

# **Einführung in die *eXtensible Markup Language* (XML)**

**Ursprünglich: Unterlagen zu einem Tutorium bei der  
TeX-Tagung DANTE2001 am 28.2.2001 in Rosenheim**

Günter Partosch, HRZ Gießen

*Das Tutorium „Einführung in XML“ richtet sich in erster Linie an Einsteiger in den Themenkomplex „Extensible Markup Language (XML)“. Vorkenntnisse sind nicht erforderlich.*

*Extensible Markup Language ist ein einfaches, sehr flexibles und von SGML abgeleitetes Text-Format, das eine zunehmende Rolle bei Austausch und Archivierung von Textdaten spielt.*

*Im Tutorium wird zunächst die Verwandtschaft von XML zu SGML und HTML aufgezeigt; im folgenden wird dann anhand von kleinen Beispielen demonstriert, wie XML-Dokumente erstellt, bearbeitet und präsentiert werden können.*

Die jeweils neueste Version dieser Tutoriumsunterlagen finden Sie beim URL <http://www.uni-giessen.de/~g029/xml/>.

Anregungen, Wünsche oder Fehlerkorrekturen können Sie mir natürlich gerne zusenden (an [Guenter.Partosch@hrz.uni-giessen.de](mailto:Guenter.Partosch@hrz.uni-giessen.de)).

## Inhaltverzeichnis

<b>1 Ausgangssituation und Problemstellung.....</b>	<b>5</b>
<b>2 Verschiedene Markup-Typen.....</b>	<b>6</b>
<b>3 SGML (<i>Standard Generalized Markup Language</i>) .....</b>	<b>7</b>
3.1 Geschichtliche Entwicklung.....	7
3.2 Ideen und Konzepte .....	7
3.2.1 Dokumenttypen .....	7
3.2.2 Deskriptives Markup .....	8
3.2.3 Aufbau eines SGML-Dokuments.....	8
3.3 SGML-Deklaration .....	8
3.4 DTD ( <i>Document Type Definition</i> ).....	10
3.4.1 Element-Deklaration .....	11
3.4.2 Attribut-Deklaration .....	12
3.4.3 Entity-Deklaration .....	12
3.5 Die Dokument-Instanz.....	13
3.6 Literatur und andere Informationsmöglichkeiten (Auswahl).....	13
<b>4 HTML (<i>HyperText Markup Language</i>) .....</b>	<b>14</b>
4.1 Geschichtliche Entwicklung.....	14
4.2 HTML-Konzepte .....	14
4.3 Aufbau eines HTML-Dokuments.....	15
4.4 Die drei DTDs für HTML 4.01 .....	16
4.5 Formatierung von HTML-Dokumenten mit CSS.....	16
4.6 Literatur und andere Informationsmöglichkeiten (Auswahl).....	18
<b>5 XML (<i>eXtensible Markup Language</i>).....</b>	<b>19</b>
5.1 Gründe für XML.....	19
5.2 Entwurfsziele für XML.....	19
5.3 XML-Konzepte.....	20
5.3.1 „Wohlgeformte“ und gültige XML-Dateien .....	20
5.3.2 XML-Deklaration ( <i>XML declaration</i> ).....	20
5.3.3 Dokument-Typdeklaration ( <i>document type declaration</i> ).....	21
5.4 Möglichkeiten zur Darstellung von XML-Dokumenten .....	22
5.4.1 Transformation durch spezialisierte Programme .....	22
5.4.2 Darstellung mittels DSSSL ( <i>Document Style Semantics and Specification Language</i> ) und eines DSSSL-Prozessors.....	23
5.4.3 Darstellung mittels CSS ( <i>Cascading Style Sheets</i> ) und Browser.....	23
5.4.4 Transformation mittels XSL ( <i>eXtensible Style Language</i> ) und Browser.....	23

