Metrix Schulung:

Trennung von Content und Layout

Teil 1: HTML & Marketing

Inhaltsverzeichnis

1Vorbemerkung

Die folgenden Ausführungen beziehen sich, falls nicht anders angegeben, auf die Erstellung von Webseiten auf HTML Basis.

2Was bedeutet Trennung von Content und Layout?

Offensichtlich werden Webseiten bei der Trennung von Content und Layout in zwei Teile zerlegt, den Content (Inhalt) und das Layout, oft mit Design bezeichnet. Auf den ersten Blick erscheint dies recht einfach, bei genauerer Betrachtung, können aber Schwierigkeiten auftreten, mit denen man vorher nicht gerechnet hat.

So würde man, wenn man sich nicht weiter mit dem Thema beschäftigt, jede Grafik einer Webseite in den Bereich des Layout stecken, schließlich wurde sie von einem Designer erstellt, sie ist ist folglich ein Element des Designs. Wie sieht das nun aus, wenn man in einer Firmenwebseite eine Auflistung der Sachbearbeiter anbietet und zu jedem Sachbearbeiter ein Photo anbietet. Dieses Photo ist sicher kein Designelement, obwohl es graphisch ist, es gehört folglich zum Inhalt der Seite, zum Content.
Starke Schwierigkeiten treten auf, wenn das Photo z.B. mit einem Rahmen versehen wird, um es dem Gesamtbild der Seite anzupassen. Das Photo ist nun auch nicht mehr 100% Teil des Inhalts, schließlich fügt es sich in ein Design ein. Man müsste das Photo nun trennen, in das eigentliche Photo, das immer noch reiner Inhalt ist - schließlich illustriert es einen bestimmten Text – und in diesem Fall den Rahmen des Photos, der zum Design gehört.

Daraus lässt sich also erkennen, dass es nicht so einfach ist, den Inhalt vom Design zu trennen. Helfen kann dabei eine Definition der beiden Begriffe.

*Content:*Der Content einer Website ist die Summe der Informationen und Daten, die dem Besucher vermittelt werden. Er besteht aus Text, Grafiken Symbolen und eventuell sogar aus Animationen.

Layout:Das Layout einer Site visualisiert den Content und verwendet dazu HTML, Grafiken, verschiedene Schriften und Farben, kann aber auch z.B. wenn das Layout in Flash erstellt wird Musik und Animationen beinhalten.

Ein Hilfsmittel bei der Trennung kann der Versuch sein, ein Element einer Website einfach wegzulassen. Bietet die Site immer noch die gleiche Information, so war das Element ein Teil des Layouts, ansonsten gehörte es zum Content.

3Gründe für die strikte Trennung von Content und Layout

„Warum soll man eigentlich den Inhalt vom Layout trennen, schließlich macht es einem Arbeit und bisher hielt es auch niemand für absolut notwendig?“ werden sicher jetzt einige fragen. Für die Trennung im Bereich der Produktion von Webseiten gibt es hauptsächlich zwei Gründe:

- Der Inhalt soll leicht veränderbar sein.
  Dies ist häufig der Fall, wenn es sich um dynamischen Inhalt handelt, z.B. PHP Code. Der Entwickler muss sich oft durch Seiten von HTML Code kämpfen, obwohl es für seine Arbeit vollkommen uninteressant ist. Wichtig wäre für ihn, dass er den eigentlichen PHP Code ändern kann, ohne dabei am HTML etwas ändern zu

müssen. Oder ein Redakteur soll Inhalte ändern, kennt aber die Bedeutung von HTML Tags nicht und kann so nur auf den Inhalt zugreifen, ohne an das Layout zu müssen.

- Das Layout soll leicht veränderbar sein.
  Dies ist entweder der Fall, wenn ein Kunde ein Redesign wünscht, ohne den eigentlichen Inhalt seiner Seiten austauschen zu wollen. Auch bei Web-Applikationen wie z.B. einem Forum muss das Layout schnell austauschbar sein, um es mehreren Kunden verkaufen und es leicht in das bestehendes Design des Kunden integrieren zu können. Dieser Vorgang kann soweit optimiert werden, dass sich das Layout zur Laufzeit ändern lässt.

4Bisherige Ansätze für die Trennung von Content und Layout

Bisher gab es schon einige Ansätze, Content von Layout zu trennen. Die wichtigsten möchte ich hier kurz vorstellen:

4.1HTML

Der wohl bekannteste Ansatz im Bereich der Webseitenerstellung, Layout und Content zu trennen, sind HTML Tags. Sie werden zwar heute nicht mehr dafür benutzt, jedoch waren sie ursprünglich dazu gedacht. Aus diesen Gründen gibt es Tags wie <p>, oder , die z.B einen Absatz kennzeichnen, oder auch einen wichtigen Teil im Text, im Gegensatz zu den Tags
, <i> oder <b>, die einen Zeilenumbruch erzwingen oder den Text kursiv oder fettgedruckt darstellen. Wenn man den Tag verwendet, so stellen die gängigsten Browser den Text zwar kursiv dar, er besagt aber eigentlich nur, das auf diesen Teil des Textes eine besondere Betonung gelegt werden soll. Es ist also nicht festgelegt, wie der Text in anderen Programmen, die in der Zukunft dargestellt werden wird, wohingegen der Tag <i> immer zur kursiven Darstellung das Textes führt.

Inzwischen stellen jedoch sowohl Designer als auch Kunde höhere Ansprüche an das Layout einer Webseite und eine Möglichkeit, einen Text irgendwie hervorzuheben reichte nicht mehr und HTML wird verwendet, um pixelgenaue Layouts umzusetzen. Es ist also nicht mehr ohne weiteres möglich, das Layout oder den Inhalt auszutauschen.

4.2Redaktionssysteme

Der Bedarf, Mitarbeiter, die nicht aus dem Bereich der HTML Programmierung stammen, mit der Pflege des Inhalts zu betreuen ist jedoch weiterhin da und muss gelöst werden. Dieser Bedarf führte zur Entwicklung von Redaktionssystemen, wie z.B. InfoOffice oder auch hausgemachten Lösungen, die auf den aktuellen Bedarf zugeschnitten sind. Damit sind natürlich immer Kosten verbunden, so dass sich der Aufwand erst ab einer bestimmten Projektgröße lohnt.

