Studienarbeit

# LateXHTML: Markup-basierte PDF-Generierung mit LaTeX


Institut für Informatik
Fakultät für Angewandte Wissenschaften
Albert-Ludwigs-Universität
Freiburg i. Br.

Lehrstuhl für Datenbanken und Informationssysteme
Prof. Dr. Georg Lausen
Betreuer: Martin Weber, Matthias Ihle

Nils Andre
nilsandre@gmail.com
20. April 2006

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Das Bereitstellen von Daten in mehreren Ausgabeformaten wird vor allem durch das Internet immer wichtiger. Neben der zumeist aufwendigen Präsentation von Inhalten in einer Webseite (beispielsweise für moderne Browser) sollen zusätzlich barrierefreie, reduzierte Dokumente bereitgestellt werden, welche auch von alten Webbrowsern richtig darstellt werden. Durch den ständigen Wandel von Domains und Links sollen Inhalte zusätzlich in dauerhaft auf Clientrechnern speicherbaren Formaten bereitgestellt werden. Ein geeignetes Dokumentformat hierfür ist inbesondere das PDF-Format[1]. Dies ist ein offengelegtes und gut dokumentiertes Dateiformat für druckbare Dokumente, welches von der Firma Adobe entwickelt und veröffentlicht wurde. Ein wichtiger Vorteil von PDF-Dokumenten ist, daß sie plattformübergreifend auf nahezu allen Betriebssystemen korrekt und eindeutig dargestellt werden können.

Liegen Informationen in einer Datenbank vor, welche bereits durch Content-Management-Systeme[2] verwaltet und in Webseiten dargestellt werden, ist es naheliegend, diese auch in Form von PDF-Dokumenten bereitzustellen. Da Änderungen der Inhalte durch ein CMS meistens automatisch neue Webseiten generieren oder bestehende aktualisieren, ist es wünschenswert, daß sich in Bezug auf die PDF-Dokumente die Änderungen ähnlich „automatisiert“ auswirken.

Es gibt etliche komerzielle und freie Programme, welche PDF-Dokumente automatisiert erstellen können. Als wichtiges komerzielles Produkt ist etwa der Adobe Document Server[3] zu nennen. Dieser ließt XML-Daten sowie Vorlagen oder andere PDF-Dateien ein und erstellt serverseitig das Ausgabedokument.

[1] „Portable Document Format“

[2] abgekürzt „CMS“, Software zur Trennung von Inhalt und Design von Internetdokumenten

[3] URL: `http://www.adobe.de/products/server/documentserver/main.html`

Nichtkommerzielle Software leistet auf vereinfachte Art ähnliches. Hierbei werden PDF-Dokumente oftmals direkt erzeugt, d.h. es werden gemäß der PDF-Spezifikation[4] Binärdaten in eine Datei geschrieben. Wiederum andere Lösungen basieren darauf, zuerst Microsoft-Word-Dokumente zu erzeugen und diese dann durch einen serverseitigen „PDF-Drucker“ konvertieren zu lassen. Derartige Lösungen berücksichtigen jedoch wichtige Aspekte wie die Gefahr des unkontrollierten „Auslaufens von großen Objekten“ über den Seitenrand weitestgehend nicht.

## 1.1 Problemformulierung und Ziel der Studienarbeit

Bei der automatischen Generierung von PDF-Dokumenten ergeben sich vorab folgende Fragestellungen:

- Wie werden die PDF-Dokumente erzeugt? Welche Software legt man der Generierung zugrunde?
- Inwiefern kann man das „Aussehen“ von automatisch generierten PDF-Dokumenten beeinflussen?
- Welche Probleme ergeben sich bei der Gestaltung des Seitenlayouts?
- Wie kann man die Struktur der PDF-Dokumente möglichst „intuitiv“ beschreiben?

Da mit LaTeX bereits eine mächtige Sprache für das Setzen von Dokumenten existiert, fiel die Wahl zur Generierung der PDF-Dokumente auf PDFLaTeX. Dieses ist freie Software, welche für alle gängigen Betriebssysteme verfügbar ist, und beinhaltet bereits viele Möglichkeiten, das Seitenlayout sowie den Schriftsatz von Dokumenten in hoher Qualität zu gestalten. Jedoch ist LaTeX in einigen Fällen sehr unintuitiv. Daher wird eine Pseudosprache entworfen, welche vor allem das intuitive Erstellen von Templates[5] durch WYSIWYG[6] in den Vordergrund stellt.

[4] URL: `http://partners.adobe.com/public/developer/pdf/index_reference.html`
[5] englich für „Vorlage“
[6] Abkürzung für „what you see is what you get“

Des Weiteren muss das in dieser Arbeit vorgestellte System sicherstellen, dass Tabellen, welche beliebig viele (aus einer Datenbank abgefragte) Datensätze beinhalten können, korrekt auf einzelne Seiten aufgeteilt werden und nicht über das Seitenende hinauslaufen.

In dieser Studienarbeit wird das System LateXHTML vorgestellt. Es erlaubt es, auf der Basis von XHTML erstellte Templates zunächst mit Daten aus einer Datenbank zu „befüllen“ sowie diese in LaTeX-Code zu übersetzen. Es ist als eigenständiges System in der Webscript-Sprache PHP[7] implementiert und benötigt lediglich eine LaTeX-Installation.

## 1.2 Aufbau der Studienarbeit

Die Studienarbeit gliedert sich in fünf Kapitel.

Nach der Einleitung wird in Kapitel 2 die Software LateXHTML vorgestellt. Nach der einer Skizze des generellen Ablaufs wird die Beschreibungssprache für die Templates entworfen (es werden die Syntax und die verfügbaren Tags und Attribute sowie der Prozess des Übersetzens in LaTeX erörtert). Dann wird die Template-Engine eingeführt, welche dazu dient, datenbankgestützte Inhalte (durch das Hinzufügen von Platzhaltern und wiederholbaren Bereichen sowie die Definition von Tabellenkopf- und -fußzeilen) in die Templates einfügen zu können.

Anschließend wird in Kapitel 3 analysiert, welche Möglichkeiten zur Verfügung stehen, über das Seitenende hinauslaufende Tabellen durch LateXHTML automatisch zu erkennen und neu berechnen zu lassen, so dass diese über mehrere Seiten korrekt aufgeteilt werden.

In Kapitel 4 finden sich technische Anmerkungen und Notizen zur Erweiterung des Systems. Dort wird beschrieben, wie das System auf einem Server installiert und ggf. in bestehende Applikationen eingebettet werden kann. Ebenfalls wird die Struktur des Quelltextes und die beigefügten Beispielanwendungen beschrieben.

Kapitel 5 beinhaltet ein Fazit und Ausblicke.

[7] Siehe `http://www.php.net/` für weitere Informationen

# Kapitel 2

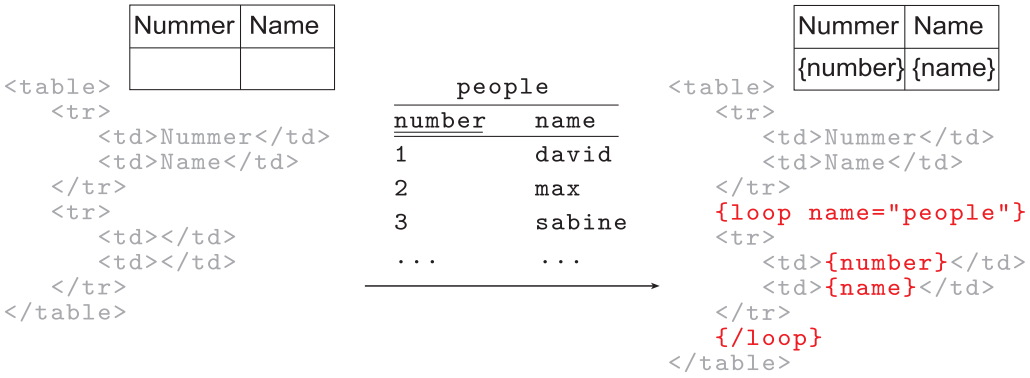
# Funktionsweise und Anwendungsablauf

In diesem Kapitel wird zunächst auf die generelle Funktionsweise von LateXHTML eingegangen. Im weiteren Verlauf wird detailliert die Beschreibungssprache der Templates erörtert (Abschnitt 2) und auf die Verarbeitung dieser durch die Template-Engine (Abschnitt 3) eingegangen sowie weitere, zur Dokumentgenerierung benötigte Komponenten (Abschnitt 4) dokumentiert.

