vlibTemplate, vlibDate: Tutorial und Beispiele

Autor: Claus van Beek
Hauptseite: http://lamp.clausvb.de
Forum: Hilfe und Support

Table Of Contents

1. Einleitung

Template Engines (auch bekannt als Templateklassen) dienen dazu PHP- und HTML-Code zu trennen. Das hat mehrere Vorteile:

- Übersichtlichkeit: PHP liefert Daten und Logik, HTML übernimmt Ausgabe und Formatierung. Beides ist klar getrennt.
- Das Design einer Webseite kann mit WYSIWYG-Editoren (Frontpage, Dreamweaver) erstellt und verändert werden.
- Bei größeren Projekten können Programmierer und Webdesigner gleichzeitig an einem Projekt arbeiten ohne sich gegenseitig zu behindern.

Jeder PHP-Programmierer kennt Skripte, die sowohl PHP- als auch HTML-Code enthalten. Oft besteht das Skript dann aus mehreren

```
echo "<table width='80%'>\n\t<tr bgcolor='$farbe'>";
```

oder einer Mischung aus solchen ECHO-HTML-Anweisungen und HTML-Blöcken. Das Ergebnis ist mitunter schwer lesbar/verständlich.

Lesen Sie zur Thematik "Templates" auch ein Zitat von Johannes Gamperl.

Übersetzung aus der Dokumentation von vlibTemplate:

vlibTemplate ist ein PHP-Klasse, welche die Trennung von PHP- und HTML-Code zu einer einfachen und natürlichen Sache machen soll.

vlibTemplate benutzt die folgenden "markup tags": **<tmpl_var>, <tmpl_loop>, <tmpl_include>, <tmpl_if>** [und andere].

Eine Datei, die solche Tags enthält nennt man Template. Ein Template kann eine HTML-Datei sein, um sie im Web zu benutzen oder eine Textdatei, die als E-Mail versendet wird ... es gibt viele Möglichkeiten.

Die Templatedatei wird immer seperat vom PHP-Skript (das es benutzt/aufruft) abgespeichert, so kann ein [Web]Designer zum Beispiel das Template ändern ohne den ganzen PHP-Code durchgehen zu müssen.

(...)

[Die Klasse] versetzt Sie in die Lage, Design und Daten, die Sie mit PHP erzeugen, zu trennen.

Noch mal zum Mitschreiben:

- Ein Template ist (meistens) eine HTML-Datei, die HTML-Tags und Templatevariablen ("markup tags") - auch Platzhalter genannt - enthält.
- Diese Templatevariablen werden im PHP-Skript mit Werten gefüllt. Beispiel: "bsp_skript.php" füllt das "bsp_template.htm" mit Werten.

- Die Templatevariablen bilden damit die Schnittstelle zwischen Design und Programmierung.
- Hinweis: Abkürzungen für "Template": tmpl oder tpl

Die Klasse vlibTemplate lässt sich am einfachsten mit Beispielen erklären. Alle Beispiele sind ebenfalls im Internet unter lamp.clausvb.de zu finden. Sie wurden in diesem Tutorial leicht modifiziert, um Platz zu sparen. Anmerkung: Templateklassen werden auch "Template Engines" genannt.

Ich gehe in diesem Tutorial davon aus, dass die Grundlagen von PHP, MySQL und OOP (Objektorientierter Programmierung) bekannt sind. Vor allem sollten die OOP-Begriffe "Klasse", "Instanz / Objekt" (Ableitung einer Klasse) und "Methoden" (Funktionen einer Klasse) bekannt sein.

Theoretisch könnten Sie auch ohne Kenntnisse der OOP mit vlibTemplate arbeiten, aber ich rate davon ab.

Links zu dem Thema:

- PHP: verschiedene PHP-Tutorials
- OOP: #php/QuakeNet, DSP (reeg.net)
- (Mehrdimensionale) Arrays: Grundlagen zu Arrays, LOOP-Arrays in vlibTemplate

Alle unten aufgeführten Beispiele sollten sofort ausprobiert werden, um Ergebnis zu sehen und das Gelesene damit besser zu verstehen.

Vorgehen:

- Beispiel im Browser ausführen
- Quelltext des Beispiels ansehen (Internet Explorer: rechte Maustaste - "Quelltext anzeigen")
- PHP- und Template-Code nach und nach durchgehen => in einem Editor das PHP-Skript (*.php) und die Templatedatei (*.htm) öffnen

Damit sollte man ein Grundverständnis erlangen, wie *vlibTemplate* und *vlibDate* funktionieren.

2. Erstes Beispiel mit zwei Templatevariablen

Unser erstes Beispiel zeigt ein einfaches PHP-Skript und das dazugehörige Template.

PHP-Skript

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/basic.htm');

$tmpl->setvar('title_text', 'TITLE: This is the vLIB basic example ...');
$tmpl->setvar('body_text', 'BODY: This is the message set using setvar()');

$tmpl->pparse();
```

Dieses Beispiel zeigt die Grundstruktur von Templating: Im PHP-Code steht kein einziger HTML-Befehl. Richtlinie: Im PHP-Code am besten keine HTML-Tags verwenden.

require_once 'vlib/vlibTemplate.php';
: Um mit vlibTemplate arbeiten zu können muss die Klasse als erstes inkludiert werden. "Inkludieren" ist ein umgangsprachlicher Programmierbegriff, der im Zusammenhang mit der PHP-Funktion "include" verwendet wird. Man kann man mit dem Befehl include() inkludieren oder mit require(). Ich bevorzuge require_once() da jedes benötigte Skript nur einmal inkludiert wird und require_once() einen FATAL ERROR liefert, wenn die Datei nicht gefunden wird. Erst nach dem require_once() sind die vlibTemplate-Methoden dem Skript bekannt.

$tmpl = new vlibTemplate('tmpl/basic.htm');
: Es wird die Instanz $tmpl der Klasse vlibTemplate generiert. Damit kann jetzt die Methode "setVar" über die Instanz $tmpl angesprochen werden.

  In der Datei "tmpl/basic.htm" sind die ganzen HTML-Tags hinterlegt (siehe unten), die für die Aufbereitung der Seite gebraucht werden.

$tmpl->setvar('title_text', 'TITLE: This is the vLIB basic example ...');
: Der Templatevariablen "title_text" wird ein beliebiger Text zugewiesen. Dieser wird nach erfolgreichem Abarbeiten des Skriptes in der Titelzeile zu sehen sein. Dieses gilt ebenfalls für "body_text".

  Hinweis: Sollte in diesem Text ein HTML-Tag verwendet werden, so wird dieser NICHT ausgewertet. Dazu aber später mehr ...

$tmpl->pparse();
: Diese Methode sorgt dafür, dass die im PHP-Skript definierten Inhalte an das Template "geparst" werden. Das heißt, die Templatevariablen werden mit Inhalten gefüllt. Danach wird das Template angezeigt. Dies ist also in vielen Fällen der abschließende Befehl des PHP-Skripts. Danach sollte zumindest keine Ausgabe mehr per echo() oder print() erfolgen.

Template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
   <title>{tmpl_var name='title_text'}</title>
   <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>

<body>

<p>{tmpl_var name='body_text'}</p>

</body>
</html>
```

In der Templatedatei ("tmpl/basic.htm") sind zwei Platzhalter bzw. zwei Templatevariablen hinterlegt. Der Aufruf "$tmpl->pparse();" füllt die beiden Templatevariablen mit den zuvor definierten Inhalten und zeigt das Ergebnis im Browser an.