4.3XML

Auch aus diesen Gründen wurde vom W3C das XML (E**x**tensible **M**arkup **L**anguage) Konzept entwickelt. XML ist eine Metasprache, die es ermöglicht, neue Markup Languages zu erstellen, ähnlich HTML. Der Vorteil lag darin, dass es möglich ist, eigene Tags zu erfinden und in einer XSL Datei zu definieren, wie der Tag in HTML umgesetzt wird. Das ganze Konzept und alle Anwendungsgebiete von XML hier zu erläutern würde sicher den Rahmen sprengen, bei Bedarf könnte ich jedoch eine XML Schulung

anbieten oder auch das Buch XML kompakt[1] empfehlen. Ich will deshalb in dieser Schulung das Konzept von XML nur kurz erläutern. XML ist dazu entwickelt worden, Datenstrukturen abzulegen und zwischen Applikationen auszutauschen. In Datenbanken liegen alle Daten flach, d.h. es gab ähnlich einer Excel Spalten und Zeilen. In einem Feld kann jedoch nur eine Information stehen, also z.B. der Name des Kunden steht in Spalte 1, seine eMail Adresse in Spalte 2 und für jeden Kunden wird eine Zeile angelegt. Hat ein Kunde zwei eMail Adressen so lässt sich die zweite Adresse nirgendwo eintragen. Man müsste dafür eine zweites Feld oder eine zweite Tabelle anlegen. An dieser Stelle kommt XML zum Einsatz. Die Daten würden in XML beispielsweise folgendermaßen aussehen:

```
<kunde>
   <name>Karl Mustermann</name>
   <email>karl@mustermann.com</email>
   <email>karl@mustermann.de</email>
</kunde>
```

Im Datensatz für einen Kunden lassen sich jetzt also beliebig viele eMail Adressen speichern. Nach obigem Beispiel ließen sich auch mehrere Namen speichern, was für einen Kunden sicherlich nicht erwünscht ist. Aus diesem Grund gibt es eine weitere Datei (DTD, Document Type Definition), in der steht, wie oft welches Element an welcher Stelle im XML Dokument auftreten darf und mit der jedes XML Dokument auf diese DTD validiert werden kann. Die großen Vorteile von XML werden offensichtlich, wenn man größere Datenstrukturen oder Dokumentstrukturen abbilden möchte, wie z.B. Artikel in einem Magazin, Seiten auf einer Webseite oder auch ganze Bücher (was inzwischen schon gemacht wird).

```
<artikel>
   <ueberschrift>Ein Artikel in XML</ueberschrift>
   <autor>Stephan Schmidt</autor>
   <paragraph>
      In diesem Artikel geht es um Datenstrukturierung mit Hilfe von XML.
      <bild>
         <quelle>img/diagramm.gif</quelle>
         <bildunterschrift>Visuelle Darstellung einer XML Datei</bildunterschrift>
      </bild>
   </paragraph>
   <paragraph>
      Wie man sieht, kann ein Artikel mehrere Paragraphen enthalten, die wiederum
      Bilder enthalten können.
   </paragraph>
</artikel>
```

Dieses XML Dokument strukturiert lediglich die Daten eines Zeitungsartikels, eine graphische Darstellung wird nicht festgelegt. Diese kann später vom Ausgabemedium abhängen, das z.B. ein Browser, ein Handy oder auch Papier sein kann.
Der größte Nachteil von XML ist, dass es bisher nur vom Internet Explorer 5.5 unterstützt wird, eine Anwendung in der Praxis also noch nicht sinnvoll ist.

1Michel, Thomas: XML kompakt: eine praktische Einführung. München; Wien: Hanser, 1999

4.4Templates

Um auch bei kleineren Projekten Content von Layout zu trennen wurde das Prinzip der Templates entwickelt. Templates sind im Prinzip normale HTML Dateien, die an den Stellen, an denen der Content stehen würde Platzhalter enthalten. Diese Templates werden von einem Programm eingelesen, die Platzhalter durch den tatsächlichen Inhalt ersetzt und das entstandene Standard HTML an den Browser geschickt. Die Programme, die die Templates bearbeiten (parsen) wurden nach und nach um einige Funktionen, wie z.B. automatische Wiederholung von Teilen erweitert und im Internet befinden sich verschiedene Applikationen, die diese Arbeiten in PHP erledigen, die sich im Ansatz alle ähneln. Zu nennen wäre hier auf jeden Fall die Klasse EasyTemplate von Kristian Köhntopp, die unter der Lizenz GPL (Gnu Public License, d.h. die Software darf frei verwendet werden) und eben die Klasse patTemplate, die ich selbst programmiert habe und die eigentlicher Grund der Schulung ist und um deren Funktionen es im weiteren Verlauf der Schulung gehen wird.

5Anwendungsgebiete und Features der Klasse patTemplate

Die Klasse patTemplate wurde aus der oben beschriebenen Problemsituation heraus entwickelt, da ich eine Möglichkeit suchte, den Inhalt meiner in PHP geschriebenen Applikationen vom Layout zu trennen. Sie wird unter GPL auf http://www.php-application-tools.de veröffentlicht (sobald es eine Dokumentation gibt) und kann dann dort als Paket heruntergeladen werden.

Da die Klasse in PHP geschrieben wurde, setzt das natürlich voraus, dass die Applikation, die sie verwendet auch in PHP geschrieben wurde. Getestet wurde die Klasse mit PHP3 und PHP4.

5.1Anwendungsgebiete

Es gibt zwei grundsätzliche Gebiete, in denen die Klasse eingesetzt werden kann.

- statische Seiten, deren Inhalt nur manchmal auf Wunsch des Kunden geändert wird. Diese Seiten wurden bisher als reine HTML Seiten erstellt.
- Seiten, die dynamisch mit PHP erstellt werden, da sie dynamische Inhalte haben, z.B. Inhalte die aus einer Datenbank stammen.