5.4.5	Transformation mittels XSL ( <i>eXtensible Style Language</i> ) und eines XSL(T)-Prozessors .....	24
5.4.6	Auslieferung durch den Server als HTML-Dokument .....	25
5.5	Begleitende Konzepte .....	25
5.6	Einige standardisierte XML-Applikationen .....	26
5.6.1	Mathematical Markup Language (MathML) .....	26
5.7	Literatur und andere Informationsmöglichkeiten (Auswahl) .....	27
<b>6</b>	<b>Zukünftige Entwicklungen .....</b>	<b>28</b>
<b>7</b>	<b>Zwei kleine XML-Beispiele.....</b>	<b>30</b>
7.1	Gedichtsammlung .....	30
7.1.1	Erstellen der XML-Datei.....	30
7.1.2	Darstellen der XML-Datei .....	31
7.2	Tabelle .....	34
7.2.1	Erstellen der XML-Datei.....	34
7.2.2	Darstellen der XML-Datei .....	37
7.2.3	Umformen der XML-Datei durch ein spezialisiertes Programm.....	38
7.2.4	Darstellen der XML-Datei mit Hilfe einer CSS-Datei und eines Browsers .....	39
7.2.5	Transformation in eine HTML-Datei mit Hilfe von XSL und eines Browsers .....	41
7.2.6	Transformation der XML-Datei in eine HTML-Datei (mit Sortierung).....	43
7.2.7	Transformation der XML-Datei in eine LaTeX-Datei.....	45

## Abbildungsverzeichnis

Abbildung 1:	Darstellung der XML-Datei <code>gedicht0.xml</code> im Internet Explorer mittels des Default-Style-Sheets.....	32
Abbildung 2:	Darstellung der XML-Datei <code>gedicht.xml</code> im Internet Explorer mittels der CSS-Datei <code>gedicht.css</code> .....	33
Abbildung 3:	Ausgangstabelle in WinWord .....	34
Abbildung 4:	ASCII-Rohfassung <code>adr.txt</code> der Tabelle in WinWord .....	35
Abbildung 5:	Darstellung der XML-Datei <code>adr2.xml</code> im XML Notepad .....	38
Abbildung 6:	Darstellung der XML-Datei <code>adr3.xml</code> im Mozilla mittels der CSS-Datei <code>adr.css</code> .....	41
Abbildung 7:	Darstellung der XML-Datei <code>adr5.xml</code> im Internet Explorer mittels der XSL-Datei <code>adr.xsl</code> .....	43
Abbildung 8:	Darstellung der XML-Datei <code>adr6.xml</code> im Internet Explorer mittels der XSL-Datei <code>adr2.xsl</code> .....	45
Abbildung 9:	Darstellung der DVI-Datei <code>adr5.dvi</code> im DVI-Previewer Yap .....	49

# 1 Ausgangssituation und Problemstellung

(noch nicht vollständig realisiert)

## **Anforderungen an Texte, die elektronisch erfasst und verbreitet werden**

Damit ein Text mit angemessenem Aufwand weltweit einsetzbar, austauschbar und verarbeitbar ist und leicht an verschiedene (Rand-)Bedingungen angepasst werden kann, sollte er folgenden Anforderungen genügen:

- *international*: Der Text ist weltweit lesbar und interpretierbar.
- *plattformunabhängig*: Die Verarbeitung des Textes ist nicht abhängig von Rechnertyp oder Betriebssystem.
- *applikationsunabhängig*: Firmenspezifische Software darf nicht bestimmen, was lesbar/verarbeitbar ist.
- *alterungsbeständig*: Der Text ist auch nach Jahren noch verarbeitbar.
- *ausgebbar in verschiedenen Formaten*

Die meisten Anforderungen lassen sich vergleichsweise unaufwändig und allgemein durch eine innere Strukturierung der Texte verwirklichen, z.B. durch logisches/deskriptives Markup.