Manchmal wird es nötig sein, Zellen auszugeben, die HTML-Tags enthalten. Zum Beispiel wenn Datenbank Einträge mit nl2br($db_eintrag) abgespeichert wurden. In dem Fall setzt man im Template nicht:

<td valign="top">{tmpl_var name='db_eintrag'}</td>

sondern

<td valign="top">{tmpl_var name='db_eintrag' escape='none'}</td>

Für das Attribut "escape" gibt es neben "none" noch andere Werte bzw. Einstellmöglichkeiten. Wie oben erwähnt bewirkt "none" bei diesem Beispiel, dass alle von nl2br() erzeugten HTML-Tags geparst werden. Würde man nicht "none" sondern "html" übergeben, dann würden alle eckigen Klammern durch die HTML-Sonderzeichen "<" und ">" ersetzt. Man kann diesen Parameter auch generell in der "vlibIni.php" setzen. Weitere Hinweise kann man im Forum finden.

3. TMPL_IF mit Boolean und Konstanten

Oft wird die HTML-Ausgabe nach verschiedenen Zuständen entschieden. Ist der User ein Administrator oder ein einfacher Benutzer beispielsweise. Oder wenn eine bestimmte Variable gesetzt ist, muss ein bestimmtes Wort in roter Farbe geschrieben werden.

PHP-Skript

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/tmpl_if.htm');

$boolean = (isset($_GET['if_condition'])) ? 1 : 0;

$tmpl->setvar('boolean', $boolean);
$tmpl->setvar('if_condition', $_GET['if_condition']);

$tmpl->pparse();
```

$boolean = (isset($_GET['if_condition'])) ? 1 : 0;

Dies ist eine vereinfachte IF-Struktur, die überprüft ob der GET-Parameter "if_condition" in der URL gesetzt wurde oder nicht. Wenn dieser Parameter einen Wert enthält, wird "$boolean" zu 1 ansonsten zu 0. Informationen zu den sogenannten "Superglobals" und zu Formularverarbeitung kann man bei php.net finden.

$tmpl->setvar('boolean', $boolean);

Die Methode "setvar" setzt die Templatevariable "boolean", die im CSS-Teil des Templates die Farbe (CSS-Eigenschaft "color") auf lila oder grau setzt.

$tmpl->setvar('if_condition', $_GET['if_condition']);
Damit die IF-Struktur im Template auf den GET-Parameter "if_condition" reagieren kann, muss dass Template diesen Wert übergeben bekommen. Der Parameter kann einfach über "tmpl_if.php?if_condition=10" übergeben werden.

Template

```
(...)
      <style type="text/css">
      body
      {
         font-family: Arial;
         font-size: 0.9em;
         color: <tmpl_if name='boolean'>purple<tmpl_else>gray</tmpl_if>;
      }
      pre { color: black; font-size: 1.1em; }
   </style>
</head>

<body>

<p>This text is gray without "if_condition" set. Click
<a href="tmpl_if.php?if_condition=1">here</a> to make it purple.
<tmpl_if name='if_condition' op='>' value='5'><span
style="color: green">"if_condition" is more than 5.</span><tmpl_else>
"if_condition" is not set or below (equal to) 5.</tmpl_if></p>
```

<tmpl_if name='boolean'>
Dies ist die einfachste Form einer TMPL_IF. Diese funktioniert folgenderdermaßen: Wenn die Templatevariable "boolean" auf 1 gesetzt wurde, wird die CSS-Eigenschaft "color" auf purple gesetzt. Ist "boolean" gleich 0, dann wird über TMPL_ELSE gray verwendet.

<tmpl_if name='if_condition' op='>' value='5'>
Ein TMPL_IF kann auch Vergleichsoperatoren enthalten. Im "value" sind nur Konstanten und keine anderen Templatevariablen möglich. Mehr zu Vergleichsoperatoren ...

4. Formulare mit vlibTemplate

Jetzt wollen wir unser erstes praktisches Beispiel in Angriff nehmen; das Skript "form.php" soll mit Templates realisiert werden.

PHP-Skript ohne vlibTemplate

```
if (isset($_POST['input']))
{
   echo '<p>You typed <b>' . $_POST['input'] . '</b>.</p>';
}
else
{
echo<<<formular
   <form action="form.php" method="post">
      <input name="input">
      <input type="submit">
   </form>
formular;
}
```

Die IF-Struktur ...
... überprüft, ob $_POST['input'] gesetzt wurde. Wenn ja, wird $_POST['input'] ausgeben, ansonsten das Formular (zur Eingabe von $_POST['input']).

echo<<<formular
Das Formular wird mit der mit Hilfe von echo() und der "Heredoc-Syntax" ausgegeben.

PHP-Skript mit vlibTemplate

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/form_tmpl.htm');

$tmpl->setVar('input', $_POST['input']);

$tmpl->pparse();
```

Die dargestellten Befehle sind alle bekannt und erst mit Betrachtung des Templates wird klar, wie die Umsetzung stattfindet.

Template

```
<tmpl_if name='input' op='<>' value=''>
    <p>You typed <b>{tmpl_var name='input'}</b>.</p>
<tmpl_else>
    <form action="form_tmpl.php" method="post">
        <input name="input">
        <input type="submit">
    </form>
</tmpl_if>
```

<tmpl_if name='input' op='<>' value="">

Im Template wird eine IF-Struktur verwendet, die überprüft ob 'input' leer oder gesetzt ist. Wenn sie nicht leer ist wird "You typed {tmpl_var name='input'}." ausgegeben, ansonsten das Formular. Das TMPL_IF prüft ob 'input' ungleich einem leeren String ist.

5. TMPL_INCLUDE

Templates können modular aufgebaut werden. Wenn Sie zum Beispiel eine bestimmte Tabelle immer wieder verwenden wollen, können Sie diese in ein extra Template speichern und dann beliebig oft inkludieren.