5.1.1Statische Seiten

Diese Seiten wurden bisher nur als reine HTML Seiten geschrieben, wurden also nicht von PHP geparst. Um in statischen Seiten den Inhalt vom Layout zu trennen müssen sie zumindest dynamisch generiert werden. Für jede HTML Seite werden nun zwei Dateien benötigt, eine Layout Datei (mit HTML Code) und eine Inhaltsdatei. Es existiert bereits ein PHP Tool in der Beta Phase, das auf Basis einer HTML Datei, die nur Layoutangaben enthält und einer XML Datei, die nur Content enthält, mit Inhalt gefüllte HTML Dateien erstellt. Die beiden Einzeldateien können natürlich beliebig oft verwendet werden. So können z.B. alle Inhaltsdateien mit der selben Layout Datei verwendet werden oder auch eine Inhaltsdatei mit verschiedenen Layoutdateien, um den gleichen Inhalt auf unterschiedliche Art darzustellen (z.B. als HTML im Browser oder WML für WAP-fähige Handys).
Der Mehraufwand beim Erstellen der beiden Dateien rechnet sich natürlich nur, wenn entweder sehr viel Inhalt oder mehrere Layouts benötigt werden.

5.1.2Dynamische Seiten

Diese Seiten können wie bisher auch programmiert werden, lediglich die Ausgabe erfolgt mit Hilfe der Template Klasse, anstatt den HTML Code einfach per „echo“ auszugeben. Sowohl der HTML- als auch der PHP Programmierer müssen mit der Klasse vertraut sein, um den Ablauf reibungslos zu gestalten. Diese Art von Seiten werden vermutlich in der nächsten Zeit das häufigste Einsatzgebiet sein, weshalb ich die statischen Seiten erst einmal vernachlässigen will.

5.2Features

Haupteinsatzgebiet der Klasse ist wie oben erläutert das Einfügen von Inhalt in HTML Dateien und die Ausgabe dieser Dateien. Dazu werden im HTML Code Platzhalter, im weiteren Variablen genannt, der Form *{HEADLINE}* eingesetzt, die die Klasse später durch die eigentliche Überschrift ersetzen wird. Nach und nach wurde die Klasse durch zusätzliche Features erweitert, auf die ich hier kurz eingehen möchte:

- Verwendung mehrerer Dateien
  Die Klasse erlaubt es, HTML Code aus mehreren Dateien einzulesen, Platzhalter zu ersetzen und die entstandenen Teile auszugeben. Die Anzahl der einzelnen Dateien ist unbegrenzt. Jeder Teil wird durch einen eindeutigen Namen identifiziert und kann seine eigenen Variablen enthalten, deren Geltungsbereich auf diesen Teil begrenzt ist. Eine eingelesene Datei wird im Folgenden als ein Template bezeichnet, die Klasse, die all diese Templates enthält als Template Objekt. Jedes Template kann mit seinem Namen angesprochen werden, um seinen Variablen Werte zuzuweisen, sie auszugeben, usw.
- „Globale“ Variablen
  Im Gegensatz zu den oben beschriebenen Variablen, kann man auch „globale“ Variablen verwenden, deren Geltungsbereich für alle Dateien gilt und nicht begrenzt ist.
- Einlesen mehrerer Templates
  Die Klasse ermöglicht das Einlesen mehrerer Templates aus einer Datei. Dabei werden Anfang und Ende des Templates durch spezielle Tags gekennzeichnet, die es auch erlauben, diesen Templates einen Namen zu geben. So kann z.B. eine Standard HTML Seite mit Hilfe dieser Tags in Seitenkopf, Seitenkörper und Seitenfuß aufgeteilt werden und diese Bereiche können hinterher separat bearbeitet werden. Die HTML Seite lässt sich im Browser also weiterhin betrachten, wie eine normale HTML Seite.
- Automatische Wiederholung von Templates
  Eine Hauptanforderung an Web-Applikationen ist die Ausgabe von Listen. Deshalb unterstützt die Klasse patTemplate die automatische Wiederholung von Templates, um z.B. die Erzeugung von Listen auf Basis von Templates zu erleichtern.
- Alternierende Listen
  Werden Templates automatisch wiederholt, so hat man die Möglichkeit, zwei Versionen anzugeben, eine für gerade und eine für ungerade Zeilen.
- Verschachtelung
  Templates können in beliebige Tiefe verschachtelt werden, jedoch muss der Name weiterhin eindeutig sein. Wird ein Template ausgegeben, so werden alle Templates mit ausgegeben, die es enthält.
- „SubTemplates“, um alternative Inhalte anzugeben
  Es können verschiedene Inhalte für ein Template angegeben werden, um z.B. dem Problem „Es wurden 0/1/mehrere Suchergebnisse gefunden.“ gerecht zu werden.

Die Ausgabe wird von einer Variable abhängig gemacht werden (ähnlich der „switch“ Anweisung in verschiedenen Skriptsprachen). Es kann immer ein alternatives Template angegeben werden, falls kein Wert („empty“) und falls ein nicht definierter Wert („default“) übergeben wird.

- Templates können aus- und auch wieder eingeblendet werden. Natürlich kann dies nur vor der eigentlichen Ausgabe passieren, da die Seiten auf dem Server generiert werden.
- Templates können externe Dateien nachladen
  Damit kann z.B. eine Navigation automatisch in jede Seite eingebunden werden und muss nur einmal verändert werden. (ähnlich include() bei PHP, oder <script src="..."> bei HTML)

6Workflow HTML - PHP

Wie bereits in Kapitel 5 erläutert, geht es hauptsächlich um die Anwendung der Klasse bei dynamisch generierten Seiten. Der bisherige Workflow sieht vor, dass ein HTML Programmierer Beispielseiten erstellt, diese an den PHP Programmierer weitergibt, der dann in diese PHP Code einpflegt. Treten danach Korrekturen auf, muss zuerst der HTML Programmierer die Beispielseiten ändern und der PHP Programmierer diese Änderungen in die von ihm erstellten HTML/PHP Hybrid Seiten einpflegen.
Aus diesem Grund muss es leichte Änderungen am Workflow HTML – PHP geben: Der HTML Programmierer sollte Dateien erstellen, an denen der PHP Programmierer nichts ändern muss, um sie zu verwenden. Somit ist gewährleistet, dass im Falle einer Änderung nur die HTML-Vorlage, die Template Datei, geändert werden muss und der PHP Programmierer auf die geänderte Datei zugreifen kann, ohne an seiner Programmlogik etwas zu modifizieren. Um dies zu erleichtern, sollte der HTML Programmierer ein Grundverständnis der entstehenden Applikation haben und auch wissen, was mit der Seite, die er erstellt, später passieren soll.