## 2 Verschiedene Markup-Typen

(noch nicht vollständig realisiert)

### Physisches Markup (Präsentations-Markup):

- Verschiedene Textteile sind mit verschiedenen Zeichen-/Absatz-Formatierungseigenschaften versehen.
- Beginn/Ende solcher Textabschnitte sind gekennzeichnet; z.B. durch Umschalt-Sequenzen oder spezielle Konstrukte:

```
Das ist [fett]fetter Text [/fett] .
```

oder

```
Das ist {\bf fetter Text} .
```

- zu wenig variabel, wenn Formatierungen nicht darstellbar sind oder geändert werden sollen
- inhaltliche Struktur nicht erkennbar/erschließbar

### Inhaltliches Markup (Deskriptives Markup):

- Man beschreibt nicht, in welchem Font, in welcher Größe, in welcher Ausprägung ein Textabschnitt präsentiert werden soll.
- Man beschreibt vielmehr, welche inhaltliche Bedeutung die einzelnen Textteile haben.

```
Das ist <em>hervorgehobener</em> Text .
```

oder

```
Das ist \emph{hervorgehobener} Text .
```

- Die tatsächliche Realisierung geschieht an anderer Stelle, z.B. durch bestimmte Einstellungen des verarbeitenden Programms bzw. durch Style-Sheets.

## 3 SGML (*Standard Generalized Markup Language*)

### 3.1 Geschichtliche Entwicklung

- 1967 Vorschlag von William Tunnicliffe (GCA = *Graphics Communications Association*), die Struktur eines Dokumentes von dessen Layout zu trennen
- Stanley Rice: Konzept der *Editorial Structure Tags*; diese beiden Ideen bilden das Grundkonzept des *Generic Coding* (im Gegensatz zum *Specific Coding*).
- 1969: Charles Goldfarb, Edward Mosher und Raymond Lorie (alle IBM) entwickeln GML (*Generalized Markup Language*) basierend auf den Ideen von Tunnicliffe und Rice.
- 1978: Beginn der Entwicklung eines Standards für Auszeichnungssprachen (basierend auf GML) beim ANSI (*American National Standard Institute*)
- 1978-1984: mehrere Normentwürfe
- 1984: Reorganisation des Projekts und Mitarbeit der ISO (*International Organization of Standardization*)
- 1985: letzter Entwurf
- 1986: endgültige Version von SGML (*Standard Generalized Markup Language*) als ISO-Standard 8879 verabschiedet

### 3.2 Ideen und Konzepte

#### 3.2.1 Dokumenttypen

- In SGML wird jedem Dokument ein bestimmter Dokumenttyp zugeordnet (grundlegende Idee: bestimmte Gruppen von Dokumenten haben ähnliche Strukturen). Beispiele für solche Gruppen von Dokumenten sind Bücher, Briefe, Artikel, Berichte, usw.
- Die Struktur einer Klasse von Dokumenten wird in einer *Document Type Definition* (DTD) festgelegt.
- Welchen Typ ein konkretes Dokument besitzt, wird am Anfang des Dokumentes in der Dokumenttypdeklaration spezifiziert.
- Solche so vereinheitlichten Dokumente lassen sich leicht(er) weiter verarbeiten.

### 3.2.2 Deskriptives Markup

- SGML verwendet deskriptives Markup:
  - Ein SGML-Dokument enthält keine Angaben darüber, wie das Dokument formatiert werden soll.
  - Das Dokument besteht nur aus den Textdaten und Informationen darüber, welche Textdaten zu welchen Textelementen gehören.
- Dies hat den Vorteil, dass ein einmal geschriebenes Dokument auf viele unterschiedliche Arten formatiert werden kann, ohne dass am eigentlichen Dokument etwas geändert werden muss.
- Zur Formatierung von SGML-Dokumenten können spezielle Programme und DSSSL (*Document Style Semantics and Specification Language*, ISO 10179) eingesetzt werden.