PHP-Skript

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/include.htm');

$tmpl->setVar('normal_part', 'tmpl/include.htm: This
   demonstrates the include functionality.');
$tmpl->setVar('include_part', 'tmpl/include_part.htm');
$tmpl->setVar('date', date("d.m.Y"));

$tmpl->pparse();
```

Wie man sehen kann, unterscheidet sich dieses PHP-Skript nicht sehr stark von anderen Skripten die Templatebefehle enthalten. date("d.m.Y") haben wir zuvor allerdings noch nicht übergeben. Wie bei anderen Funktionen / Methoden mit, kann die Methode "setVar" nicht nur konstante Werte/Strings übermitteln, sondern sie unterstützt auch direkte Funktionsaufrufe oder Variablen. Die Funktionsweise des TMPL_INCLUDE zeigt sich jedoch erst in der Templatedatei:

Template

```
{tmpl_var name='normal_part'}

<br><br>

If you would have a "site.htm" including the following
examples for a modular template structure:

<ul>
   <li>&lt;tmpl_include file='header.htm'&gt;</li>
   <li>&lt;tmpl_include file='body.htm'&gt;</li>
   <li>&lt;tmpl_include file='footer.htm'&gt;</li>
</ul>

<tmpl_include file='vlibTemplate_include_part.htm'>
```

```
</body>

vlibTemplate_include_part.htm
I am the footer of the page, this is from <b>{tmpl_var name='include_part'}</b>
included using <tmpl_include>.<br>

<br><br>

Date (ISO 8601): {tmpl_var name='date'}
```

Der TMPL_INCLUDE führt also dazu, dass der Inhalt der inkludierten Datei, genau an der Stelle steht, wo der Aufruf stattfindet. In einer INCLUDE-Datei können ebenfalls Templatevariablen und -befehle stehen. Ich rate allerdings davon ab, in einer INCLUDE-Datei wieder zu inkludieren.

6. einfacher LOOP: User ausgeben

Kommen wir nun zu dem interessanten (und schwierigstem) Thema: den LOOPS. Anmerkung: Der Begriff "Loop" wird nicht in allen Template Engines verwendet. Ein LOOP ist in TinyButStrong ein "Block" und in Smarty werden dafür die Begriffe "section" und "loop" verwendet. Worum geht es bei einem LOOP? Dieses Konstrukt gleicht einer WHILE-Schleife, die eine bestimmte Anzahl von Daten oder Datensätzen ausgibt.

Um einen LOOP zu verwenden müssen die Daten in einem Array zur Verfügung stehen. Die Theorie, die hinter LOOPs steht, ist nicht ganz leicht zu verstehen. Die Variable $loop im untenstehenden Skript enthält einen sogenannten LOOP-Array. Bitte lesen Sie nach diesem Beispiel das letzte Kapitel Aufbau und Struktur eines LOOP-Arrays sehr sorgfältig durch. Das Verständnis von mehrdimensionalen Arrays ist für LOOP-Strukturen äußerst wichtig.

PHP-Skript

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/loop.htm');

$loop = array(
  0 => array('u_id' => 1, 'u_name' => 'Claus'),
  1 => array('u_id' => 2, 'u_name' => 'Kelvin'),
  2 => array('u_id' => 3, 'u_name' => 'Skrol'),
  3 => array('u_id' => 4, 'u_name' => 'Micheal')
);

$tmpl->setloop('loop', $loop);

$tmpl->pparse();
```

Dieses Beispiel benutzt einen klassischen LOOP-Array, der INTEGER und STRING Indizes (Mehrzahl von "Index") verwendet. In PHP spricht man von assoziativen Arrays und Arrays mit numerischem Index. php.net benutzt nicht nur den Begriff "Index", sondern auch "Schlüssel", aber andere Programmiersprachen verwenden immer das Wort "Index" oder "Indizierung", wenn es um den Arrays geht. Deswegen werde ich diesen Begriff ebenfalls verwenden.

Mit der Methode setloop() wird der LOOP-Array dem Template übergeben.

Template

```
<h1>vlibTemplate LOOP is displayed using <p></h1>

<tmpl_loop name='loop'>
   <p>User: {tmpl_var name='u_id'} - {tmpl_var name='u_name'}</p>
</tmpl_loop>
```

Die Kunst bei LOOP-Problematiken liegt nicht auf der Seite des Templates, sondern darin, dass der LOOP richtig aufgebaut sein muss. Für Debug-Zwecke ist der Befehl print_r() sehr zu empfehlen. Auch wenn man ein Problem mit LOOP-Arrays hat, sollte man diesen Befehl in sein PHP-Skript einbauen und die Ausgabe mit den Beispielen vergleichen.

7. verschachtelte LOOPs: User und Posts ausgeben

Im Gegensatz zum letzten Kapitel, werden wir nicht immer in der glücklichen Lage sein, perfekte LOOP-Strukturen vorzufinden. In den meisten Fällen müssen wir uns die LOOP-Arrays zusammenbauen. Das macht man mit dem Befehl array_push(). Denn wie wir im letzten Kapitel gelernt haben, müssen alle Daten, die wir in einem TMPL_LOOP verwenden wollen, in einem LOOP-Array gespeichert werden. Dieser muss mindestens zwei, bei verschachtelten LOOPS vier, sechs, acht oder mehr Dimensionen (durch 2 teilbar) haben.

Die nächste Frage: Reichen LOOP-Arrays mit zwei Dimensionen aus, um alle Problemstellungen zu lösen? Die Antwort: Natürlich nicht. Deswegen gibt es verschachtelte LOOPs.

Ein klassisches Problem ist der Gruppenwechsel oder eine 1:N Beziehung einer Datenbankstruktur. In einem Forum zum Beispiel kann ein User beliebig viele Posts erstellt haben - eine klassische 1:N Beziehung. Wir wollen jetzt jeden Post seinem User zuordnen oder anders ausgedrückt, jeder User soll mit seinen Posts aufgelistet werden. Der Einfachheit halber bilden wir diese Struktur in zwei Arrays ab:

PHP-Skript

```
require_once 'vlib/vlibTemplate.php';
```

```
$tmpl = new vlibTemplate('tmpl/nested_loops.htm');

$users = array(
    array('u_id' => 1, 'u_name' => 'Claus'),
    array('u_id' => 2, 'u_name' => 'Kelvin'),
    array('u_id' => 3, 'u_name' => 'Skrol'),
    array('u_id' => 4, 'u_name' => 'Micheal')
);

$posts = array(
    array('user_id' => 1, 'post' => '1st post by Claus'),
    array('user_id' => 1, 'post' => 'Claus has something to say'),
    array('user_id' => 1, 'post' => 'Claus: Addtional informations'),
    array('user_id' => 1, 'post' => 'Claus: Problem solved!'),
    array('user_id' => 2, 'post' => '1st post by Kelvin'),
    array('user_id' => 2, 'post' => 'Kelvin: Best regards!'),
    array('user_id' => 3, 'post' => '1st post by Skrol'),
    array('user_id' => 3, 'post' => 'Skrol: Use TBS!'),
    array('user_id' => 3, 'post' => 'Skrol made a different approach'),
    array('user_id' => 4, 'post' => '1st post by Micheal'),
    array('user_id' => 4, 'post' => 'Micheal: Thanks!')
);

$i = 0;
$outer = array();
foreach ($users as $key => $value)
{
    $inner = array();
    while ($value['u_id'] == $posts[$i]['user_id'])
    {
        array_push($inner, array(
            'p_post' => $posts[$i]['post']
            )
        );

        $i++;
    }

    array_push($outer, array(
            'u_id' => $value['u_id'],
            'u_name' => $value['u_name'],
            'inner' => $inner
            )
    );
}

$tmpl->setloop('outer', $outer);
$tmpl->pparse();
```

$users und $posts
: Die beiden Arrays sind ähnlich wie eine Datenbankabfrage aufgebaut. Eine Abfrage mit mysql_fetch_assoc() liefert ein assoziatives Array zurück, mit den Spaltennamen der Tabelle als Indizes. Die Problematik ist jetzt, die beiden Arrays in einen LOOP-Array zusammenzuführen. Dies geschieht mit verschachtelten LOOPs.

$i = 0;
: $i ist eine Zählvariable. Wie man im PHP Coding Standard nachlesen kann, sollten Zählvariablen beim Buchstaben "i" beginnen und dann alphabetisch fortlaufen.

$outer = array();
: Um den Befehl array_push() benutzen zu können, muss die Array $outer vorher initialisiert werden. Das gilt ebenfalls für $inner.

foreach ($users as $key => $value)
: Um den Array $users zu durchlaufen, wird hier mit foreach der Index (key) und der Wert (value) ausgelesen. Der Index wird nicht weiter verwendet im Skript.

while ($value['u_id'] == $posts[$i]['user_id'])
: Um die Zuordnung von jeden Post seinem User zu realisieren wird $i und eine WHILE-Schleife verwendet, die alle Posts durchläuft und sie im Array $inner speichert.

array_push($outer, array(...));
: Um einen verschachtelten LOOP-Array zu generieren, muss der innere LOOP ein Teil des äußeren LOOPs sein. Deswegen enthält $outer den Array $inner und wie man sehen kann, sind auch noch weitere (Template)Variablen möglich.

Template

```
<tmpl_loop name='outer'>
   <h1>User: {tmpl_var name='u_id'} - {tmpl_var name='u_name'}</h1>
    <tmpl_loop name='inner'>
      <div style="margin-left: 20px">#{tmpl_var name='__ROWNUM__'}:
              {tmpl_var name='p_post' escape='html'}</div>
    </tmpl_loop>
</tmpl_loop>
```

Das Template enthält die verschachtelte LOOP-Struktur. Wie man auf einen Blick sehen kann, gleichen die Postionierungen der beiden TMPL_LOOPs einer verschachtelten WHILE-Schleife. Genauso funktioniert dann auch die Ausgabe. Die eigentliche "Intelligenz" liegt im PHP-Skript und genau dahin gehört die Programmierlogik auch.

Ebenfalls neu ist "__ROWNUM__". Hierbei handelt es sich um eine Zählervariable, die in jedem LOOP automatisch gesetzt wird und bei 1 beginnt. Mehr Informationen können Sie unter Options - GLOBAL_CONTEXT_VARS nachlesen. Das Attribut "escape" wurde schon früher in diesem Tutorial erläutert. Trotzdem hier noch einmal der Hinweis, dass man weitere Informationen zu diesem und anderen Attributen im Forum finden kann.

8. einfache Datenbankausgabe

Ich verwende LOOPS fast ausschließlich für Datenbankausgaben. Legen Sie bitte eine beliebige Datenbank plus Tabelle mit folgenden Spalten an:

- name (Datentyp: CHAR)
- birthday (Datentyp: DATE)

- city (Datentyp: CHAR)
- country (Datentyp: CHAR)

Wir wollen in unserem ersten einfachen Beispiel zwei der Spalten in einer HTML-Tabelle ausgeben. Dieses geht relativ leicht mit der Methode setdbloop(), die Rückgabewerte eines RDBMS sofort in einem Template ausgeben kann.

PHP-Skript

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/db_simple.htm');

// DATABASE variables and query
require_once 'db_config.php';
mysql_connect($db_host, $db_user, $db_pw);
mysql_select_db($db_name);
$select = "SELECT name, city FROM $db_table";
$result = mysql_query($select);

$tmpl->setdbloop('table_data', $result);
$tmpl->pparse();

mysql_close();
```

Mit setdbloop() lassen sich die Daten nur ungefiltert aus der Datenbank ausgeben. Konvertierungen im Template sind mit vLIB nicht möglich, dazu ist der Befehl array_push() notwendig. Um das zu demonstrieren werden wir im nächsten Beispiel das Datum mit vlibDate formatiert ausgeben.

Wie man am Template sehen kann, ist hier keine Logik hinterlegt. Das ist der Ansatz, der von vLIB konsequent verfolgt wird: Alle (Programmier)Logik sollte im PHP-Skript erfolgen.

Template

```
<table border="1" width="70%" summary="MySQL data in a table using TMPL_LOOP and SETDBL
<caption>MySQL data in a table using TMPL_LOOP and SETDBLOOP</caption>
<thead>
   <tr>
      <th align="left">name</th>
      <th align="left">city</th>
   </tr>
</thead>
<tbody>
 <tmpl_loop name='table_data'>
   <tr>
      <td valign="top">{tmpl_var name='name'}</td>
      <td valign="top">{tmpl_var name='city'}</td>
   </tr>
 </tmpl_loop>
</tbody>
```

```
</table>
```

Die Datei "db_config.php" enthält alle notwendigen Variablen um die Verbindung zum RDBMS und der Datenbank herzustellen. Die Datei könnte beispielsweise folgenden Inhalt haben:

PHP-Skript