Dieses Grundverständnis sollte folgende Bereiche umfassen:

- Welche Daten in der Seite werden dynamisch generiert? Woher kommen diese Daten?
- Werden Teile der Seite wiederholt? (z.B. Listen)
- Wenn es solche Wiederholungen gibt, soll die Liste z.B. alternierende Hintergrundfarben haben?
- Können an einer Stelle in der Seite unterschiedliche Teile dargestellt werden, je nachdem, welche Bedingung erfüllt ist? (z.B. „Es wurden x Suchergebnisse gefunden.“ oder „Leider keine Suchergebnisse gefunden.“
- Kann es sein, dass Teile der Seite unter bestimmten Bedingungen überhaupt nicht dargestellt werden?
- Gibt es Teile in der Seite, die auch in anderen Seiten des gleichen Projekts wieder auftauchen?

Um dieses Grundverständnis zu erlangen, ist es wichtig, die Kommunikation zwischen HTML- und PHP Programmierer zu maximieren. Alle Beteiligten sollten sich also vor der HTML Umsetzung zusammensetzen und den Seitenaufbau besprechen.

7Praxisanwendung und Beispiele

Um dem PHP Programmierer die Arbeit zu erleichtern, ist es also wichtig, Template-Seiten, statt reiner Bespielseiten zu erstellen. Dabei gibt es einiges zu berücksichtigen.

7.1HTML Strukturen

Am Wichtigsten ist es natürlich, sich sehr genau Gedanken über die Strukturierung der HTML Seiten zu machen. Die Seiten sollten so aufgebaut werden, dass es später einfach ist, wiederholbare Teile zu wiederholen, auch ohne das colspan Attribut anpassen zu müssen. Weiterhin sollte die Seite so modular aufgebaut werden, dass man einige Teile entfernen kann, ohne dass dies Einfluss auf die anderen Teile nimmt. Die Anforderungen an den Seitenaufbau hängen natürlich vom späteren Einsatz der Seite ab, also welche Teile werden wiederholt oder weggelassen, wo kann alternativer Content stehen, wo wird später Text von der Applikation eingesetzt.
Meist erreicht man dies durch einen geschickten Einsatz von Tabellen.

7.2Platzhalter

In den bisherigen Seiten hat man an Stellen, die später durch dynamischen Inhalt ersetzt werden sollen einfach Fantasiebegriffe wie z.B. „Karl Mustermann“ als Platzhalter für einen Kundennamen, geschrieben. An dieser Stelle gibt es die erste und gleichzeitig größte Änderung beim Erstellen von HTML Seiten: Platzhalter, oder auch Variablen. Im HTML Code können bestimmte Platzhalter definiert werden, die der PHP Programmierer später einfach durch den dynamischen Inhalt ersetzt. Diese Platzhalter werden zwischen geschweiften Klammern ({ }) eingeschlossen. Im Beispiel des Kundennamen sähe das folgendermaßen aus:

```
<table>
	<tr>
		<td>Kundenname:</td>
		<td>{CUSTOMER_NAME}</td>
	</tr>
</table>
```

Mit einem einfachen Befehl kann der PHP Programmierer den Platzhalter {CUSTOMER_NAME} durch den tatsächlichen Namen, den er z.B. aus einer Datenbank liest ersetzen und den ganzen HTML Code ausgeben. Er verwendet dazu:

```
$template->addVar( „meinTemplate“, „CUSTOMER_NAME“, „Stephan Schmidt“ );
```

Es ist natürlich wichtig, das HTML- und PHP-Programmierer die gleichen Namen für die Platzhalter verwenden. Will der PHP-Programmierer den Platzhalter CUSTOMER_NAME ersetzen, der HTML Programmierer verwendet allerdings den Platzhalter KUNDE_NAME, so wird bei der Ausgabe nichts passieren. Es ist also ratsam, sich hier genau abzusprechen. Die Wahl der Platzhalternamen ist vollkommen frei, sie müssen lediglich in Großbuchstaben geschrieben werden. Eventuell wäre es ratsam, innerhalb von Metrix Projekten Konventionen für die Namen von Platzhaltern festzulegen.
Wird ein Template wiederholt, so steht automatisch eine Variable für die Wiederholungsnummer zur Verfügung. {PAT_ROW_VAR} kann verwendet werden um z.B. Listeneinträge durchzunummerieren.

7.3Templates

In der Praxis werden einfache Seiten, bei denen lediglich ein paar skalare Platzhalter auftreten eher eine Seltenheit sein. Kunden lieben Listen. Seien es Adresslisten Ihrer Filialen, Produktlisten, oder auch Linklisten. Es hat sich herausgestellt, dass die Ausgabe einer Liste mit Informationen z.B. aus einer MySQL Tabelle die

Hauptanwendung ist. Deshalb verfügt die Klasse patTemplate über einige Funktionen, die einem bei der Ausgabe von Listen helfen können.

7.3.1Mehrere Dateien

Wie bereits angesprochen, kann ein Template Objekt mehrere HTML-Templates, oder auch Schnipsel, also Teile einer HTML Seite enthalten. Jeder Teil kann benannt werden und später über diesen Namen angesprochen werden. Diese einzelnen Schnipsel können z.B. in verschiedenen Dateien abgelegt werden. Der HTML Programmierer muss sich nur darüber klar werden, wie er seine HTML Seite „zerlegt“. In Falle einer Liste wäre folgende Unterteilung ratsam:

- Listenkopf
- Listeneintrag, der eventuell wiederholt werden kann
- Listenfuss

Der HTML Code dieser einzelnen Teile sähe folgendermaßen aus:

Listenkopf:
<table border=“0“ cellpadding=“2“ cellspacing=“2“>

Listeneintrag:
```
<tr>
        <td>{CUSTOMER_NAME}</td>
        <td>{CUSTOMER_EMAIL}</td>
</tr>
```

Listenfuss:
</table>

Die Applikation liest dann diese drei Dateien ein, gibt zuerst den Listenkopf aus, danach den Listeneintrag mit beliebig vielen Wiederholungen und am Ende den Listenfuss. Der eindeutige Nachteil dieser Lösung ist die große Anzahl an Dateien, die erstellt werden müssen und einzeln nicht mehr im Browser betrachtet werden können.
Es müssen also Steuercodes implementiert werden, die es ermöglichen, eine HTML Datei durch die Template Klasse in beliebig viele Einzeltemplates zu unterteilen, ohne dabei die Ausgabe der Datei ohne PHP in einem normalen Browser zu behindern. Naheliegend war also die Einführung neuer Tags, da diese, sofern sie dem Browser unbekannt sind bei der Ausgabe einfach ignoriert werden.