### 3.2.3 Aufbau eines SGML-Dokuments

Ein SGML-Dokument besteht aus drei Teilen:

- [Deklaration](#) (spezifiziert grundlegende Dinge, wie z.B. den verwendeten Zeichensatz)
- [Dokument-Typ-Definition](#) (legt fest, welche Strukturelemente ein Dokument enthalten darf und wie sie ineinander geschachtelt sind)
- [Dokument-Instanz](#) (enthält die eigentlichen Textdaten zusammen mit dem dazugehörigen Markup)

## 3.3 SGML-Deklaration

In der SGML-Deklaration werden bestimmte grundlegende Eigenschaften eines SGML-Dokuments festgelegt, beispielsweise:

- Zeichensatz, der bei der Verarbeitung des Dokumentes verwendet werden soll,
- Länge der Bezeichner für Elemente,
- Festlegung der zugelassenen und verbotenen Zeichen,
- Festlegung ihrer zahlenmäßigen Kodierung,
- Regeln für gültige Bezeichner und
- Aktivierung bzw. Deaktivierung bestimmter Syntaxelemente



**Beispiel:**

```
[1] <!SGML "ISO 8879:1986"  
  . .  
  . .  
[15] CAPACITY SGMLREF  
[16] TOTALCAP 35000  
  . .  
  . .  
[34] SCOPE DOCUMENT  
[35]  
[36] SYNTAX  
[37] SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13  
14 15 16 17  
[38] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 127 255  
[39] BASESET "ISO 646-1983//CHARSET  
[40] International Reference Version (IRV)//ESC 2/5 4/0"  
[41] DESCSET 0 128 0  
[42] FUNCTION RE 13  
[43] RS 10  
[44] SPACE 32  
[45] TAB SEPCHAR 9  
[46] NAMING LCNMSTRT ""  
  . .  
  . .  
[52] DELIM GENERAL SGMLREF  
[53] SHORTREF SGMLREF  
[54] NAMES SGMLREF  
[55] QUANTITY SGMLREF  
[56] ATTCNT 40  
  . .  
  . .  
[72] FEATURES  
[73] MINIMIZE DATATAG NO OMITTAG YES RANK NO SHORTTAG  
YES  
[74] LINK SIMPLE NO IMPLICIT NO EXPLICIT NO  
[75] OTHER CONCUR NO SUBDOC NO FORMAL NO  
[76] APPINFO NONE>
```

**Erläuterungen:**

- [15] nach CAPACITY: Angaben über die benötigte Speicherkapazität
- [39] nach BASESET: Angabe des verwendeten Basiszeichensatzes (hier ISO 646)
- [42] Zeichen-Codes und ihre vordefinierte Bedeutung
- [46] Konventionen bezüglich der erlaubten Bezeichner
- [73] Minimierungsregeln für Tags

**3.4 DTD (*Document Type Definition*)**

Eine vollständige SGML-Dokument-Instanz könnte beispielsweise wie folgt aussehen:

```
<!DOCTYPE sammlung SYSTEM "sammlung.dtd">
<sammlung>
  <gedicht>
    <kopf>
      <titel>The SICK ROSE</titel>
      <autor>Unknown Poet</autor>
      <jahr/>
    </kopf>
    <strophe>
      <zeile>O Rose thou art sick.</zeile>
      <zeile>The invisible worm,</zeile>
      <zeile>That flies in the night</zeile>
      <zeile>In the howling storm:</zeile>
    </strophe>
    <strophe>
      <zeile>Has found out thy bed</zeile>
      <zeile>Of crimson joy:</zeile>
      <zeile>And his dark secret love</zeile>
      <zeile>Does thy life destroy.</zeile>
    </strophe>
    <strophe>
      <zeile>Has found out thy bed</zeile>
      <zeile>Of crimson joy:</zeile>
      <zeile>And his dark secret love</zeile>
      <zeile>Does thy life destroy.</zeile>
    </strophe>
  </gedicht>
</sammlung>
```

Die erste Zeile enthält hier einen Verweis auf die zugehörige DTD `sammlung.dtd`.

Die *Document Type Definition* (DTD) gibt an,

- welche Textelemente ein Dokument enthalten darf und
- wie diese ineinander geschachtelt sein dürfen.