```
<?php
        $db_host = 'localhost';
        $db_name = 'vlib';
        $db_user = 'admin';
        $db_pw = 'admin123';
        $db_table = 'vlibtemplate';
?>
```

$select und $result
: ... sind Datenbankvariablen. Sie werden gebraucht, um die Datensätze aus der Datenbank zu extrahieren. Die genaue Funktionsweise kann man auf php.net oder #php/QuakeNet nachlesen.

Die Templatevariablen "name" und "city"
: Wir wollen die Spalten "name" und "city" ausgeben. Deswegen definieren wir im Template

  {tmpl_var name='name'}

  und

  {tmpl_var name='city'}.

  Die Templatevariablen müssen mit den Spaltennamen übereinstimmen. Die Methode "setdbloop" funktioniert nur dann, wenn wir genau wissen, welche Spalten wir von der Tabelle zurück erhalten. Ein "SELECT * FROM tabelle" würde also nur dann funktionieren, wenn wir die Struktur der Tabelle im Template abbilden. Bitte beachten Sie, dass der SELECT die Spalten "name" und "city" selektiert und diese IDENTISCH im Template hinterlegt sind. Auch Groß- und Kleinschreibung muss in diesem Fall beachtet werden.

Wenn man im Template keine Logik hinterlegen kann, ist es dann überhaupt möglich mit setdbloop() Daten aus einer Tabelle zu manipulieren? Ja und Nein. Nein, weil die Methode das nicht hergibt. Ja, weil String-, Datums- und sonstige Funktionen mit SQL sehr wohl zu realisieren sind. Selbst IF-Strukturen, die zwei Datenbankfelder miteinander vergleichen, sind mit MySQL möglich.

9. Einen DB-LOOP erstellen mit ARRAY_PUSH

Wenden wir uns nun dem etwas komplizierteren Beispiel zu. Oft wird es nötig sein, Daten zu manipulieren, egal ob diese aus der Datenbank kommen oder aus einer anderen Quelle stammen. Wenn diese Daten in einen LOOP sollen, müssen wir auf den Befehl array_push() zurückgreifen.

PHP-Skript

```
require_once 'vlib/vlibTemplate.php';
require_once 'vlib/vlibDate.php';

$tmpl = new vlibTemplate('tmpl/db.htm');
$date = new vlibDate('de'); // you may set this to 'en' (english)

// DATABASE variables and query
require_once 'db_config.php';
mysql_connect($db_host, $db_user, $db_pw);
mysql_select_db($db_name);
$select = "SELECT name, birthday, city FROM $db_table";
$result = mysql_query($select);

$table_data = array();
while ($row = mysql_fetch_assoc($result))
{
   array_push($table_data, array(
      'name' => $row['name'],
      'birthday' => $date->formatDate($row['birthday'], '%A, %d.%m.%Y'),
      'city' => $row['city']
      )
   );
}
$tmpl->setloop('table_data', $table_data);

$tmpl->pparse();

mysql_close();
```

$date = new vlibDate('de');

Dieser Befehl erstellt die Instanz $date der Klasse vlibDate. Das 'de' stellt den deutschen Sprachsatz ein. So erhalten wir "Samstag" oder "Mittwoch" (je nach Wochentag) und nicht die englischen Bezeichnungen. vlibDate ist (wie strftime) in der Lage, verschiedene Formate und Variablen zu unterstützen. Lesen Sie bitte die Dokumentation zu vlibDate, um sich über die anderen Sprachen und Möglichkeiten dieser Klasse zu informieren.

array_push($table_data, ...)

Mit array_push() werden die einzelnen LOOP-Elemente dem Array $table_data hinzugefügt. Während in unserem ersten LOOP die LOOP-Elemente nur aus einfachen Variablen bestanden haben, findet bei dieser Zuweisung:

'birthday' => $date->formatDate($row[birthday], '%A, %d.%m.%Y');

etwas Besonderes statt - dieses LOOP-Element wird vom ISO 8601 Format (2005-11-09) in ein gebräuchliches deutsches Format umgewandelt.

Template

```
<table border="1" width="70%" summary="MySQL data in a table using TMPL_LOOP and ARRAY_
   <caption>MySQL data in a table using TMPL_LOOP and ARRAY_PUSH</caption>
   <thead>
      <tr>
         <th align="left">name</th>
         <th align="left">birthday</th>
         <th align="left">city</th>
      </tr>
   </thead>
   <tbody>
    <tmpl_loop name='table_data'>
      <tr>
         <td valign="top">{tmpl_var name='name'}</td>
         <td valign="top">{tmpl_var name='birthday'}</td>
         <td valign="top">{tmpl_var name='city'}</td>
      </tr>
    </tmpl_loop>
   </tbody>
</table>
```

Das Template unterscheidet sich nicht wesentlich von den vorangegangen Beispielen. Der LOOP enthält 3 Elemente, die alle über Templatevariablen ausgegeben werden.

Die Ausgabe

name	birthday	city
Nordheimer, Ute	Samstag, 21.05.1960	Berlin
Smith, Daniel	Dienstag, 22.06.1965	Salt Lake City
Pullu, Murath	Donnerstag, 23.07.1970	Kairo
Leone, Gorgio	Sonntag, 24.08.1975	Rom

Die Möglichkeiten von vlibDate werden hier nur leicht angekratzt. Die deutsche Formatierung ist nur einer der vielen Vorteilen dieser Klasse.

10. variable SELECT's / Übungsbeispiel

Wir haben gelernt mit LOOPS umzugehen und mehrere Datensätze mit TMPL_LOOP auszugeben. Das waren jedoch statische SELECT-Anweisungen, die eine vordefinierte Anzahl von Spalten hatten. Ebenso waren unsere Templates klar definiert.

Was passiert, wenn wir einen dynamischen LOOP bauen wollen, der jeden SELECT ausgeben kann? Wenn die Anzahl der Spalten unbekannt ist? Dann kommen wir mit den erläuterten Beispielen nicht weiter.

Eine Datenbank-Tabelle besteht aus Zeilen und Spalten (siehe die Zeichnung weiter unten). Das heißt, eine Datenbank-Tabelle kann normalerweise mit einem *zweidimensionalen Array* dargestellt

werden. Da wir aber für einen LOOP immer auch die Namen der Templatevariablen im Array benötigen, brauchen wir für die Zeilen und Spalten einer Tabelle einen *vierdimensionalen Array* (2 x 2 = 4). Hinweis: Ein LOOP-Array hat immer die doppelte Anzahl von Dimensionen als der ursprüngliche Array (siehe Aufbau und Struktur eines LOOP-Arrays).

Fangen wir mit der Erstellung des Templates an:

```
<html>
<head>
    <title>display data from an unknown table</title>
</head>

<body>

<table border="1" width="100%">
```

Damit hätten wir den Anfang. Jetzt müssen wir die Spaltenüberschriften in das Template einbinden. Dazu reicht uns ein einfacher LOOP mit einem zweidimensionalen LOOP-Array. Die Anzahl der Spaltenüberschriften ist zwar ebenfalls unbekannt, aber diese sind in einem *eindimensionalen Array* darstellbar und der LOOP-Array hat demnach **zwei** Dimensionen.

Erklärung zu den Spalten

$spalten = array('name', 'city', 'country');

oder

$spalten = array('country', 'city');

Wie schon erwähnt, ist $spalten ein Array mit einer Dimension. Aufbereitet muss er zwei Dimensionen und somit folgende Struktur haben:

```
$loop_spalten = Array
(
        [0] => Array (  [colname] => name  )
        [1] => Array (  [colname] => city  )
        [2] => Array (  [colname] => country  )
)
```

$loop_spalten ist unser LOOP-Array, indem wir "colname" als Templatevariable festgelegt haben.