7.3.2Mehrere Templates in einer Datei

Um mehrere Templates in einer HTML Datei zu halten, werden <patTemplate:tmpl> Tags verwendet. Um die drei Template-Teile, die für die obige Liste verwendet wurden in einer Datei zu definieren, wird folgende Syntax verwendet:

```
<patTemplate:tmpl name=“listenkopf“>
<table border=“0“ cellpadding=“2“ cellspacing=“2“>
</patTemplate:tmpl>

<patTemplate:tmpl name=“listeneintrag“>
<tr>
        <td>{CUSTOMER_NAME}</td>
        <td>{CUSTOMER_EMAIL}</td>
```

```
</tr>
</patTemplate:tmpl>


<patTemplate:tmpl name="listenfuss">
</table>
</patTemplate:tmpl>
```

Diese Datei wird von der Template Klasse eingelesen und es werden automatisch drei Templates erstellt:

- listenkopf
- listeneintrag
- listenfuss

Die Templates enthalten jeweils den HTML Code, der zwischen dem öffnenden und schließenden <patTemplate:tmpl> Tags steht. Das erste Attribut, das der <patTemplate:tmpl> Tag enthalten kann und auch muss ist das *name="..."* Attribut, mit dem das folgende Template eindeutig identifiziert werden kann und über welchen es auch ausgegeben wird. Die Templates können also wie im ersten Beispiel verwendet werden, in dem sie noch in unterschiedlichen Dateien waren.

Achtung: *Sowohl der öffnende, als auch der schließende Tag müssen alleine in einer Zeile stehen. Das einzige Attribut, dass jeder <patTemplate:tmpl> Tag enthalten MUSS ist das name="..." Attribut. Jeder Name darf nur einmal auftreten.*

7.3.3Verschachtelte Templates

Um die Liste im obigen Beispiel auszugeben, müssen drei Teile nacheinander ausgegeben werden, es werden also drei PHP Befehle benötigt, obwohl im Normalfall immer die gesamte Liste ausgegeben werden soll. Um dies zu umgehen, gibt es eine Möglichkeit, die drei Templates zu „gruppieren" und dann unter dem neuen Namen „liste" anzusprechen, die Templates müssen dazu verschachtelt werden:

```
<patTemplate:tmpl name="liste">
        <patTemplate:tmpl name="listenkopf">
<table border="0" cellpadding="2" cellspacing="2">
        </patTemplate:tmpl>

        <patTemplate:tmpl name="listeneintrag">
<tr>
        <td>{CUSTOMER_NAME}</td>
        <td>{CUSTOMER_EMAIL}</td>
</tr>
        </patTemplate:tmpl>

        <patTemplate:tmpl name="listenfuss">
</table>
        </patTemplate:tmpl>
</patTemplate:tmpl>
```

Nachdem die Datei eingelesen wurde, stehen vier Templates zur Verfügung:

- liste
- listenkopf

- listeneintrag
- listenfuss

Das Template Liste enthält keinen eigentlichen HTML Code, sondern lediglich die Templates „listenkopf“, „listeneintrag“ und „listenfuss“. Es ist also möglich über einen Aufruf die ganze Liste auszugeben und es kann weiterhin ein Teil der Liste, durch Zuweisung mehrerer Werte, wiederholt werden.

Eigentlich sind jedoch die beiden Templates „listenkopf“ und „listenfuss“ überflüssig, da beide nur einmal auftreten und auch nur, wenn die ganze Liste ausgegeben wird. Wenn diese beiden Templates weggelassen werden sollen, sieht die Syntax folgendermaßen aus:

```
<patTemplate:tmpl name="liste">
<table border="0" cellpadding="2" cellspacing="2">
        <patTemplate:tmpl name="listeneintrag">
<tr>
        <td>{CUSTOMER_NAME}</td>
        <td>{CUSTOMER_EMAIL}</td>
</tr>
        </patTemplate:tmpl>
</table>
</patTemplate:tmpl>
```

Nach dem Einlesen der Datei steht das Template Liste zur Verfügung, das folgendermaßen aussieht:

```
<table border="0" cellpadding="2" cellspacing="2">
{TMPL:listeneintrag}
</table>
```

An die Stelle *{TMPL:listeneintrag}* wird bei der Ausgabe automatisch der Inhalt des Template listeneintrag gesetzt, das bei Bedarf wiederholt werden kann. Auf diese Art und Weise hat man einfachen Zugriff auf die gesamte Liste, der eigentliche Listenkörper ist aber noch von der gesamten Liste abgekoppelt, um ihn separat mit Inhalt zu füllen (durch Variablen Ersetzung).

7.3.4Sub Templates

Bisher konnte an einer Stelle immer nur ein bestimmter Inhalt stehen, der maximal wiederholt werden konnten. Es treten jedoch häufig Anforderungen auf, bei denen an einer Stelle der Seite je nach erfüllter Bedingung unterschiedlicher HTML Code stehen muss (z.B. Fehlermeldungen in rot, statt normalem Text oder Listen, bei denen die Hintergrundfarbe zeilenweise wechselt). Aus diesem Grund wurden Sub Templates eingeführt.
Bisher gibt es zwei Einsatzgebiete für Subtemplates:

7.3.4.1Listen mit alternierenden Zeilentypen

Als Beispiel soll in unserer Liste der Hintergrund abwechselnd in einer Zeile hellgrau und in einer Zeile dunkelgrau dargestellt werden. Die einfachste Lösung wäre natürlich eine Variable BGCOLOR, die jetzt Zeile auf einen anderen Wert gesetzt wird. Das würde jedoch gegen die strikte Trennung verstoßen, schließlich ist die Hintergrundfarbe ein Teil des Layouts und gehört somit nicht in die PHP Datei. Es muss also eine

Möglichkeit geben, zwei Templates anzugeben, eines für gerade und eines für ungerade Zeilen, in denen man frei allen HTML Code verwenden kann. Die Syntax für die Liste lautet:

```
<patTemplate:tmpl name="liste">
<table border="0" cellpadding="2" cellspacing="2">
        <patTemplate:tmpl name="listeneintrag" type="OddEven">

                <patTemplate:sub condition="even">
<tr bgcolor="#EEEEEE">
        <td>{CUSTOMER_NAME}</td>
        <td>{CUSTOMER_EMAIL}</td>
</tr>
                </patTemplate:sub>

                <patTemplate:sub condition="odd">
<tr bgcolor="#CCCCCC">
        <td>{CUSTOMER_NAME}</td>
        <td>{CUSTOMER_EMAIL}</td>
</tr>
                </patTemplate:sub>

        </patTemplate:tmpl>
</table>
</patTemplate:tmpl>
```

Neu hinzugekommen sind das type="..." Attribut und der <patTemplate:sub> Tag.
Die neuen <patTemplate:sub> Tags umschließen jeweils HTML Code, der sich sehr ähnelt und sich nur in der Hintergrundfarbe unterscheidet. Als condition="..." Attribut wird einmal even (gerade) und einmal odd (ungerade) angegeben um für ein Template zwei Ausgabemöglichkeiten zu speichern, eine für gerade und eine für ungerade Zeilen.

