsste also in den Steuerteil eingreifen und könnte damit diesen gefährden und sogar unbrauchbar machen.

Mit einer Templateengine erhält er enorme Möglichkeiten die Daten zu präsentieren.

Da in der Regel neben einem Haupttemplate auch eine Anzahl von kleineren Templates eingesetzt werden ist der Aufwand sehr gering in Teilbereichen eine Änderung auszuführen.

Das gilt selbstverständlich auch für das Haupttemplate.

Die Variablen von TPLE sind praktisch Platzhalter die in Templates gesetzt werden um dort real die Inhalte darzustellen.

Aber eine gute Templateengine geht noch weit darüber hinaus.

Über Plugins kann der Funktionsumfang der Hauptanwendung beliebig erweitert werden in dem man mit Hilfe dieser Plugins Nebenaufgaben erledigt, die ansonsten von der Steuerlogik übernommen werden.

Modulsysteme die man von anderen Titeln kennt haben damit ein erhebliches Problem.

Modulsysteme können nur erkennen ob ein Module installiert ist, nicht aber ob und wo es praktisch eingesetzt wird.

Um aber für den Fall der Fälle Gewähr bei Fuss zu stehen, muss ein Modulsystem genau aus dem Grunde alles laden, was es an Modulen gibt.

Plugins , die auf einer Templateengine basieren, werden eingesetzt und zwar in Inhalten oder Templates genau da wo man es benötigt.

Die Templateengine bemerkt den Einsatz von Plugins und verarbeitet deren Funktion einfach mit.

Die Vorteile liegen auf der Hand:

- kein Moduleumfeld nötig
- Plugins sind klein und sparsam
- alles wird kompakter und wesentlich schneller
- alles kann beliebig ausgebaut werden ob nun von der Aufgabenseite oder von der Darstellungsseite her

Diese Pluginmöglichkeiten sind mit TPLE ideal gegeben, da es fast nichts gibt was besonders zu berücksichtigen wäre.

Und wer sich daran hält, das Ausgaben grundsätzlich über ein Template gemacht werden, der hat gewonnen und befindet sich auf der Sonnenseite.

Die Anzahl der möglichen Plugins ist völlig unbegrenzt.