## 2.1 Aufbau der Software und Anwendung

Um LateXHTML benutzen zu können, müssen – wie bereits erwähnt – zunächst Templates erstellt, geeignete Daten hierfür ausgewählt, sowie ein Grundlayout angepasst werden. Die Templates stellen den Kern der von der Software verarbeiteten Daten dar. Diese im folgenden Abschnitt genauer beschriebenen Vorlagen definieren das „Aussehen“ der Inhalte der zu erzeugenden PDF-Datei mittels einer an XHTML angelehnten Sprache. Sollen sich die Inhalte der PDF-Dokumente dynamisch verändern, so müssen geeignete Datenquellen bereitgestellt werden und die Templates daraufhin angepasst werden. Dies ermöglichen so genannte Platzhalter, welche an die Datenquelle gebunden werden. Dies wird durch Abbildung 2.1 verdeutlicht.

Des Weiteren muß noch ein Layout erstellt werden, welches als globale PDF-Dokumenteinstellungen (beispielsweise Seitenmaße und Schriftgrößen) festlegt. Dies wird in Abschnitt 4 erläutert.




Eine Tabellentemplate (links) wird durch geeignete Platzhalter (rechts) an Daten einer Relation (hier `people`) gebunden

**Abbildung 2.1:** Bindung eines Templates

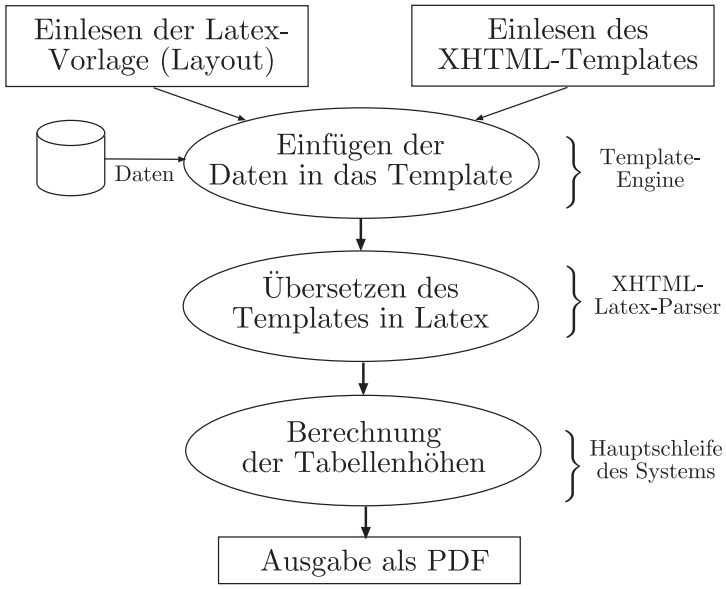



**Abbildung 2.2:** Anwendungsablauf

Die so definierten Templates und Daten werden dann durch LateXHTML verarbeitet:

- Die Template-Engine ersetzt die im Template eingebetteten Platzhalter durch konkrete Daten.
- Der XHTML-Latex-Parser übersetzt das entstandene „befüllte“ Template in LaTeX-Code.
- In der Hauptschleife wird PDFLaTeXaufgerufen und Tabellen, welche durch das Ersetzen der Platzhalter durch konkrete Daten „zu groß“ geworden sind, auf mehrere Seiten aufgeteilt.

Am Ende dieses Prozesses entsteht entweder eine DVI- oder PDF-Datei. Abbildung 2.2 verdeutlicht dies.

## 2.2 Die Beschreibungssprache der Templates

In diesem Abschnitt wird die gesonderte Beschreibungssprache für Templates entwickelt sowie die Notwendigkeit eines solchen „Zwischenschrittes“ begründet. Des Weiteren werden die Sprachelemente dokumentiert und an Beispielen erörtert.

### 2.2.1 XHTML als Grundlage

Um ein Templatelayout möglichst intuitiv erzeugen zu können, wird eine auf XHTML basierende Pseudosprache verwendet, welche dann in LaTeX-Code übersetzt wird. Dies erleichtert ein schnelles Erstellen mittels WYSIWYG-Editoren wie beispielsweise Dreamweaver oder GoLive und hat den zusätzlichen Vorteil, dass XHTML sehr weit verbreitet und – zumindest in Grundlagen – als bekannt vorausgesetzt werden kann.

Des Weiteren können mittels einer Übersetzung einige LaTeX-spezifische Eigenheiten umgangen und so automatisiert verarbeitet werden. Bei der Gestaltung von Tabellen zählen hierzu insbesondere die Gestaltung von horizontalen und vertikalen Linien, die Ausrichung von Zellinhalten und die Definition der Kopf- und Fußbereiche (Header und Footer) einer Tabelle.

Die Probleme verschärfen sich, wenn Datensätze (in vorab unbekannter Anzahl) in eine Tabelle eingefügt werden sollen.

Bei der Gestaltung von Tabellen in LaTeX sind so genannte „Multicolumns“, d.h. Zellen, welche sich über mehrere Spalten erstrecken und diese verbinden,

sehr unanschaulich verwaltbar. Hier kommt hinzu, dass man zwar in der normalen `tabular`-Umgebung jeder Spalte eine horizontale Orientierung zuweisen kann (im Tabellenkopf, mittels `begin{tabular}{l|c|r}` beispielsweise eine dreispaltige Tabelle, mit jeweils links-, zentriert-, und rechtsausgerichteten Spalten), es jedoch prinzipiell nicht direkt möglich ist, einer einzelnen Tabellenzelle selbst eine horizontale Orientierung zuzuweisen.

Hierzu muß für jede Zelle eine `multicolumn`-Umgebung definiert werden. Diese Umgebung hat drei Parameter und kann überall dort stehen, wo in LaTeX eine einzelne Zellendefinition stünde. Die Syntax ist

`\multicolumn`{⟨*Anzahl Zellen*⟩}{⟨*Zellendefinition*⟩}{⟨*Inhalt*⟩}

Jeweils die ⟨*Zellendefinition*⟩ bestimmt die horizontale Ausrichtung. Daher wür- de man (manuell) in LaTeX sehr viel Quelltext schreiben, um einzelne Zellen auszurichten. Hier kann der Übersetzer ebenfalls dafür sorgen, dass ein Template zuerst eingelesen wird und beispielsweise die zentrierte Ausrichtung einer gesamten Zeile durch diesen durchgeführt wird.

Ebenfalls ist die Unterstützung für die Angabe von Breiten einzelner Zellen unzulänglich. Hierzu wird in LaTeX normalerweise das `Array`-Paket verwendet, jedoch erlaubt dies nur einen einzigen Parameter in der `multicolumn`-Umgebung. Möchte man eine feste Breite setzen, sowie zentrierte horizontale Ausrichtung, so meldet LaTeX, dass das Array-Paket nur jeweils einen Parameter pro Spaltendefinition zulässt. Dieses Problem kann nur dadurch umgangen werden, dass die Ausrichtung durch zusäzuliche LaTeX-Befehle festgelegt wird.

Alle diese Probleme können dadurch vermieden werden, dass Tabellen durch XHTML beschrieben und durch einen Parser in LaTeX-Code übersetzt werden. Der erstellte LaTeX-Code ist dann zwar nicht mehr kompakt und übersichtlich, jedoch werden alle hier beschriebenen Probleme automatisch durch LateXHTML erkannt und überwunden.

### 2.2.2 Generelle Synatx

Die Syntax der Tags entspricht – wie bereits erwähnt – genau der XHTML-Syntax.

Alle Tags, welche Inhalte umschließen, werden durch eine Anweisung der Form `<tag>` begonnen und durch eine Anweisung `</tag>` abgeschlossen. Weitere Tags, welche kein abschließendes Attribut benötigen, werden als `<singleTag />` geschrieben (der Slash am Ende indiziert den Abschluß des Tags). Allen Tags können beliebig viele Attribute der Form `attribut="wert"`

hinzugefügt werden. Im folgenden werden die Basis-Tags und die dazugehörigen Attribute beschrieben. Zu einigen XHTML-konformen Tags werden neue Attribute hinzugefügt, welche LaTeX-spezifische Funktionalität beschreiben.

In gültigem XHTML müssen alle Tags und Attribute klein geschrieben werden sowie einzelne Attribute grundsätzlich mit einem Wert versehen werden. Ebenso müssen alle Tags gültig abgeschlossen sein. Dies wird jedoch von dem Übersetzer *nicht* überprüft, d.h. es könnten auch Templates in HTML 4 gegeben werden. Dies ist aus Gründen der Rückwärtskompatibilität eingerichtet.

### 2.2.3 Tabellen

Tabellen stellen neben Schriftformatierungen das wichtigste Element dar, um ein Dokument-Layout zu beschreiben und Informationen strukturiert zu präsentieren. Sie sind daher auch das zentrale Konstrukt der Sprache zur Beschreibung der Templates.