Das type="OddEven" Attribut im <patTemplate:tmpl> Tag veranlasst die Klasse, das folgende Template nicht als normales Template abzulegen, sondern als alternierendes Template. Bei der Ausgabe entscheidet die Klasse anhand der Zeilennummer, welches Sub Template ausgegeben werden muss. Variablen können in den Subtemplates ganz normal verwendet werden. Zugriff auf das Template erfolgt weiterhin über den Namen, auf die Sub Templates kann nicht direkt zugegriffen werden.

In den Subtemplates kann das Condition Attribut nur die Werte „odd" oder „even" annehmen.

7.3.4.2Condition Templates

Eine erneute Anforderung ist die Ausgabe von HTML Code, je nachdem, welche Bedingung erfüllt wurde. Ein gutes Beispiel dafür ist die Ausgabe einer Zeile Text über einer List von Suchergebnissen. Es können drei Fälle auftreten:

1.Es wurden keine Ergebnisse gefunden.
2.Es wurde genau ein Ergebnis gefunden.
3.Es wurde mehr als ein Ergebnis gefunden.

Die Anzahl der Suchergebnisse steht in PHP in der Variable ERGEBNISSE zur Verfügung und je nach Fall soll ein anderes Layout möglich sein. Dazu kann ein Teil der Logik ans Template übergeben werden.
Auch hierzu wieder die Syntax des Beispiels:

```
<patTemplate:tmpl name="listeneintrag" type="Condition" conditionvar="ERGEBNISSE">

    <patTemplate:sub condition="0">
        <font color="#FF0000">Leider keine Ergebnisse gefunden.</font>
    </patTemplate:sub>

    <patTemplate:sub condition="1">
        <font color="#000000">Folgende Seite passt zu Ihren Suchkriterien:</font>
    </patTemplate:sub>

    <patTemplate:sub condition="default">
        <font color="#000000">Zu Ihren Suchkriterien wurden {ERGEBNISSE} Seiten
        gefunden:</font>
    </patTemplate:sub>

</patTemplate:tmpl>
```

Die erste Zeile definiert das nachfolgende Template als Condition Template, d.h. der dargestellte HTML Code ist abhängig von einer Bedingung. Als Bedingung kann bisher nur eine Variable, die übergeben wird mit einem Wert verglichen werden. In diesem Fall soll die Variable *ERGEBNISSE* überprüft werden, die die Anzahl der Suchergebnisse enthält. Danach folgende drei Subtemplates, die den HTML Code für die drei möglichen Fälle enthalten. Hierbei wird wieder das Attribut *condition="..."* verwendet, wobei die Bedingungen dieses Mal frei gewählt werden können.
Wird das Template ausgegeben, so wird also überprüft, ob eine 0 oder eine 1 für die Variable *ERGEBNISSE* übergeben wurde und dementsprechend einer der Inhalte für das Template ausgegeben. Wurde weder 0 noch 1 übergeben tritt ein Sonderfall ein. Für ein Condition Template gibt es zwei vordefinierte Werte, die als condition="..." Attribut verwendet werden können:

- *empty*
  Wird kein Wert für die Variable übergeben, so wird das Sub Template mit der Bedingung empty ausgegeben, falls es existiert.
- *default*
  Wird ein Wert für die Variable übergeben, für den kein Sub Template angelegt wurde, so wird der Inhalt des Sub Templates mit der Bedingung default ausgegeben, falls es existiert.

Wer schon etwas Programmiererfahrung hat, wird feststellen, dass das Condition Templates stark der switch Anweisung ähnelt, die es in einigen Programmiersprachen gibt.

7.3.5Templates wiederholen

Will man einen Bereich unabhängig von den gesetzten Variablen wiederholen, so ist das über das *loop* Attribut des *<patTemplate:tmpl>* Tags möglich.Beispiel:

```
<patTemplate:tmpl name="wiederholung" loop="5">
{PAT_ROW_VAR}. <input type="text" name="feld{PAT_ROW_VAR}"><br>
</patTemplate:tmpl>
```

Dieses Beispiel würde 5 Formularfelder erzeugen, deren Namen mit *feld1* bis *feld5* durchnummeriert sind.

7.3.6Templates ausblenden

In Web Applikationen treten oft Teile in einer HTML Seite auf, die nur bei manchen Bedingungen angezeigt werden sollen, wie z.B. Fehlermeldungen. Eine Möglichkeit, dieses Problem mit Templates zu lösen wäre natürlich ein Condition Template, dem man z.B. für die Variable *ERROR* den Wert „show“ übergibt, für den ein Subtemplate existiert, das einen Fehler ausgibt oder den Wert „hide“, für den ein leeres Subtemplate angelegt wurde. Das wäre jedoch mit Sicherheit zu rechenintensiv und auch zu umfangreich, das Template für jede Fehlermeldung anzulegen. Aus diesem Grund gibt es für den Tag <patTemplate:tmpl> das Attribut *visibility=“...“* das die Werte „visible“ oder „hidden“ enthalten kann. Wird das Attribut auf „hidden“ gesetzt, so wird dieses Template bei der Ausgabe nicht mit ausgegeben, kann aber mit einer Zeile von PHP aus wieder eingeblendet werden.
Beispiel:

```
<patTemplate:tmpl name="liste">
<table border="0" cellpadding="2" cellspacing="2">
        <patTemplate:tmpl name="error" visibility="hidden">
<tr>
        <td colspan="2">{Es ist ein Fehler aufgetreten!</td>
</tr>
        </patTemplate:tmpl>
        <patTemplate:tmpl name="listeneintrag">
<tr>
        <td>{CUSTOMER_NAME}</td>
        <td>{CUSTOMER_EMAIL}</td>
</tr>
        </patTemplate:tmpl>
</table>
</patTemplate:tmpl>
```

Sollte das Template *error* nicht von PHP aus wieder eingeblendet werden, so wird es bei der Ausgabe nicht ausgegeben.