```
  <tr>
    <tmpl_loop name='table_header'>
      <th align="left">{tmpl_var name='colname'}</th>
    </tmpl_loop>
  </tr>
```

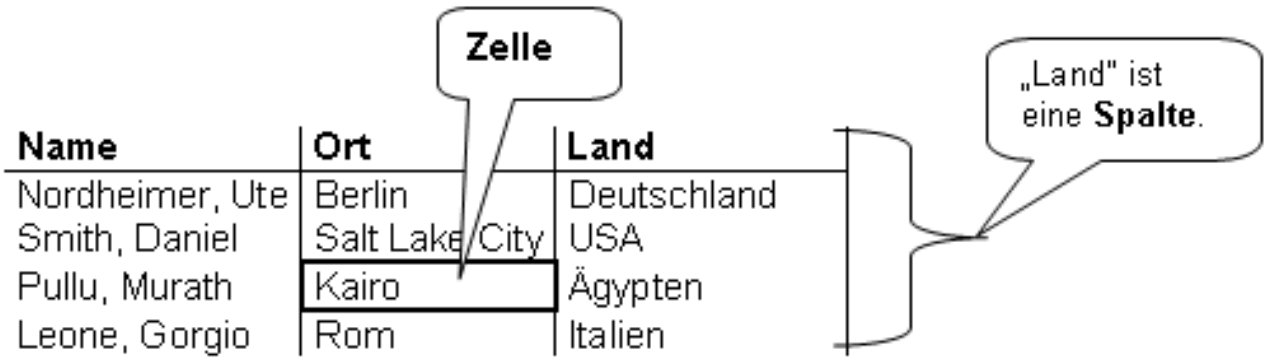
Die Spaltenüberschriften haben wir jetzt im Template untergebracht, jetzt kommen die Datensätze. Dazu werden wir einen verschachtelten LOOP brauchen.

Wie bauen wir diesen LOOP am besten auf? Der äußere LOOP enthält alle Tabellendaten, sowohl die Zeilen, als auch die Spalten. Also nennen wir LOOP und LOOP-Array "table_data".

- Name des äußeren LOOPs: **table_data**
- Name des äußeren LOOP-Arrays: **$table_data**

Der innere LOOP wird eine komplette Zeile enthalten, also wäre "row" ein logischer Name für den LOOP und "cell" der Name für die Templatevariable bzw. den Zelleninhalt. Den LOOP-Array kann man entweder "$row" oder "$cells" (es sind ja mehrere Zellen enthalten) nennen, allerdings wird "$row" schon durch $row = mysql_fetch_row($result) vorbelegt, also können wir diesen Variablennamen nicht mehr vergeben.

- Name des inneren LOOPs: **row**
- Name des inneren LOOP-Arrays: **$cells**



„Smith, Daniel; Salt Lake City; USA" ist eine **Zeile** bzw. ein **Datensatz**.

Es steht Euch natürlich frei, andere Namen zu verwenden. Wie dem auch sei, der (vierdimensionale) LOOP-Array muss folgendermaßen aufgebaut werden:

```
Array
(
   [0] => Array
   (
      [row] => Array
      (
         [0] => Array ( [cell] => Nordheimer, Ute )
         [1] => Array ( [cell] => Berlin )
         [2] => Array ( [cell] => Deutschland )
      )
   )
   [1] => Array
   (
      [row] => Array
```

```
      (
         [0] => Array ( [cell] => Pullu, Murath )
         [1] => Array ( [cell] => Kairo )
         [2] => Array ( [cell] => Ägypten )
      )
   )
)
```

Und das Template so:

```
  <tmpl_loop name='table_data'>
      <tr bgcolor="<tmpl_if name='__EVEN__'>#eeeeee<tmpl_else>#dcdcde</tmpl_if>">
        <tmpl_loop name='row'>
           <td valign="top">{tmpl_var name='cell'}</td>
        </tmpl_loop>
      </tr>
  </tmpl_loop>
</table>

</body>
</html>
```

Wir haben an dieser Stelle noch eine kleine Neuheit:

"__EVEN__" ist eine Variable, die nur in einem LOOP gesetzt ist. Sie ist entweder TRUE oder FALSE (1 oder 0). Siehe dazu auch den Abschnitt "Options - GLOBAL_CONTEXT_VARS" im Original der vlibTemplate-Dokumentation.

Da jetzt das Template fertig ist, können wir uns dem PHP-Skript "db_adv.php" zuwenden.

```
<?php
   require_once 'vlib/vlibTemplate.php';

   $tmpl = new vlibTemplate('tmpl/db_adv.htm');

   // DATABASE variables and query
   require_once 'db_config.php';
   mysql_connect($db_host, $db_user, $db_pw);
   mysql_select_db($db_name);
   $select = "SELECT * FROM $db_table";
   $result = mysql_query($select);
```

Ok, der Einstieg wäre geschafft. Jetzt müssen wir uns überlegen, wie wir den Header aufbauen. Das ist noch relativ einfach; wir brauchen einen

- LOOP-Array ($table_header)
- eine FOR-Schleife, die solange läuft, wie wir Spalten haben
- und die Zuweisung jeder Spaltenüberschrift in $table_header

```
   $table_header = array();
   for ($i = 0; $i < mysql_num_fields($result); $i++)
   {
```

```
   array_push($table_header, array('colname' => mysql_field_name($result, $i)));
}
$tmpl->setloop('table_header', $table_header);
```

Anhand dieses Beispiels können wir auch den nächsten LOOP aufbauen. Für den äußeren LOOP verwenden wir den Array $table_data und der innere wird in $cells abgespeichert. Nun besteht eine Tabelle ($table_data) aus mehreren Zellen, also muss $cells ein Teil von $table_data sein.