Mittels `<table>...</table>` wird eine Tabelle ein- bzw. ausgeleitet. Die Attribute sind in Tabelle 2.2 beschrieben. Durch sie werden Informationen festgelegt, welche für die gesamte Tabelle relevant sind. Die Auswirkungen dieser Attribute werden unter anderem in Abbildung 2.3 verdeutlicht.

| Attribut | Beschreibung |
|---|---|
| `border="n"` | Setzt einen äusseren Rahmen um die Tabelle<br>$n = 0 \Rightarrow$ kein Rahmen, $n > 0 \Rightarrow$ Rahmen mit $n$ pt<br>Alle horizontalen/vertikalen Linien werden<br>ebenfalls in der Dicke $n$ pt gesetzt. |
| `extrarowheight="n"` | Erhöht den Abstand von Zellenanfang und Text<br>(in pt). Standardwert ist 1 |
| `linespace="n"` | jede Zellenhöhe wird um den Faktor $n$ gestreckt.<br>Standartwert ist 1 |

**Tabelle 2.2:** Attribute des `table`-Tags

Mittels `<tr>...</tr>` wird eine Tabellenzeile begrenzt und Attribute festgelegt, welche für jede Zelle dieser Zeile gelten soll.

| Attribut | Beschreibung |
|---|---|
| `align="x"` | Richtet alle Zellen der Zeile links- bzw. rechtsbündig oder zentriert aus ($x \in$ {`left, right, center`}) |
| `hline="x"` | Setzt $x$ horizontale Linien unter die aktuelle Zeile |
| `<hline>` | Alternativ: Dieses Tag steht immer hinter einer Zeilendefinitionen, d.h. im Bereich nach einem ausleitenden `</tr>`-Tags und zeichnet eine horizontale Linie in der gesamten Breite der Spalte. |

**Tabelle 2.4:** Attribute des `tr`-Tags

Die Zellen werden ebenfalls XHTML-konform mit `<td>...</td>` begrenzt. Eine einzelne Zelle kann durch sehr viele Attribute (Tabelle 2.4) näher spezifiziert werden, da diese immer in einer `multicolumn`-Umgebung gesetzt wird.

| Attribut | Beschreibung |
|---|---|
| `align="x"` | Richtet den Inhalt der Zelle der Tabelle links- bzw. rechtsbündig oder zentriert aus. Eine möglicherweise vorhandene Definition im `<tr>`-Tag wird hiermit überschrieben ($x \in$ {`left, right, center`}). |
| `colspan="x"` | Eine Zelle erstreckt sich über $x$ Zellen der aktuellen Zeile (analog zu XHTML) |
| `width="x"` | Die Zeile wird auf eine Breite von $x$ gesetzt. $x$ sollte die eine Einheit beinhalten, beispielsweise $x = 3$cm |
| `hline="x"` | Schreibt $x$ horizontale Linien unter die aktuelle Zeile |
| `rl="x"` | Rechts neben der Zelle wird eine vertikale Linie gesetzt. Ist $x$ eine Zahl, so wird diese über die nächsten $x$ Zeilen fortgesetzt.<br>Ist $x \in$ {`on, off`}, so wird die vertikale Linie beginnend von der Zelle des `on`-Attributes bis zu einer Zelle mit dem `off`-Attribut fortgesetzt. |
| `ll="x"` | Links neben der Zelle wird eine vertikale Linie gesetzt. Das Verhalten ist analog zum `rl`-Tag. |
| `bl="x"` | Unterhalb der Zelle wird eine (sich über $x$ Zellen in der gleichen Zeile erstreckende) horizontale Linie gesetzt. $x$ kann ebenfalls `on` bzw. `off` sein (analog zum `rl`-Tag) |
| `tl="x"` | Oberhalb der Zelle wird (analog zu `bl`) eine horizontale Linie geschrieben. |

**Tabelle 2.6: Attribute des `td`-Tags**

Die `multicolumn`-Umgebung muß gesetzt werden, um sicherzustellen, dass jede Zelle in der Breite angepasst und der Text individuell ausgerichtet werden kann. Die Textausrichtung ist normalerweise horizontal linksbündig, es sei denn, dies wird anders festgelegt. Jede Zelle kann die horizontale Ausrichtung aus der möglicherweise gegebenen Definition einer gesamten Zeile überschreiben.

Das XHTML-Analogon `rowspan` wird Aufgrund fehlender Implementierung in LaTeX nicht unterstüzt, ebensowenig wie die Angabe von relativen Werten (in Prozent) im `width`-Attribut.

Mit diesen Tags lassen sich sehr flexibel auch komplexere Tabellenlayouts gestalten. Insbesondere die Tags mit der Optionen `on`/`off` erlauben später, komplexe mehrseitige Tabellen einfacher zu formatieren. Abb. 2.3 zeigt die Attribute der `<td>`-Tags nocheinmal anschaulich, in der Abbildung unterstrichene Attribute werden innerhalb des `<table>`-Tags gesetzt.

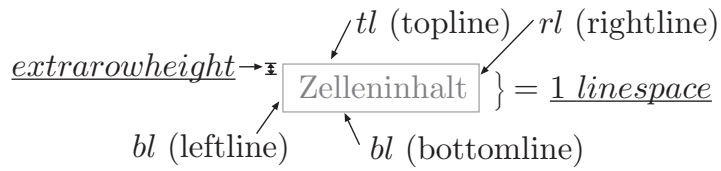



**Abbildung 2.3:** Eine Tabellenzelle mit möglichen Attributen

### 2.2.4 Textformatierungen

Um weitere Textformatierungen analog zu XHTML vornehmen zu können, werden eine Reihe weiterer Tags erkannt und verarbeitet.

| Tag | Beschreibung |
|---|---|
| `<strong>...</strong>` | Der zwischen den Tags stehende Bereich wird mittels `\textbf{...}` **fett** formatiert |
| `<em>...</em>` | Der zwischen den Tags stehende Bereich wird mittels `\textit{...}` *kursiv* formatiert |
| `<hr />` | Zeichnet eine horizontale Linie in der Breite des Textes (horizontal rule). Innerhalb einer Tabelle kann dieser Befehl nicht verwendet werden |
| `<hfill />` | Analog zu dem LaTeX-Befehl `\hfill` |
| `<newpage />` | Weist LaTeX an, die Seite umzubrechen |
| `<today />` | Gibt das aktuelle Datum aus |

**Tabelle 2.8:** Weitere Tags

Ebenfalls ist eine Änderung der Schriftgröße möglich. Dies bewerkstelligt der `<font>`- Tag. Der Aufruf erfolgt mit `<font size="`$x$`">`. Die entsprechend umgesetzten Schriftgrößen abhängig von Parameter $x$ findet man in Tabelle 2.9. Hierbei sei angemerkt, dass ein Abschließen eines solchen Tags durch ein `</font>`-Tag zwar XHTML-konform ist, jedoch durch den Übersetzter die Schriftgröße dann auf die normale Größe zurückgesetzt wird. Dies kann bei der Verwendung von kleinen Schriften unter Umständen zu größeren Zeilenabständen führen. Ein korrekter Abschluß des Tags ist also nicht unbedingt erforderlich.

| $x$ | LaTeX-Entsprechung |
|---|---|
| $-2$ | `\tiny` |
| $-1$ | `\scriptsize` |
| 0 | `\footnotesize` |
| 1 | `\small` |
| 2 | `\normalsize` (Standardgröße) |
| 3 | `\large` |
| 4 | `\Large` |
| 5 | `\LARGE` |
| 6 | `\huge` |
| 7 | `\Huge` |

**Tabelle 2.10:** Schriftgrößen

### 2.2.5 Grafiken

Grafiken lassen sich über das `<img>`-Tag einbinden. Die Attribute dieses Tags sind in Tabelle 2.12 dargestellt. Hierbei sind die Grafiken im Pfad zu hinterlegen, in welchem man das Template speichert. Die Bilder sollten – falls es Vektorgrafiken darstellen – sowohl im EPS- als auch im PDF-Format vorliegen und die Endung im `src`-Attribut *nicht* angegeben werden. Im Falle von anderen Grafikformaten konvertiert man diese am Besten nach PDF und/oder EPS. Die Grafiken sollten im Templateordner oder in einem Subordner abgelegt werden.

| Attribut | Beschreibung |
|---|---|
| `src="..."` | Pfad zur Bilddatei |
| `width="x"` | $x$ = Breite der Bilddarstellung |
| `height="y"` | $y$ = Höhe der Bilddarstellung |

**Tabelle 2.12:** Attribute des `img`-Tags

### 2.2.6 Weitere Befehle und Ergänzungen

Leicht lassen sich weitere XHTML- bzw. LaTeX-Befehle einbetten, um so die Funktionalität schrittweise zu verfeinern. Dies ist jedoch nicht Gegenstand dieser Studienarbeit. Prinzipiell könnte dieses Grundgerüst auch dazu dienen, einen HTML-to-LaTeX-Übersetzer zu implementieren; Tabellen lassen sich bereits beliebig schachteln[1] – somit ist prinzipiell die Möglichkeit gegeben, jede Webseite in LaTeX bzw. in PDF zu konvertieren. Im weiteren Verlauf wird auf mögliche Erweiterungen des Systems eingegangen.