Eine zu dem obigen Beispiel passende DTD `sammlung.dtd` könnte in SGML wie folgt aussehen:

```
<!--      Element   Min.  Inhaltsmodell      -->
<!ELEMENT sammlung - - (gedicht*)              >
<!ELEMENT gedicht  - O (kopf?, strophe+)        >
<!ELEMENT kopf     - - (titel?, autor?, jahr?)   >
<!ELEMENT titel    - O (#PCDATA)                >
<!ELEMENT autor    - O (#PCDATA)                >
<!ELEMENT jahr     - O (#PCDATA)                >
<!ELEMENT strophe  - O (zeile+)                 >
<!ELEMENT zeile    - O (#PCDATA)                >
```

- Eine `sammlung` enthält also beliebig viele (\*) Gedichte (`gedicht`).
- Ein `gedicht` besteht aus einem optionalem `kopf` (?), gefolgt von mindestens einer `strophe` (+).
- Der `kopf` kann die Elemente `titel`, `autor` und `jahr` enthalten.
- Eine `strophe` enthält mindestens eine `zeile` (+).

### 3.4.1 Element-Deklaration

Die obige Beispiel-DTD besteht aus einer Kommentarzeile und acht Zeilen, in denen jeweils ein Element deklariert wird:

- Eine Elementdeklaration beginnt immer mit der Zeichenkette `<!ELEMENT`, gefolgt von dem Elementnamen, den Minimierungsregeln und dem Inhaltsmodell. Sie endet mit dem Zeichen `>`.
- Der Elementnamen wird in der Dokument-Instanz innerhalb der Start- und End-Tags benutzt.
- Die Minimierungsregeln geben Auskunft darüber, ob Start- und End-Tags zwingend vorhanden sein müssen oder ob sie weggelassen werden können. Das Zeichen `-` besagt, dass ein Tag vorhanden sein muss, während das Zeichen `O` anzeigt, dass das Tag optional ist.
- Das Inhaltsmodell gibt an, in welcher Reihenfolge aus welchen Tochterelementen das Element bestehen kann.
- Wie oft ein Element auftreten kann, wird durch eine Häufigkeitsanzeige (nichts, `+`, `*` oder `?`) angedeutet.

- Besteht das Inhaltsmodell aus mehreren unterschiedlichen Elementen, so wird die Reihenfolge, in der die Elemente auftauchen dürfen, durch *Group Connectors* (Komma, | oder &) festgelegt.
- #PCDATA (*parsed character data*) steht für Text.

### 3.4.2 Attribut-Deklaration

Die in der DTD aufgeführten Elemente können durch Attribute genauer spezifiziert werden:

<!--	Elem.	Attr.	Wertebereich	Default	-->
<!	ATTLIST	bild	name	<u>CDATA</u>	<u>#REQUIRED</u>
			groesse	CDATA	<u>#IMPLIED</u>
			titel	CDATA	"Default"
			typ	(inline display)	'inline' >

- Attribute enthalten Informationen, die für eine bestimmte Elementinstanz spezifisch sind, jedoch nicht zum Inhalt des Elementes gehören. Attribute werden in der DTD definiert.
- Jedes Attribut benötigt drei Angaben: Namen, Wertebereich und Default-Wert.
- CDATA (*raw, unparsed text*) deutet eine Zeichenkette an.
- Die Angabe #REQUIRED als Default-Wert bedeutet, dass dieses Attribut in der Dokument-Instanz vom Autor immer mit einem Wert versehen werden muss.
- Die Angabe #IMPLIED als Default-Wert bedeutet, dass dieses Attribut keinen explizit definierten Default-Wert besitzt. Wird das Attribut vom Autor nicht festgelegt, so verwendet der SGML-Prozessor einen Wert, der irgendwo implizit in der Dokument-Instanz enthalten ist.

### 3.4.3 Entity-Deklaration

- Mit Hilfe einer Entity wird einem bestimmten Objekt ein symbolischer Namen zugewiesen.
- Um dieses Objekt in der Dokument-Instanz zu verwenden, wird an der entsprechenden Stelle eine Entity-Referenz eingefügt, in der Form  
... &name; ...