```
$table_data = array();
while ($row = mysql_fetch_row($result))
{
   $cells = array();
   for ($j = 0; $j < mysql_num_fields($result); $j++)
   {
      array_push($cells, array('cell' => $row[$j]));
   }

   array_push($table_data, array('row' => $cells));
}
$tmpl->setloop('table_data', $table_data);

$tmpl->pparse();

mysql_close();
```

Unser Beispiel besteht jetzt im Wesentlichen aus den 2 LOOPs $table_header und $table_data. Beide werden abhängig vom $result gebildet und somit ist jeder SELECT ausführbar und das Ergebnis kann abgebildet werden.

11. 3er Methode: newLoop, addRow, addLoop

Die "3er Methode" (englisch: *3 stage method*) ist dem "array_push-Verfahren" sehr ähnlich. Man kann sich die komplette Anwendung in zwei Beispielen sehr genau ansehen. Das erste Beispiel enthält eine Navigation, wie sie oft bei Forensoftware verwendet wird. Das zweite Beispiel zeigt einen Gruppenwechsel, der über einen verschachtelten LOOP User und Posts aus einer Datenbank ausibt.

PHP-Skript

```
$tmpl->newLoop('outer');
while ($row_users = mysql_fetch_assoc($result_users))
{

(...)

   $tmpl->addRow(array(
         'u_id' => $row_users['id'],
         'u_name' => $row_users['name'],
         'inner' => $inner
```

```
        )
    );
}
$tmpl->addLoop();
```

Wie man sehen kann, ist die Struktur mit dem "array_push-Verfahren" nahezu identisch. Der große Vorteil der "3er Methode" liegt bei der Ausgabe von eindimensionalen Arrays.

PHP-Skript

```
$array_names1 = array('Claus', 'Kelvin');
$array_names2 = array('Skrol', 'Daniela');
$array_names3 = array('Marion', 'John');

$tmpl->newLoop('loop');
$tmpl->addRow($array_names1);
$tmpl->addRow($array_names2);
$tmpl->addRow($array_names3);
$tmpl->addLoop();
$tmpl->pparse();
```

Die einzelnen Elemente können dann über die Templatevariablen

- {tmpl_var name='_0'}
- {tmpl_var name='_1'}

dargestellt werden. Das PHP-Skript ist sauber und übersichtlich.

12. Modulares Programmieren mit TMPL_INCLUDE

Bei komplexen Designs bietet sich ein modularer Aufbau der Templates und der PHP-Skripte an. Dieses kann auf zwei Arten realisiert werden:

1. Parameterübergabe für TMPL_INCLUDE
2. einzelne PHP-Skripte, die jeweils ein Template haben

Welche der beiden Methoden besser ist, liegt im Auge des Betrachters. Ich persönlich bevorzuge Variante 2, weil sie die Modularisierung (Programmierlogik) in die Hände von PHP legt und nicht in die des Templates. Vielleicht ist das auch möglich mit Variante 1. Dennoch wird an dieser Stelle zuerst Variante 1 erklärt.

PHP-Skript

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/modular_tmpl_include.htm');

$tmpl->setvar('header', 'modular_tmpl_include_header.htm');
$tmpl->setvar('body', 'modular_tmpl_include_body.htm');
$tmpl->setvar('title_text', 'TITLE: Modular programming with TMPL_INCLUDE');
$tmpl->setvar('body_text', 'BODY: Modular programming with TMPL_INCLUDE');

$tmpl->pparse();
```

Die Befehle sind alle bekannt. Mit den Tags <head></head> und <body></body> unterteilt HTML die Bereiche. Offensichtlich unterteilt unser PHP-Skript das Template ebenfalls in diese Bereiche. Was allerdings genau passiert wird erst durch das Template klar.

Template: modular_tmpl_include_header.htm

```
<tmpl_include file='{var:header}'>
<tmpl_include file='{var:body}'>
```

Wir haben im PHP-Skript die Variablen "header" und "body" gesetzt. Diese werden vom Template dann inkludiert. Somit sind auch komplexe Naviagation über $_GET oder aus einer Datenbank möglich.

Template: modular_tmpl_include_header.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
<head>
   <title>{tmpl_var name='title_text'}</title>
   <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>
```

Das Template ist mit dem ersten Beispiel in diesem Tutorial identisch. Nur ist hier der HEAD-Tag vom BODY-Tag getrennt bzw. in zwei verschiedenen Dateien untergebracht. Kompliziertere Strukturen sind natürlich auch denkbar.

Template: modular_tmpl_include_body.htm

```
<body>

<p>{tmpl_var name='body_text'}</p>

</body>
</html>
```

13. Modulares Programmieren mit REQUIRE_ONCE

Wie oben schon erwähnt bevorzuge ich die untenstehende Variante der modularen Programmierung. Die Inkludierung erfolgt über den Befehl require_once() und das PHP-Skript ist sehr einfach.

PHP-Skript

```
<?php
   require_once 'modular_php_require_header.php';
   require_once 'modular_php_require_body.php';
?>
```

Diese Methode ermöglicht mehrere PHP-Skripte zu einem Ganzen zusammenzufügen. Diese sind ansonsten völlig unabhängig voneinander. Darin liegt für mich auch der Vorteil der Modularisierung: Die Skripte können modifiziert und erweitert werden, ohne das die anderen Skripte verändert werden müssen. Und genau darin liegt der Vorteil der Modularisierung.

PHP-Skript: modular_php_require_header.php

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/modular_php_require_header.htm');
$tmpl->setvar('title_text', 'TITLE: Modular programming with REQUIRE_ONCE');
$tmpl->pparse();
```

Die Templates sind völlig identisch mit den Beispielen aus Modulares Programmieren mit TMPL_INCLUDE - auch hier sind HEAD- und BODY-Tag getrennt.

PHP-Skript: modular_php_require_body.php

```
require_once 'vlib/vlibTemplate.php';

$tmpl = new vlibTemplate('tmpl/modular_php_require_body.htm');
```

```
$tmpl->setvar('body_text', 'BODY: Modular programming with REQUIRE_ONCE');
$tmpl->pparse();
```

Der Aufwand dieser Modularisierung ist höher, weil man ein Skript mehr hat und Befehle wie $tmpl = new vlibTemplate() doppelt vorkommen. Dafür ist die Zuordnung und die Modularisierung in meinen Augen klarer und deutlicher.

14. Aufbau und Struktur eines LOOP-Arrays

Damit vlibTemplate einen LOOP ausgeben kann, muss ein LOOP-Array erzeugt werden, der eine bestimmte Struktur haben muss:

LOOP-Array in PHP

```
$loop = array(
             0 => array('u_id' => 1, 'u_name' => 'Claus'),
             1 => array('u_id' => 2, 'u_name' => 'Kelvin'),
             2 => array('u_id' => 3, 'u_name' => 'Skrol'),
             3 => array('u_id' => 4, 'u_name' => 'Micheal')
     );

echo '<pre>'; print_r($loop); echo '</pre>';
```

Der oben dargestellte Array hat genau die Struktur eines LOOP-Arrays. Mit print_r() lassen sich Arrays sehr übersichtlich ausgeben:

LOOP-Array: $loop

```
Array
(
    [0] => Array
        (
            [u_id] => 1
            [u_name] => Claus
        )
    [1] => Array
        (
            [u_id] => 2
            [u_name] => Kelvin
        )
    [2] => Array
        (
            [u_id] => 3
            [u_name] => Skrol
```

```
        )
    [3] => Array
        (
            [u_id] => 4
            [u_name] => Micheal
        )
)
```

Dieses Array wird im Beispiel "loop.php" verwendet. Wie man deutlich sehen kann, handelt es sich um einen *zweidimensionalen Array*. Bei LOOP-Arrays ist eine Mischung aus Integer- und String-Indizes zwingend vorgeschrieben.