Des Weiteren lassen sich in den Templates bereits LaTeX-Befehle einbetten. Diese werden von dem XHTML-Übersetzer ignoriert und somit in das Zieldokument übernommen. Lediglich nicht erkannte XHTML-Tags werden ignoriert, d.h. durch den Übersetzer entfernt.

### 2.2.7 Beispiel

Im folgenden Beispiel werden die bereits besprochenen Tags und Attribute in XHTML implementiert. In Abb. 2.4 wird die WYSIWYG-Ansicht in Dreamweaver dargestellt, in Abb. 2.6 der Quelltext und in Abb. 2.5 das durch den Übersetzer und PDFLaTeX erzeugte Resultat dargestellt.

In dem Codebeispiel in Abbildung 2.6 sind bereits einige spezielle, in { und } eingeschlossene Schlüsselwörter eingefügt; diese werden später durch das Templatesystem mit (möglicherweise datenbankgestützten) Inhalten befüllt. Da diese Inhalte hier noch nicht eingefügt werden sollten, wurden die Steuerfelder durch den Übersetzer entfernt.

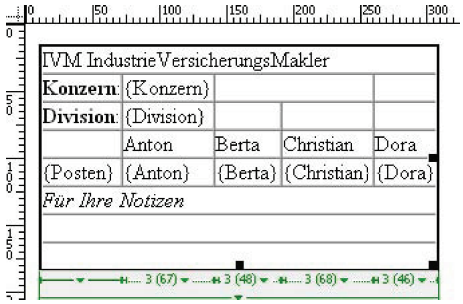



**Abbildung 2.4:** Template in WYSIWYG-Editor „Dreameweaver“

[1] Dies gilt nicht, wenn man die `longtable`-Umgebung verwendet

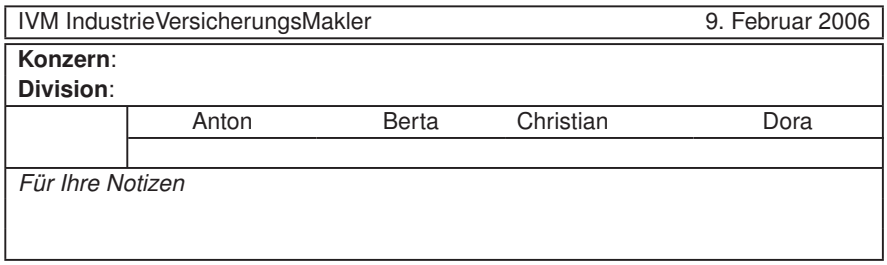



**Abbildung 2.5:** Durch den Übersetzer erzeugtes PDF

```
<table border="1" cellpadding="0" cellspacing="0" >
  <tr>
    <td colspan="5">IVM IndustrieVersicherungsMakler<hfill /><today/></td>
  </tr>
  <hline />
  <hline />
  <tr>
    <td colspan="1"><strong>Konzern</strong>:</td>
    <td>{Konzern}</td>
    <td colspan="2"></td>
    <td> </td>
  </tr>
  <tr height="15">
    <td><strong>Division</strong>:</td>
    <td>{Division}</td>
    <td></td>
    <td></td>
    <td></td>
  </tr>
  <hline />
  <tr align="center">
    <td rl="on"></td>
    <td align="center" bl="on" width="3cm">Anton</td>
    <td width="3cm">Berta</td>
    <td align="right" width="3cm">Christian</td>
    <td width="3cm">Dora</td>
  </tr>
  <tr>
    <td>{Posten}</td>
    <td>{Anton}</td>
    <td>{Berta}</td>
    <td>{Christian}</td>
    <td>{Dora}</td>
  </tr>
  <hline />
  <tr>
    <td colspan="5" rl="off"><em>F&uuml;r Ihre Notizen</em></td>
  </tr>
  <tr>
    <td colspan="5"></td>
  </tr>
  <tr>
    <td colspan="5"></td>
  </tr>
</table>
```

**Abbildung 2.6:** Das Template-Beispiel

## 2.3 Die Template-Engine

Im folgenden Abschnitt wird die Template-Engine eingeführt, welche die Beschreibungssprache um zusätzliche Markup-Tags erweitert und für die korrekte Einbettung von externen Informationen in die eigentlichen Templates zuständig ist. Die Syntax dieser Markup-Tags ist von dem der XHTML-ähnlichen Beschreibungssprache verschieden, da durch sie die Platzhalter für Inhalte und Inhaltsstrukturen beschrieben werden.

### 2.3.1 Funktionsweise

Die Template-Engine repräsentiert die Schnittstelle zwischen den eigentlichen Templates und den in den Templates darzustellenden Daten. Dieses System erkennt spezielle Tags, ersetzt diese durch Inhalte, welche aus Datenbanken oder anderen Quellen in das Dokument eingefügt werden. Gewissermaßen verarbeitet es Platzhalter, indem es diese durch Daten ersetzt. Ebenfalls lässt sich hiermit ein Dokument syntaktisch strukturieren, d.h. durch bestimmte spezielle Tags lassen sich Bereiche des Templates von anderen abgrenzen (sich wiederholende Platzhalter).

Wie bereits angedeutet sind alle Tags, welche durch das Template-System interpretiert werden, in geschweiften Klammern „{“ und „}“ gesetzt. Dies hat den Grund, dass diese Syntax sehr häufig von CMS-Distributionen verwendet wird. Ein weiterer Grund ist, dass sich ein Template, welches möglicherweise für eine Webseite bereits zur Verfügung steht, mittels minimaler Änderungen auch durch LateXHTML verarbeiten lässt.

Das Template-System liest also zunächst das gesamte XHTML-Template und durchsucht es mittels regulärer Ausdrücke auf Ausdrücke der Form

$$\{Tag\ Attribut_1 = "Wert_1"\ \ldots\ Attribut_n = "Wert_n"\}$$

LateXHTML verarbeitet das XHTML-Dokument, indem es alle Platzhalter durch in einem geeigneten Array gegebene Daten ersetzt. In diesem Array werden für das gesamte Dokument globale Platzhalter (als Strings) sowie für einzelne, wiederholbare Bereiche durch ebenfalls als Array gegebene Mengen von Datensätzen definiert. Dieses Array wird im Folgenden `$data`-Array genannt.

### 2.3.2 Daten-Tags

Zunächst bezeichnet ein Tag der Form `{"Bezeichner"}` einen Container für Daten der Variable `Bezeichner`. Hierbei kann man die Anführungszeichen

weglassen, wenn das Tag eindeutig bleibt. Diese einfachen Daten-Tags definieren Platzhalter, welche über das gesamte Dokument genau einen Wert annehmen können. Wird ein Tag `{Firma}` in das Template gesetzt, so wird der entsprechende Inhalt (`$data["Firma"]`) durch die Template-Engine ersetzt. Alle auf der ersten Ebene des `$data`-Arrays gesetzten Informationen können somit global für das Dokument festgelegt werden.

```
$data["Konzern"] = "Anton␣Zeilinger␣GmbH";
$data["Division"] = "Buchhaltung";
$data["Sachbearbeiter"] = "Herr␣Dr.␣Maier";
$data["posten"] = array(
    array("Anton" => "Anton-Wert1",
          "Berta" => "Berta-Wert1",
          "Christian" => "Christian-Wert1",
          "Dora" => "Dora-Wert1"),
    array("Anton" => "Anton-Wert2",
          "Berta" => "Berta-Wert2",
          "Christian" => "Christian-Wert2",
          "Dora" => "Dora-Wert2")
);
```

**Abbildung 2.7:** Daten zur Übergabe an die Template-Engine

Der Eintrag `$data["posten"]` enthält ein Array als Eintrag, welches ebenfalls wieder Arrays (Paare von Schlüssel $\mapsto$ Wert) enthält. Die Daten der Variable `$data["posten"]` werden durch spezielle Tags verarbeitet, welche im folgenden Abschnitt besprochen werden.