#### Einsatzmöglichkeiten:

- Verteilung eines Dokumentes auf mehrere Dateien:

```
<!ENTITY kap1 SYSTEM "kap1.sgml">
... &kap1; ...
```

- Verwaltung von Textbausteinen:

```
<!ENTITY XML "eXtensible Markup Language">
... &XML; ...
```

- Systemabhängige Zeichensätze ([weitere Beispiele](#)):

```
<!ENTITY quot "&#34;">
... &quot; ...
```

- Integration von Binärdaten

### 3.5 Die Dokument-Instanz

Die Dokument-Instanz besteht aus dem eigentlichen Inhalt des Dokumentes:

- Text
- Auszeichnungen
- [Entity-Referenzen](#)

### 3.6 Literatur und andere Informationsmöglichkeiten (Auswahl)

- [comp.text.sgml](#) (Usenet-News-Gruppe)
- [sgml-l@urz.uni-heidelberg.de](#) (E-Mail-Diskussionsliste)
- Goldfarb, Charles F.: *The SGML Handbook*; Oxford University Press, Oxford; 1990; ISBN 0198537379
- Bryan, Martin: *SGML. An Author's Guide to the Standard Generalized Markup Language*; Addison-Wesley, Wokingham; 1993; ISBN 0-201-17535-5
- Szillat, Horst: *SGML. Eine praktische Einführung*; ITP, Bonn; 1995; ISBN 3-929821-75-3

## 4 HTML (*HyperText Markup Language*)

### 4.1 Geschichtliche Entwicklung

- 1989 Tim Berners-Lee (CERN) veröffentlicht das Dokument *WWW Project Proposal*: Verwaltung von Informationen mit Hilfe eines verteilten Hypertextsystems.
- Tim Berners-Lee entwickelt eine auf SGML basierende Auszeichnungssprache (später *HyperText Markup Language* genannt).
- Juni 1993 Dave Raggett und Tim Berners-Lee veröffentlichen das Internet-Draft-Dokument *HyperText Markup Language, Ver 1.0*.
- April 1993 erste Version von XMosaic (mit eigenen HTML-Erweiterungen)
- April 1994 Dan Connolly veröffentlicht erstes Internet-Draft zu HTML 2.0.
- September 1995 [HTML 2.0](#) wird Standard.
- Anfang 1996 Firmeneigene Entwicklungen bringen HTML 3.0 zu Fall.
- Mai 1996 *Editorial Review Board* des *World Wide Web Consortium* (W3C) stellt [HTML 3.2](#) vor (abwärtskompatibel zu HTML 2.0; mit einigen Merkmalen von HTML 3.0, aber mit vielen Elementen für Layout-Zwecke)
- November 1997 wieder stärkere Trennung von Struktur und Layout mit HTML 4.0 (<http://www.w3.org/TR/html40/>)
- Dezember 1999 HTML 4.01: überarbeitete Fassung von HTML 4.0 (<http://www.w3.org/TR/html401/>); Abschluss der eigentlichen HTML-Entwicklung

### 4.2 HTML-Konzepte

- HTML ist eine speziell für das Internet entwickelte Dokumentenbeschreibungssprache.
- HTML ist eine SGML-Applikation.
- Die Syntax wird durch eine SGML-Deklaration und eine Dokument-Typ-Definition (DTD) festgelegt.
- Zusätzlich enthält die Spezifikation einige umgangssprachlich formulierte Nebenbedingungen, die nicht durch eine DTD dargestellt werden können.
- Mit HTML kann (könnte!) ein Dokument mit einer Struktur versehen werden.