7.3.7Templates aus externen Dateien

Setzt man ganze Websites auf Basis von Templates um, stößt man oft auf Seitenelemente, die auf fast jeder oder jeder Seite auftreten. Diese Teile müssten also auch als HTML Code in jedes Template eingebunden werden. Wird dieser Teil verändert, muss er in jedem Template verändert werden. Das kann unter Umständen einen sehr großen Arbeitsaufwand bedeuten. Aus diesem Grund wurde das *src=“...“* Attribut eingeführt. Es ermöglicht den Inhalt eines Templates aus einer externen Datei nachzuladen.

7.3.7.1Reine HTML Dateien

Die einfache Variante mit externen Dateien umzugehen ist, nur den eigentlichen HTML Code auszulagern und das *src=“...“* Attribut des *<patTemplate:tmpl>* Tags zu verwenden. Die Struktur wird weiterhin über die eigentliche Template Datei definiert.

Syntax:

```
<patTemplate:tmpl name="seite">
        <patTemplate:tmpl name="seitenkopf" src="seitenkopf.html">
        </patTemplate:tmpl>
```

```
        <patTemplate:tmpl name="seiteninhalt">
<table border="0" cellpadding="2" cellspacing="2">
        <tr>
                <td colspan="2">Hier steht der eigentliche Seiteninhalt.</td>
        </tr>
</table>
        </patTemplate:tmpl>
        <patTemplate:tmpl name="seitenfuss" src="seitenfuss.html">
        </patTemplate:tmpl>
</patTemplate:tmpl>
```

Das Template „seite“ enthält jetzt drei Templates: Seitenkopf, Seiteninhalt und Seitenfuß, wobei der Inhalt von Seitenkopf und Seitenfuß aus externen Dateien nachgeladen werden.

7.3.7.2Einschub: Leere XML Tags

Bisher gehörte zu einem öffnenden Tag stets auch ein schließender <patTemplate> Tag, da beide den HTML Code oder weitere Tags umschlossen. Arbeitet man mit externen Dateien, kann es vorkommen, dass zwischen dem öffnenden und schließenden Tag keine weiteren Daten stehen, der schließende Tag ist also überflüssig. Auch in HTML gibt es solche Tags, wie z.B. den
 oder <img> Tag. In XML dürfen solche Tags jedoch nicht auftreten, jeder öffnende Tag muss stets geschlossen werden. Um solche leeren Tags darzustellen bedarf es also einer speziellen Syntax. Ein leerer Tag endet in XML stets mit einem Slash (/), die Schreibweise für den
 Tag wäre XML konform also
. Genau so können auch die <patTemplate:tmpl> Tags verwendet werden, wenn sie keine weiteren Daten enthalten:

```
<patTemplate:tmpl name="leer" src="meinHTML.html"/>
```

Dieser Tag würde als vollwertiges Template erkannt werden und hätte als Inhalt den Inhalt der HTML Datei meinHTML.html. Genauso könnte man im obigen Beispiel leere Tags verwenden, anstatt einen Tag zu öffnen und sofort wieder zu schließen.

Schreibt man jedoch zwischen den öffnenden und schließenden <patTemplate:tmpl> Tag weiteren HTML Code, so wird er an den Inhalt des Templates angehängt.

7.3.7.3externe Template Dateien

Es ist jedoch auch möglich, dem Parser mitzuteilen, dass er die externen Dateien auch auf <*patTemplate*> Tags durchsuchen soll. Dazu wird das *parse="..."* Attribut des <*patTemplate:tmpl*> Tags auf „*on*“ gesetzt. Der Parser behandelt den Inhalt der externen Datei dann so, als ob er im Inhalt des <*patTemplate:tmpl*> Tags stehen würde, d.h. die neuen Templates aus der externen Datei sind Teile des umgebenden Templates. Die externe Datei kann alle Tags enthalten, die von patTemplate bisher unterstützt werden und auch wieder externe Dateien einbinden.
Will man nur den Inhalt einer externen Datei parsen, kann auch hier mit leeren Tags gearbeitet werden, steht HTML Code zwischen dem öffnenden und schließenden Tag, so wird er wiederum an den Inhalt des Templates angehängt.

7.3.8Mehrfach auftretende Templates

In Webseiten gibt es häufig Teile, die mehrfach in einer Seite auftreten, wie z.B. horizontale Trennbalken, die den Inhalt strukturieren sollen. Diese Teile sollen immer 100% identisch sein, um ein einheitliches Erscheinungsbild zu gewährleisten. Um dies zu vereinfachen und auch den eigentlichen HTML Code, der vom Template Parser eingelesen wird, zu minimieren, wurde der Tag *<patTemplate:link>* eingeführt. Er kann verwendet werden, um ein Template beliebig oft an beliebigen Stellen in der Seite darzustellen. Er unterstützt bisher nur ein einziges Attribut, das *src* Attribut, mit dem festgelegt wird, welches Template anstelle des Tags dargestellt werden soll. Da der *<patTemplate:link>* Tag keinen Inhalt hat, wird er normalerweise als leerer Tag geschrieben:

```
<patTemplate:tmpl name="page">
Willkommen auf meiner Homepage.

        <patTemplate:tmpl name="horizontal_bar">
<hr width="80%" size="2">
        </patTemplate:tmpl>

ich hoffe euch gefällt es hier!

        <patTemplate:link src="horizontal_bar"/>
</patTemplate:tmpl>
```

In diesem Beispiel würde der Trennbalken zweimal dargestellt werden, müsste aber bei einer Änderung nur einmal geändert werden. Dieser Prozess kann optimiert werden, in dem der horizontale Trenner ausgelagert wird und über das *src* Attribut des *<patTemplate:tmpl>* Tags auf allen Seiten einmal eingebunden werden muss und dann dort beliebig oft verwendet werden kann.
Leider können die Templates bisher nicht mit unterschiedlichem Inhalt gefüllt werden, da keine Kopie erzeugt wird, sondern immer das gleiche Template ausgegeben wird, d.h. der *<patTemplate:link>* Tag eignet sich nicht zur Ausgabe von Content, sondern nur zur Wiederholung von Layout Elementen.