### 2.3.3 Steuertags

Da in den Tabellen beliebige Listen von Daten aufgenommen werden können, müssen Bereiche definiert werden, in welchen die entsprechenden Inhalte eingefügt werden. Bei Listen gleich strukturierter Datensätze bietet sich an, einen Teil der Tabelle für die Aufnahme von mehreren Tupeln zu definieren. Die Struktur eines einzelnen Tabelleneintrags in Bezug auf die Zellenaufteilung ändert sich dabei nicht. Es werden lediglich die Inhalte durch jedes Tupel neu eingefügt und die Tabelle fortgesetzt.

Hierzu dient das `loop`-Tag. Es wird mit dem Parameter `name` gesetzt und muß ebenfalls wieder abgeschlossen werden. Der zwischen dem einleitenden und ausleitenden Tag stehende Bereich der Tabelle wird für jeden Datensatz wiederholt eingefügt. In Bezug auf den im `$data`-Array definierten Wert für `posten` würde ein solcher `loop`-Bereich wie folgt aussehen:

```
 {loop name="posten"}
2     <tr hline="1">
          <td></td>
          <td>{Anton}</td>
          <td>{Berta}</td>
          <td>{Christian}</td>
7         <td>{Dora}</td>
      </tr>
 {/loop}
```

**Abbildung 2.8:** `loop`-Bereich für `posten`

Würde man dieses Codefragment in den Code aus Abb. 2.6 anstelle der Zeile 28-34 einfügen und den Übersetzer dann mit den Daten aus Abb. 2.7 starten, so entstünde die in Abb. 2.9 zu sehende Ausgabe.

| IVM IndustrieVersicherungsMakler | | | | 10. Februar 2006 |
|---|---|---|---|---|
| **Konzern**: Anton Zeilinger GmbH<br>**Division**: Buchhaltung | | | Ihr Zeichen: _____ | |
| | Anton | Berta | Christian | Dora |
| | Anton-Wert1 | Berta-Wert1 | Christian-Wert1 | Dora-Wert1 |
| | Anton-Wert2 | Berta-Wert2 | Christian-Wert2 | Dora-Wert2 |
| *Für Ihre Notizen* | | | | |

**Abbildung 2.9:** PDFLatex-Kompilat mit befülltem `loop`-Bereich

Hier kann die Tabelle jedoch immer noch über das Seitenende hinauslaufen, wenn die Anzahl der Datensätze in dem `loop`-Bereich zu groß wird. Wie der Auslauf kontrolliert und unterbunden wird, ist Gegenstand des folgenden Kapitels. Um die Tabellen korrekt umzubrechen und die entsprechenden Header und Footer auf jeder Seite zu setzen, bietet sich an, die Kopf- und Fußzeilen von Tabellen ebenfalls durch das Template-System markieren zu lassen.

Um derartige Kopf- und Fußzeilen definieren zu können, benötigt man weitere Tags, die hier `section`-Tags genannt werden. Diese definieren in Verbindung mit gleichnamigen `loop`-Tags die Kopf- und Fußbereiche. Hierbei ist alles, was zwischen einleitendem `section`-Tag und einleitendem `loop`-Tag gesetzt ist, der Tabellenkopf und wiederrum alles, was zwischen den ausleitenden Tags gesetzt wird, der Tabellenfuß. Abbildung 2.10 verdeutlicht dies.

```
{section name="tabelle"}
<table>                                          } Tabellenkopf
   <tr><td>Spalten</td></tr>                     } (Header)
  {loop name="tabelle"}
   <tr><td>{Spalteninhalt}</td></tr>             } Tabelleninhalt
   {/loop}                                       } (Body)
   <tr><td>Copyright (c)..</td></tr>             } Tabellenfuß
</table>                                         } (Footer)
{/section}
```

**Abbildung 2.10:** Bereiche einer Tabelle

## 2.4 Das LaTeX-Layout

Um die durch die Übersetzung generierten LaTeX-Tabellen in eine Seite zu setzen, muß noch ein Grundlayout definiert werden, in welches der Dokumentkörper eingebettet wird. Dort werden Pakete definiert und das Seitenlayout festgelegt.

```
1 \documentclass[12pt,twoside]{article}
  \begin{document}
  %{container}%
  \end{document}
```

**Abbildung 2.11:** Beispiellayout mit Container

Ein solches Layout muß zusätzlich einen Platzhalter („Container“) für den in ihm zu setzenden Inhalt definieren. Dies geschieht durch einen Kommentar innerhalb dieses Layouts. In Abbildung 2.11 wurde ein solcher Platzhalter in Zeile 3 eingefügt (dieser lautet immer `%{container}%`).

Zur korrekten Formatierung sind insbesondere einige Zusatzpakete im Kopf der Layout-Datei einzubinden. Diese unterteilen sich in die folgenden:

- **Unterstützung für die deutsche Sprache.** Um korrekte Silbentrennung und komfortabelere Einbindung von Umlauten zu erreichen, sollten in den Dokumentkopf die folgenden Anweisungen eingebunden werden:
  `\usepackage{german}`, `\usepackage[T1]{fontenc}`,
  `\usepackage[latin1]{inputenc}`, `\usepackage{ngerman}`.

- **Erweiterte Tabellenumgebungen.** Um die durch LateXHTML implementierten Funktionen vollständig nutzen zu können, muß zusätzlich das `Array`-Paket eingebunden werden. Das Paket `Array` erweitert die Umgebungen `array` und `tabular` um weitere Deklarationen, wie unter anderem das Setzen von Breitenangaben in der Tabellenpräambel. Um es einzubinden setze man die Anweisung `\usepackage{array}`

in den Dokumentkopf. Ebenfalls sollte das Paket `longtable` eingebunden werden, da es durch LateXHTML ebenfalls als Tabellenumgebung unterstützt wird.

- **Weitere Schriftarten.** Um die LaTeX-Standardschriftarten durch andere Standardschriften zu ersetzen, eignen sich die Pakete `helvetica` und `times`. Helvetica ist eine serifenlose, Arial-ähnliche Schrift, Times hingegen ist die bekannte Times New Roman-Schriftart. Diese Schriftarten bindet man ebenfalls über einen `\usepackage{...}`-Befehl ein.

- **Grafik-Unterstützung.** Um EPS- und PDF-Grafiken in die Dokumente einzubetten, verwendet man am besten das Paket `graphicx`.

Möchte man das entsprechende Papierformat im Querformat, so genügt es, den Parameter `landscape` in der `documentclass`-Anweisung einzubinden. Für die genauere Gestaltung des Seitenlayouts bieten sich – ebenfalls im Dokumentkopf – `\setlength{...}`-Anweisungen an, deren Erläuterungen hier den Rahmen sprengen würde. Für eine sehr gute Beschreibung siehe [3], Seite 20ff.

## 2.5 Beispiele auf CD

Auf der dieser Arbeit beiliegenden CD finden sich mehrere Beispiele, in denen alle in dieser Arbeit beschriebenen Funktionen angewendet werden.

- Im Verzeichnis `templates/` befinden sich drei Layouts, `a4hoch.tex`, `a4quer.tex` und `a4querhelv.tex`. Sie verwenden jeweils das DIN-A4-Format, die Schriftpakete `times`, `helvetica` und Computer Modern. Alle besitzen jeweils ein Container-Tag.

- Die Dateien `pruefung.php` und `templates/pruefung.htm` geben ein Beispiel, wie LateXHTML für eine Leistungsübersicht von Prüfungsnoten (ähnlich wie im Studierendenportal der Universität Freiburg) verwendet werden kann.

- `simple_demo.php` und `templates/simple_demo.htm` demonstrieren eine über mehrere Seiten laufende, große Tabelle.

- `tabelle.php` und `templates/tabelle.htm` geben ein Beispiel für die geschachtelte Verwendung von Tabellen.

# Kapitel 3

# Über das Seitenende auslaufende Tabellen

Im vorigen Kapitel wurde erörtert, wie man ähnlich wie mit XHTML Templates beschreiben kann, ihnen zusätzliche Markups für die LaTeX-Umgebung hinzufügt und diese dann mit Daten füllt. Das eigentliche Problem, nämlich das korrekte Auffüllen der Datenbereiche dahingehend, dass sich die Tabelle über mehrere Seiten erstrecken kann, wurde noch nicht besprochen. Hierzu werden drei Lösungsansätze vorgestellt, von denen im folgenden Verlauf die implementierte Lösung näher erörtert werden sollen.