Jeder LOOP muss so aufgebaut werden, dass es sich um einen *zweidimensionalen Array* handelt, der über zwei Schlüsselfelder angesprochen wird:

- Zahl (Integer)
- Zeichenkette (String)

Im Beispiel für den "basic_loop" könnte man ein Feld des LOOPS so ausgeben:

echo "Ein Name: {$loop[2][u_name]}";

Dieselbe Struktur gilt auch für verschachtelte LOOPS. Allerdings hat ein verschachtelter LOOP nicht 2 sondern 4, 6, 8 ... also eine *gerade* Anzahl von Dimensionen. Beispiel:

LOOP-Array: $outer

```
Array
(
    [0] => Array
        (
            [u_id] => 122
            [u_name] => jermyn
            [inner] => Array
                (
                    [0] => Array
                        (
                            [p_id] => 1235
                            [p_post] => Hallo Claus und die anderen, die mitlesen.
                        )
                    [1] => Array
                        (
                            [p_id] => 1237
                            [p_post] => Hallo Coda, ich hab mir da mal eben dei
                        )
                    [2] => Array
                        (
                            [p_id] => 1238
                            [p_post] => Hallo Claus, ich hab mir die vLIB Templ
                        )
                    [3] => Array
                        (
```

```
                            [p_id] => 1243
                            [p_post] => Hallo Claus, ich sehe das ja genau so w
                        )
                )
        )
    [1] => Array
            [u_id] => 119
            [u_name] => rehmaster
            [inner] => Array
                (
                    [0] => Array
                        (
                            [p_id] => 1203
                            [p_post] => Ich habe folgendes Problem! Aus 2 MySQL Tabell
                        )
                    [1] => Array
                        (
                            [p_id] => 1205
                            [p_post] => Der kann die DB wohl nciht öffnen! Wäre n
                        )
                )
        )
    [2] => Array
...
)
```

Auch hier wird die strikte Mischung aus Integer- und String-Indizes eingehalten. Die Ausgabe könnte zum Beispiel mit echo "Ein Post von rehmaster: {$outer[1][inner][0][p_post]}"; erfolgen. Hinweis: Die Debug-Funktion von vlibTemplate zeigt immer einen Array mehr an, als es in Wirklichkeit ist:

vlibTemplateDebug: $loop

```
Array
(
   [loop] => Array
      (
          [0] => Array
              (
                  [u_id] => 1
                  [u_name] => Claus
                  [__FIRST__] => 1
                  [__ODD__] => 1
                  [__ROWNUM__] => 1
              )
          [1] => Array
              (
                  [u_id] => 2
                  [u_name] => Kelvin
                  [__INNER__] => 1
                  [__EVEN__] => 1
                  [__ROWNUM__] => 2
              )
...
)
```

Hier steht 3x Array, obwohl es in Wirklichkeit nur 2 sind. Es kann sein, dass vLIB intern aus 2 dann 3 Dimensionen macht. Man braucht es für die Programmierung jedoch nicht zu beachten.

Wie man sehen kann, werden in jedem LOOP-Array Konstanten mitgeführt wie: __ODD__ und __ROWNUM__.

Zum Schluss die Arrays aller LOOP-Beispiele auf einen Blick:

db_simple.php

```
Array
(
    [0] => Array
        (
            [name] => Nordheimer, Ute
            [city] => Berlin
        )
    [1] => Array
        (
            [name] => Smith, Daniel
            [city] => Salt Lake City
        )
    [2] => Array
        (
            [name] => Pullu, Murath
            [city] => Kairo
        )
    [3] => Array
        (
            [name] => Leone, Gorgio
            [city] => Rom
        )
)
```

db.php

```
Array
(
    [0] => Array
        (
            [name] => Nordheimer, Ute
            [birthday] => Samstag, 21.05.1960
            [city] => Berlin
        )
    [1] => Array
        (
            [name] => Smith, Daniel
            [birthday] => Dienstag, 22.06.1965
            [city] => Salt Lake City
        )
```

```
        [2] => Array
            (
                [name] => Pullu, Murath
                [birthday] => Donnerstag, 23.07.1970
                [city] => Kairo
            )
        [3] => Array
            (
                [name] => Leone, Gorgio
                [birthday] => Sonntag, 24.08.1975
                [city] => Rom
            )
    )
```

db_adv.php

```
Array
(
    [0] => Array
        (
            [colname] => name
        )
    [1] => Array
        (
            [colname] => birthday
        )
    [2] => Array
        (
            [colname] => city
        )
    [3] => Array
        (
            [colname] => country
        )
)

Array
(
  [0] => Array
      (
          [row] => Array
              (
                  [0] => Array
                      (
                          [cell] => Nordheimer, Ute
                      )
                  [1] => Array
                      (
                          [cell] => 1960-05-21
                      )
                  [2] => Array
                      (
                          [cell] => Berlin
                      )
                  [3] => Array
                      (
```

```
                        [cell] => Deutschland
                    )
            )
    )
[1] => Array
    (
        [row] => Array
            (
                [0] => Array
                    (
                        [cell] => Smith, Daniel
                    )
                [1] => Array
                    (
                        [cell] => 1965-06-22
                    )
                [2] => Array
                    (
                        [cell] => Salt Lake City
                    )
                [3] => Array
                    (
                        [cell] => USA
                    )
            )
    )
[2] => Array
    (
        [row] => Array
            (
                [0] => Array
                    (
                        [cell] => Pullu, Murath
                    )
                [1] => Array
                    (
                        [cell] => 1970-07-23
                    )
                [2] => Array
                    (
                        [cell] => Kairo
                    )
                [3] => Array
                    (
                        [cell] => Ägypten
                    )
            )
    )
[3] => Array
    (
        [row] => Array
            (
                [0] => Array
                    (
                        [cell] => Leone, Gorgio
                    )
                [1] => Array
                    (
                            [cell] => 1975-08-24
                    )
```

```
                    [2] => Array
                        (
                            [cell] => Rom
                        )
                    [3] => Array
                        (
                            [cell] => Italien
                        )
                )
        )
    )
```

Die Ausgaben wurden alle mit dem Befehl print_r() vorgenommen.