7.3.9Templates einlesen, aber nicht ausgeben

Der Vollständigkeit halber wird auch noch auf dieses Feature eingegangen, das eigentlich recht selten gebraucht wird. Werden verschachtelte Templates gefunden, so wird normalerweise im äußeren (parent) Template ein Platzhalter eingefügt, damit die Template Klasse weiß, an welcher Stelle sie das enthaltene Template einsetzen muss. Dieser Platzhalter (placeholder) hat das Format *{TMPL:templatename}*. Es kann aber zu Situationen kommen, in denen das Einsetzen dieses Platzhalters unerwünscht ist, z.B. wenn man einen XML Parser schreiben möchte, der Seiten aus statischen XML Dateien erzeugt. Um die Ausgabe des Platzhalters zu ändern, oder zu unterdrücken gibt es das *placeholder="..."* Attribut des <patTemplate:tmpl> Tags. Das Attribut kann jeden Wert annehmen, beim Einlesen des Templates wird an die Stelle des verschachtelten Templates der Wert geschrieben, den das placeholder Attribut annimmt. Will man keinen Platzhalter, so verwendet man *placeholder="none"*.
Beispiel:

Ohne placeholder Attribut:

```
<patTemplate:tmpl name="seite">
```

```
Gleich kommt der Seiteninhalt.
        <patTemplate:tmpl name="seiteninhalt">
<table border="0" cellpadding="2" cellspacing="2">
        <tr>
                <td colspan="2">Hier steht der eigentliche Seiteninhalt.</td>
        </tr>
</table>
        </patTemplate:tmpl>
</patTemplate:tmpl>
```

Nach dem Einlesen der Datei stehen zwei Templates zur Verfügung:

- seite

  Gleich kommt der Seiteninhalt.
  {TMPL:seiteninhalt}

- seiteninhalt
  enthält den eigentlichen HTML Code

Wird das Template *seite* ausgegeben, so wird automatisch das Template *seiteninhalt* ausgegeben.

Mit placeholder Attribut:

```
<patTemplate:tmpl name="seite">
Gleich kommt der Seiteninhalt.
        <patTemplate:tmpl name="seiteninhalt" placeholder="INHALT">
<table border="0" cellpadding="2" cellspacing="2">
        <tr>
                <td colspan="2">Hier steht der eigentliche Seiteninhalt.</td>
        </tr>
</table>
        </patTemplate:tmpl>
</patTemplate:tmpl>
```

Nach dem Einlesen der Datei stehen wiederum zwei Templates zur Verfügung:

- seite

  Gleich kommt der Seiteninhalt.
  INHALT

- seiteninhalt
  enthält den eigentlichen HTML Code

Wird jetzt das Template *seite* ausgegeben, wird das Template *seiteninhalt* nicht mit ausgegeben.

8Tagreferenz

Tag/Attribut	Mögliche Werte	Beschreibung
<patTemplate:tmpl>		Kennzeichnet den Anfang eines neuen Templates. Alles, was bis zum schließenden Tag folgt, ist Teil dieses Templates, sowohl reines HTML, als auch weitere Templates
name	string	Benennt das Template. Der Name muss eindeutig sein.
type	*Standard* *OddEven* *Condition*	Kennzeichnet den Typ des Templates. Standard Templates enthalten keine Subtemplates. OddEven Templates können zwei Subtemplates enthalten, eines für gerade und für ungerade Wiederholungen Condition Templates enthalten beliebige Subtemplates.
conditionvar	string	Kann nur verwendet werden, wenn type="Condition" gesetzt wurde. Diese Variable wird verwendet, um ein Template aus den folgenden Subtemplates zu wählen.
loop	integer	Wiederholt ein Template, auch wenn nicht so viele Variablen gesetzt wurden.
visibility	*visible* *hidden*	Wird ein Template ausgegeben, oder nicht.
placeholder	string *none*	Ändert den Platzhalter, der in das umgebende Template eingefügt wird. Wenn das Schlüsselwort none verwendet wird, wird kein Platzhalter eingesetzt.
src	string	Externe Quelle für das Template. Als Inhalt des Templates wird der Inhalt einer externen Datei verwendet.
parse	*on* *off*	Kann nur in Verbindung mit src verwendet werden. Gibt an, ob der Inhalt der Datei einfach nur eingelesen werden soll (off), oder ob die externe Datei auch den Parser durchlaufen soll (on).
<patTemplate:sub>		Kennzeichnet den Anfang eines Subtemplates. Ein Subtemplate kann nur innerhalb eines normalen Templates auftreten und auch nur, wenn dieses vom Typ „OddEven" oder „Condition" ist.
condition	mixed *empty* *default*	Wird verwendet, um die Bedingung für ein Subtemplate zu setzen. Bei der Ausgabe wird überprüft, ob die mit conditionvar="..." im <patTemplate:tmpl> gesetzte Variable gleich dem Wert in condition="..." ist. Stimmen die Werte überein, wird für das Template dieses Subtemplate verwendet. Wird als Wert empty gesetzt, wird dieses Subtemplate verwendet, wenn kein Wert für die Variable übergeben wurde. Wird default gesetzt, so wird das Template verwendet, falls kein passendes Template gefunden wurde.
<patTemplate:link>		Wird verwendet, um ein Template an mehreren Stellen innerhalb einer Seite auszugeben
src	string	Name des Templates, dass an dieser Stelle dargestellt werden soll.

9Testumgebung

Es ist ratsam, erstellte Templates erst zu testen, bevor man sie an den PHP Programmierer übergibt. Da Testen jedoch nur mit PHP möglich ist, habe ich auf dem Metrix Entwicklungsserver devel eine Testumgebung eingerichtet. Unter der URL http://devel/pat/patTemplate/Schulung/upload.php findet man ein Formular, über das man Templates uploaden kann und entweder die geparsten Templates oder deren Struktur ausgeben kann. Bis jetzt ist es noch nicht möglich, mehrere Dateien auf einmal hoch zu kopieren, Templates die externe Dateien mit Hilfe des src="..." Attributs nachladen, funktionieren also nicht.
In Kürze soll es auch möglich sein, den Templates Variablen über ein Webinterface zuzuweisen, um somit automatische Wiederholungen zu testen.