## 3.1 Lösungsmöglichkeiten

- **Feste Zeilenhöhe.** Eine erste Möglichkeit, lange Tabellen korrekt umzubrechen, könnte durch das Festsetzen der Schriftgröße auf einen festen Wert erreicht werden. Die Template-Engine könnte dann Zeilenumbrüche und Tabellenzeilen zählen und nach einer bestimmten (ebenfalls festzusetzenden oder zu errechnenden) Anzahl von Umbrüchen die aktuelle Tabelle abschließen, eine `\newpage`-Anweisung setzen, den Tabellenheader erneut auszugeben und mit der restlichen Anzahl von Datensätzen fortzufahren (und genanntes Verfahren gegebenfalls zu wiederholen).

  Diese Lösung ist jedoch keinesfalls robust, da mit jedem neuen LaTeX-Layout (jeweils abhängig von Hoch-/Querformat, Dokumentklasse etc.) diese Parameter manuell festgesetzt werden müssten. Daher schied dieses Verfahren bei den Überlegungen bereits von Anfang an aus und wurde nicht näher betrachtet.

- **Verwenden des `Longtable`-Pakets.** Um Tabellen über mehrere Seiten korrekt direkt durch Latex setzen zu lassen, kann man das Paket „`longtable`“verwenden. Diese Umgebung ermöglicht es, dass (unter anderem) Tabellenkopf und -fuß direkt in LaTeX definiert werden können und beim Kompilieren einer über das Seitenende auslaufende Tabellen automatisch umgebrochen und auf der nächsten Seite (mit korrektem Kopf und Fuß) fortgesetzt werden.

  Diese Lösung ist jedoch als unsauber anzusehen, da wir für jede Zelle `\multicolumn`-Umgebungen verwenden und das Dokument bei Verwendung dieses Pakets teilweise bis zu fünfmal kompiliert werden müsste, da die Zellenbreiten für jede sich über mehrere Zellen erstreckende `multicolumn`-Umgebung neu berechnet werden müssen. Hierzu findet sich in der französischen Dokumentation ein Beispiel[1]. Da für den folgenden Lösungsvorschlag das Dokument durchschnittlich nur dreimal compiliert werden muß, um die korrekten Tabellenhöhen zu bestimmen, wurde diesem Vorzug gegeben.

  Allerdings wird die `longtable`-Umgebung dennoch unterstützt. Im folgenden Kapitel genauer beschrieben, wie diese verwendet werden kann.

- **Einbeziehen der Informationen des Latex-Compilers.** Eine Beobachtung beim Compilieren von Latex-Dokumenten ist, dass Latex bei zu großen Tabellen Fehlermeldungen der Art

  ```
  Overfull \vbox (16.99936pt too high) has occurred while
     \output is active
  ```

  ausgibt. Die so erhaltene Information kann nun ausgenutzt werden, um die Tabelleninhalte auf mehrere Seiten korrekt aufzuteilen. Die Implementierung des Systems nutzt diese Möglichkeit und soll im folgenden Abschnitt näher vorgestellt werden.

## 3.2 Algorithmus zur korrekten Bestimmung der Tabelleninhalte

Compiliert man TEX-Dokumente serverseitig, indem man den PHP-Befehl `exec` aufruft, kann die Ausgabe, welche normalerweise in der Konsole dargestellt würde, in ein Array eingelesen und zeilenweise ausgewertet werden. Somit genügt es, alle Zeilen des Ausgabearrays zu durchlaufen und zu prüfen, ob „overfull vertical boxes“ vermeldet werden und mittels eines regulären

[1] `ftp://dante.ctan.org/tex-archive/info/french-translations/macros/latex/required/tools/f-longtable.dvi`

Ausdrucks den Wert (in pt) zu extrahieren, für welchen LaTeX das Dokument als „zu lang“ bewertet. Durch diesen Wert kann eine Abschätzung gewonnen werden, nach wievielen Tupeln eine Tabelle in einem `loop`-Bereich umgebrochen werden muß.

Da ein Dokument mehrere, unabhängige Tabellen mit `loop`- und `section`-Bereichen enthalten kann, sind zusätzlich zwei Dinge zu beachten:

- Per se ist nicht klar, zu welchem `section`-Tag LaTeX die „overfull boxes“-Warnung generiert.
- Eine Tabelle (`section`) sollte immer am Anfang einer neuen, leeren Seite beginnen. Andernfalls würde bei einem Auslauf der Tabelle ein Wert errechnet, der die Tabelle auf den Folgeseiten „zu früh“ umbrechen lassen würde.

Das erste Problem kann umgangen werden, indem das System bei jeder einleitenden `section` eine LaTeX-Warnung ausgibt (dies geschieht mit dem LaTeX-Befehl `\PackageWarning`) und diese selbst wieder ausließt. Da der Name der `section` ebenfalls ausgegeben wird, muß einer etwaigen Warnung von „overfull boxes“ eine interne Warnung vorausgegangen sein. Somit ist zu jeder Tabelle bekannt, inwiefern diese eventuell über das Seitenende hinausläuft.

Das zweite Problem kann derzeit nur über eine `longtable`-Umgebung gelöst werden. Hierzu finden sich weitere Anmerkungen im Fazit.

Der implementierte Algorithmus ist in Abbildung 3.1 dargestellt.

$limits[s] = \infty \; \forall$ sections $s$ *// überschätze limits*
**do** {
  $html$ = **parse_template**($limits$);
    *// Daten in das Template packen*
  $latex$ = **parse_html**($html$);
    *// Template in Latex übersetzen*
  $output$ = **pdflatex**($latex$);
    *// Array zum Abfangen der Ausgabe*
  $limits$ = **recalculate_limits**($output$);
    *// berechne neue Section-Limits*
} **until** (recalculate_limits entdeckt in *output* keine Fehler)

**Abbildung 3.1:** Die Hauptschleife

Vor der Hauptschleife werden alle *limits*-Werte überschätzt, d.h. die Tabelle wird mit allen vorhandenen Daten in LaTeX übersetzt und nicht umgebrochen. Solange LaTeX beim Compilieren Fehler meldet, werden die *limits*-Werte neu berechnet und das Dokument erneut compiliert. Für jede `section` bezeichnet *limits* die maximale Anzahl von Tupeln, welche auf einer Seite der Tabelle Platz finden.

Tauchen also „overfull vertical boxes“ auf, so wird *limits* (ausgehend von der maximalen Anzahl der Datensätze) verringert. Ein Umbruch der Tabelle erfolgt immer nach *limits* Tupeln, ist die Tabelle zu „voll“, wird *limits* folgendermaßen neu berechnet:

$$limits[section] = \left\lceil limits[section] - \frac{val}{17} \right\rceil$$

Hierbei bezeichnet *val* den Wert (in der Einheit pt), welcher durch LaTeX in der Warnung ausgegeben wurde. Da ein Dokument normalerweise in eine (normalen) Schriftgröße von 12 Punkt gesetzt wird und in Tabellen in der Regel durch Trennzeilen und zusätzlichen Raum in den Zellen mehr Platz für eine Zeile benötigt wird, so ist ein Wert von 17 pt pro Zeile realistisch.

# Kapitel 4

# Technische Anmerkungen

In diesem Kapitel finden sich Anmerkungen zur Lösung des Problems auslaufender Tabellen sowie zu verwendeten Techniken und Algorithmen. Ebenso wird ein Überblick gegeben, wie das konkrete System aufgebaut ist und wie es sich in bestehende Anwendungen einbetten lässt. Ebenfalls wird ein Überblick gegeben, wie man eine für LateXHTML reduzierte LaTeX-Installation erstellt und installiert.

## 4.1 Aufbau der Quelltexte

Der gesamte Quelltext ist in der Bibliothek-Datei `lateXHTML.lib.php`[1] abgelegt und dort in Klassen und Funktionen aufgeteilt. Die Aufteilung soll hier kurz skizziert werden.

In der Bibliothek befinden sich zwei Klassen: `LaTeXHTML` und `fileSystemObj`. Auf die zweite Klasse wird am Ende des Abschnittes kurz eingegangen; diese Klasse implementiert lediglich die grundlegenden Dateisystemoperationen und wird nicht näher erläutert. Sie ist usprünglich als Teil eines anderen Projektes entstanden; wegen der übersichtlichen und einfachen Handhabe verwende ich sie jedoch immer wieder.

[1] Im Folgenden nur noch „Bibliothek“ genannt

## Instanzvariablen

Die `Klasse LateXHTML` benutzt die folgenden Instanzvariablen:

- `limits` Array, welches für jedes `section`-Tag speichert, wie viele Datensätze auf jeweils einer Seite dargestellt werden können.
- `env` String, welcher die Tabellenumgebung in LaTeX enthält. Dieser kann `tabular`, `tabularx` oder `longtable` sein, normalerweise wird `tabular` verwendet.
- `debug` Boolescher Wert, wenn dieser auf `true` gesetzt ist, so werden Debugmeldungen direkt ausgegeben. Dieser ist per default `false`.
- `debugLog` Array, welches die Ausgaben der Klasse und der Aufrufe von PDFLaTeX zeilenweise enthält. Dieses ist für eventuelle Fehlerbehebung nützlich.
- `data` Assoziatives Array, welches alle in den Templates durch Platzhalter gesetzten Daten entählt.
- `latexpath` String, welcher den LaTeX-Pfad enthalten kann. Ist dieser beispielsweise bei Windows-Umgebungen in der Umgebungsvariable `PATH` gesetzt, so kann dieser String leer bleiben. Dies ist auch standartmäßig so eingestellt.
- `type` Ausgabetyp: 1 = PDF, 2 = DVI. Normalerweise auf 1 eingestellt.
- `pdflatexcall` und `latexcall` Befehle, die aufgerufen werden, wenn eine PDF- oder DVI-Datei erstellt werden soll. Normalerweise sind diese Strings `pdflatex` bzw. `latex`.

## Setzen von Einstellungen

Die Methoden zum Setzen von Einstellungen sind folgende:

- `useTemplate($file)` Legt das durch die aktuelle Instanz zu benutzende XHTML-Template fest.
- `useLayout($file)` Legt das durch die aktuelle Instanz zu benutzende LaTeX-Layout fest.
- `setData($data)` Übergibt ein assoziatives `data`-Array, in welchem die Daten für das Template in der Form $Platzhalter \mapsto Inhalt$ gespeichert sind.

- `setEnv($env)` setzt die Tabellenumgebung auf `$env`. So kann man beispielsweise mit `setEnv("longtable")` die `longtable`-Umgebung einschalten.
- `setDebug($bool)` Setzt die `debug`-Instanzvariable auf `true` oder `false`.
- `debugAppend($string)` intern genutzte Funktion zum Anhängen einer einzelnen Zeile an das `debugLog`-Array. Wenn die `debug`-Variable `true` ist, so wird die Meldung `string` nicht nur an das Array angehängt, sondern zusätzlich direkt ausgegeben.

## Hauptmethoden

- `writeOut($contents)` Schreibt `$contents` in die Zieldatei
- `parseTpl($parsecontents,$data)` Diese Funktion überführt den im Parameter `parsecontents` gegebenen XHTML-Inhalt in XHTML-Code mit eingesetzten Daten aus dem `data`-Parameter. Hier werden alle `loop`- und `section`-Bereiche sowie andere Platzhalter verarbeitet und sehr lange Tabellen auf mehrere Seiten aufgeteilt. Rückgabewert ist das „befüllte“ Template.
- `parseHtml($parsecontent)` Diese Funktion übersetzt den im Parameter `parsehtml` gegebenen XHTML-Quelltext in LaTeX-Code. Dieser wird durch die Funktion zurückgegeben.
- `buildFile($file,$typ)` Diese Funktion wird aufgerufen, wenn alle zur Generierung benötigten Informationen bekannt sind. In dieser Funktion ist der Algorithmus aus dem vorherigen Kapitel implementiert. Diese Funktion erzeugt die im Parameter `file` angegebene TEX-Datei und baut aus ihr das zu generierende Dokument auf (im durch `type` bestimmten PDF- oder DVI-Format). Ist der Parameter nicht gesetzt, so wird auf die Instanzvariable `type` zurückgegriffen und der Typ des Dokuments so bestimmt. Im Erfolgsfall (das Dokument `$file` wurde korrekt erzeugt) wird `true` zurückgegeben, andernfalls `false`.

## Die Klasse `fileSystemObj`

Die Klasse `fileSystemObj` ist die zweite, in der Bibliothek angegebene Klasse. Sie implementiert elementare Dateisystem-Operationen wie das Öffnen, Lesen, Schreiben und Speichern von Dateien. Da die Dateisystemfunktionen in PHP nicht objektbasiert implementiert sind, bot es sich an, diese dahingehend zu implementieren und immer wieder benötigte Operationen (wie beispielsweise das Auslesen der gesamten Datei) zu vereinfachen. So wird ein neues Dateisystemobjekt durch den Konstruktoraufruf

```
$fso = new fileSystemObj("/temp/temp.txt");
```

im Speicher erzeugt. Auf dieses können dann die einzelnen Methoden angewendet werden. Man beachte hier, dass der Pfad absolut angegeben wird. Daher muß noch ein Stammverzeichnis („root path“) angegeben werden. Im folgenden Abschnitt wird beschrieben, wie dieser angepasst wird.

Die Methoden dieser Klasse sind unter anderem:

- `exists()` prüft, ob die Datei existiert
- `open()` öffnet die Datei und liest den Inhalt aus
- `create()` erstellt die Datei, insofern sie noch nicht existiert
- `delete()` löscht die Datei
- `save()` speichert die Datei

## 4.2 Einrichtung eines minimalen LaTeX-Systems

In diesem Abschnitt soll eine minimale Installation einer LaTeX-Distribution auf einem Computer eingerichtet werden. Hierbei beschränke ich mich auf MiKTeX, da die Installation der TeX-Dateien auf anderen Betriebssystemen analog verläuft.

Die Struktur einer TeX-Installation ist unabhängig von Distribution und Betriebssystem immer in zwei Verzeichnisse eingeteilt: `texmf` und `localtexmf`. Im `texmf`-Baum finden sich alle benötigten TeX-Pakete und Binaries sowie Schriften und Konfigurationsdateien; Im Verzeichnis `localtexmf` hingegen werden alle veränderlichen Daten wie Zeichensätze, die online erzeugt wurden oder eigene Erweiterungen und Pakete gespeichert.

Da ein System möglichst „schlank“ sein sollte, kann man nun überlegen, welche Komponenten einer TeX-Installation benötigt werden, um das in dieser Studienarbeit vorgestelle LateXHTML in vollen Funktionsumfang zu unterstützen. Hierzu gibt es prinzipiell mehrere Möglichkeiten. Die von mir ausprobierte Methode möchte ich hier vorstellen. Als Vorraussetzung sollte die MikTeX-Basisinstallation[2] heruntergeladen werden.

Man gehe wie folgt vor:

[2] `http://prdownloads.sourceforge.net/miktex/basic-miktex-2.4.2207.exe`, 32 MB

(1) **Automatische Installation durch das Setup.** Der erste und möglicherweise bequemste Weg ist es, die heruntergeladene Setup-Datei auszuführen und MikTEX so auf dem System zu installieren. Es enthält bereits viele Pakete, die für LateXHTML nicht notwendigerweise erforderlich sind. Diese können nun mit dem MikTEX-Package-Manager deinstalliert werden. Diese sind unter anderem

- `amslatex` und `amsfonts` mathematische Symbole
- `latex2e-help-texinfo` LaTeX-Dokumentation
- `miktex-bibtex-*`[3] BibTeX (Dokumentenzitate)
- `miktex-cjkutils-bin` Schriftkonvertierung
- `miktex-cweb-bin`
- `miktex-dvi*` Utilities, um DVI-Dateien zu manipulieren
- `miktex-eomega-bin`
- `miktex-ghostscript-*` GhostScript-Utilities
- `miktex-gsf2pk-*` Font-Konvertierungsutilities
- `miktex-mktex-*` Font-Konvertierungsprogramme
- `miktex-omega*` Omega-Package
- `miktex-psutils*` PostScript-Utilities
- `miktex-tex4ht-*` TeX-2-HTML-Konverter
- `miktex-texify-*`
- `miktex-texinfo*` TeXInfo Binaries
- `miktex-texware-*` TexWare für DVI-Dateien
- `miktex-web-bin` Tangle, Weave und Tie
- `tds` - Dokumentation

Jedoch kann man die Basis-Setupdatei auch mit einem Packprogramm wie WinRAR öffnen, und die (oben geannten) nicht benötigten Pakete aus dem Archiv entfernen. Dann kann man das reduzierte Archiv manuell entpacken. Ein `texmf`-Verzeichnis wird hierbei erstellt. Wurde dieser Weg gewählt (also das Setup-Programm nicht ausgeführt), so sollte man nun auf der Kommandozeile in das Verzeichnis `miktex/bin/` wechseln und den Befehl `texhash` eingeben. So wird die Package-Datenbank aktualisiert. Um TEX von allen Verzeichnissen aufrufen zu können, sollte das `bin`-Verzeichnis der Windows-Umgebungsvariable `PATH` hinzugefügt werden.

[3] der Stern (*) ist hier als „wildcart“ anzusehen

(2) **Manuelles Minimieren.** Hier wurde ein wenig „herumprobiert“, und im Verzeichnis `texmf` einige Ordner entfernt, welche ich vom Ordnernamen und Inhalt als nicht relevant eingeschätzt habe. Dies waren die Ordner `etex`, `script` und `web2c`. Des Weiteren habe ich im Ordner `bin/` alle Dateien bis auf die vorhandenen DLL-Dateien sowie `pdflatex.exe`, `etex.exe`, `inimf.exe`, `iniexmf.exe` und `texhash.exe` gelöscht. Ausgehend davon habe ich die Package-Datenbank aktualisiert und sämtliche auf der CD befindlichen Beispiele sowie die Studienarbeit selbst mit dem minimierten MikTeX-System kompiliert. Es wurden automatisch einige Pakete nachinstalliert, und im `bin`-Ordner kamen einige EXE-Dateien hinzu. Danach kompilierten alle Dokumente fehlerfrei.

Die minimale Installation befindet sich auf der CD im Order `miktex-min`. Hierbei genügt es ebenfalls, diese in ein Verzeichnis zu entpacken und im Ordner `texmf/miktex/bin/` den Befehl `texhash` auszuführen, um diese auf einem Computer „lauffähig“ zu machen.

## 4.3 Installation des Systems

Um LateXHTML in bereits vorhandene Applikationen einzubetten, ist relativ wenig Aufwand erforderlich. Die einzelnen Installationsschritte werden hier angegeben.

(1) **Vorbereiten der Dateien.** Zunächst muß in der Bibliothek eine Pfadanpassung vorgenommen werden, da der Root-Pfad korrekt gesetzt werden muß. Der Root-Pfad bezeichnet das auf höchster Ebene liegende Verzeichnis, auf welches die Klasse `fileSystemObj` Zugriff hat.

Dies geschieht in der Bibliothek in Zeile 3, durch eine Anweisung

```
define(ROOT_PATH, "Pfad");
```

wobei dieser Pfad beispielsweise wie folgt sein kann:

- `C:/inetpub/wwwroot/` für IIS-Windows-Systeme
- `/home/dbis/` für das DBIS-Stammverzeichnis

Am Ende des Pfades sollte ein Slash (`/`) stehen.

(2) **Einbeziehen der Bibliothek.** In jedem PHP-Dokument, in welcher PDF-Dateien automatisch erzeugt werden sollen, muß die Bibliothek eingebunden werden. Dies erfogt in der Regel über eine Anweisung der Form

```
include("lateXHTML.lib.php");
```

(3) **Instantiieren und Konfigurieren des Parsers.** Der Parser benötigt zunächst eine Instanz seiner selbst, um mit ihm arbeiten zu können. Eine neue Instanz wird mit

```
$parser = new LaTeXHTML();
```

irgendwo im PHP-Quelltext aufgerufen. Nun können wir mittels der oben beschriebenen Methoden die Eigenschaften des Parsers verändern, also ein zu verarbeitendes Layout, ein Template und Daten übergeben.

```
$parser->useTemplate("/studienarbeit/templates/pruefung.htm");
$parser->useLayout("/studienarbeit/templates/a4hoch.tex");
$parser->setData($data);
```

Natürlich sollte das Template sinnvolle Tabellen- und Platzhalterdefinitionen beinhalten, im Layout ein Container eingefügt sein und die Schlüssel des `data`-Arrays sollten mit den Platzhaltern im Template korrespondieren.

(4) **Aufruf des Parsers.** Sind alle Einstellungen erfolgt, so kann der Parser aufgerufen und das Ergebnis geprüft werden:

```
$return = $p->buildFile("/studienarbeit/output/pruefung.tex");
if ($return == true) {
    echo "Generierung erfolgreich";
} else {
    echo "Fehler! Siehe this->debug[] für Details.";
}
```

Hier wird im Erfolgsfall die Datei `/studienarbeit/output/pruefung.tex` erzeugt und im selbigen Verzeichnis das PDF-Ausgabedokument generiert.

# Kapitel 5

# Fazit

Das automatisierte Erstellen von PDF-Dokumenten im Internet mittels LaTeX ist ein einfaches und kostengünstiges Verfahren. Es findet jedoch gerade im nichtwissenschaftlichen Bereich kaum Anwendung, da LaTeX dort unbekannt ist oder als zu kompliziert eingeschätzt wird. Meist kommt kommerzielle Software zum Einsatz, wie beispielsweise der Adobe Document Server. Jedoch zeigt diese Arbeit Möglichkeiten auf, derartige Dokumente durch das schnelle und frei verfügbare LaTeX-System generieren zu lassen.

Speziell bei der Erstellung von Tabellen bieten sich jedoch in LaTeX viele verschiedene, komplizierte Wege an, diese zu gestalten. Setzt man Tabellen manuell, so wird der LaTeX-Code – speziell bei sehr großen und komplexen Tabellen – sehr unübersichtlich, insbesondere wenn Zellen über mehrere Tabellenspalten gesetzt werden. Hier setzt das in der Studienarbeit vorgestellte System LateXHTML an. Es wurde eine an XHTML angelehnte Markup-Sprache entworfen, welches die Möglichkeit beinhaltet, Tabellen in für Anwender sehr anschaulichen WYSIWYG-Editoren zu erstellen und danach in LaTeX-Code umzuwandeln zu lassen.

Die Probleme, die sich ergeben, wenn Tabellen für einzelne Seiten „zu groß“ werden können, können auf mehrere Arten umgangen werden. Hierzu ist es jedoch notwendig, das Dokument mehrere Male zu compilieren, was meiner Ansicht nach einen Nachteil der aktuellen LaTeX-Version darstellt. Eine Lösung durch die `longtable`-Umgebung ist daher zwar denkbar, jedoch stellte ich bei Versuchen fest, dass bereits minimale Änderungen in der Zeilenanzahl oder in der Veränderung der Spaltenbreiten ein mehrfaches Neucompilieren notwendig machten. Trotzdem kann die `longtable`-Umgebung auch benutzt werden, wenn man sie explizit einschaltet.

Die in dieser Arbeit vorgestellte Lösung kann ebenfalls noch verbessert werden, indem man zwischen Erst- und Folgeseiten unterscheidet (was es nicht notwendig machen würde, lange Tabellen stets am Seitenanfang einer freien Seite zu platzieren). Dies würde jedoch weitere Compilierungsschritte mit sich bringen. Um jedoch generell diese Restriktion zu umgehen, wird in LateXHTML das `longtable`-Paket unterstützt.

Große Vorteile der Übersetzung von XHTML in LaTeX bestehen jedoch darin, dass man zeilen- und spaltentrennende Linien wesentlich intuitiver in XHTML setzen kann. Ebenso können Templates – insofern sie einmal für eine Ausgabe auf einer Webseite programmiert wurden – meist ohne große Veränderungen durch das hier vorgestellte System verwendet und verarbeitet werden. Ebenfalls kann dieser Übersetzer zu einem kompletten XHTML-to-LaTeX-Übersetzer weiterentwickelt werden.

Jedoch zeigt sich, daß das automatische Generieren von Dokumenten durch die heute zur Verfügung stehenden Mittel nach wie vor problematisch bleibt. Dies bezieht sich sowohl auf LateXHTML, jedoch auch auf die Darstellung von Inhalten von Webseiten. Werden beispielsweise durch Benutzer sehr lange Worte gebildet (was in der deutschen Sprache durchaus legitim sein kann), so können Layouts auf Webseiten wie in PDF-Dokumenten auf einfache Weise zum Kollaps gebracht werden.

# Literaturverzeichnis


[1] Walter Schmidt u.a.: *LaTeX2e Kurzbeschreibung*, ftp://ftp.dante.de/tex-archive/info/lshort/german/

[2] Manuela Jürgens: *LaTeX- eine Einführung und ein bißchen mehr...*, Teil 1, ftp://ftp.fernuni-hagen.de/pub/pdf/urz-broschueren/broschueren/a0260003.pdf

[3] Manuela Jürgens: *LaTeX- Fortgeschrittene Anwendungen*, Teil 2, ftp://ftp.fernuni-hagen.de/pub/pdf/urz-broschueren/broschueren/a0279510.pdf

[4] Michael Goossens u.a: *Der LaTeX-Begleiter* Addison Wesley, ISBN 3-89319-646-3

[5] Ingo Klöckl: *LaTeXTipps & Tricks,* dpunkt.verlag, ISBN 3-89864-145-7

[6] Wikipedia: *XHTML* http://de.wikipedia.org/wiki/XHTML/

[7] XHTML-Spezifikation: http://www.w3.org/TR/xhtml2/

[8] LaTeX-Tabellen-Website: http://www.informatik.hu-berlin.de/~musidlow/latex/Tabellen.html