- Innerhalb eines Dokuments können Verweise auf andere WWW-Ressourcen zeigen.
- HTML-Dokumente werden üblicherweise mit Hilfe eines Browsers dargestellt.
- Um den verschiedenen Browser-Typen zu ermöglichen, ein Dokument sinnvoll darzustellen, sollte zwischen Inhalt, Struktur und Darstellung eines Dokumentes klar getrennt werden.
- Das Konzept der Trennung von Struktur und Layout war in der [HTML-2.0](#)-Spezifikation noch streng realisiert.
- Durch die Einführung neuer firmenspezifischer Elemente zur Layout-Darstellung wurde dieses Prinzip entscheidend aufgeweicht und nicht jedes Dokument konnte auf allen Browsern angemessen dargestellt werden.
- Mit [HTML 3.2](#), in das zahlreiche proprietäre Elemente aufgenommen wurden, wurde das Problem der Vermischung von Struktur und Layout noch verschärft. HTML entsprach damit nicht mehr der ursprünglichen SGML-Philosophie.
- Mit der Entwicklung von [HTML 4.0](#) reagierte das W3C auf dieses Problem.
- Da HTML 4.0 (HTML 4.01) abwärtskompatibel zu HTML 3.2 sein sollte, wurden [drei DTDs](#) definiert, die jeweils eine unterschiedlich starke Trennung von Struktur und Layout realisieren.

## 4.3 Aufbau eines HTML-Dokuments

Ein HTML-Dokument besteht aus drei Teilen:

- (optionale) Angabe über die verwendete HTML-Version
- Kopf (HEAD) des Dokumentes
- Inhalt (BODY) des Dokumentes

### Beispiel:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>

<head><title>Das ist der Titel!</title>
... andere HEAD-Elemente ...
</head>

<body>
... eigentlicher Inhalt des Dokuments ...
</body>

</html>
```

- Das Beispieldokument entspricht der [Strict-DTD](#) (*HTML 4.01 Strict DTD*).
- Nach der DOCTYPE-Deklaration folgen das head- und das body-Element, eingeschlossen vom html-Element.
- Das head-Element enthält u.a. Informationen, die ggf. von Suchmaschinen zur Indizierung von Web-Dokumenten verwenden können.
- Das title-Element, das immer vorhanden sein muss, enthält üblicherweise den Titel des Dokumentes.
- Der Inhalt des HTML-Dokumentes befindet sich im body-Element.

## 4.4 Die drei DTDs für HTML 4.01

- *HTML 4.01 Strict DTD* enthält alle Elemente und Attribute aus HTML 3.2, die nicht verworfen wurden.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html401/strict.dtd">
```

- Die *HTML 4.01 Transitional* oder *loose DTD* enthält neben den Elementen und Attributen der *Strict DTD* auch die verworfenen Elemente und Attribute.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html401/loose.dtd">
```

- *HTML 4.01 Frameset DTD*: Sie enthält alle Elemente und Attribute der *Transitional DTD* und zusätzlich die für Frames benötigten Elemente.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Frameset//EN"
"http://www.w3.org/TR/html401/frameset.dtd">
```

## 4.5 Formatierung von HTML-Dokumenten mit CSS

Mit Hilfe der *Cascading Style Sheets* können Sie trotz einer strengen Trennung von Struktur und Layout optisch ansprechende WWW-Dokumente gestalten.



**Beispiel 1** (Style lokal bei einem Element aufgeführt):

```
<html>
<head><title>Beispiel 1</title></head>
<body>
<h1 style="color:red; font-size:36pt;">Überschrift</h1>
</body>
</html>
```

Durch dieses Vorgehen eröffnen sich zwar zahlreiche neue Möglichkeiten zur Gestaltung des Layouts; die enge Verquickung von Layout und Struktur entspricht aber nicht der CSS-Philosophie und ist deshalb nicht zu empfehlen.

**Beispiel 2** (Definition des Style-Sheets im Dokumentkopf):

```
<html>
<head>
<title>Beispiel 2</title>
<style type="text/css">
h1 { font-size:48pt; color:red; font-style:italic; }
</style>
</head>
<body>
<h1>Überschrift</h1>
</body>
</html>
```

Damit haben Sie für (fast) alle Elemente Ihres Dokuments eine zentrale Stelle zur Änderung des Layouts.

**Beispiel 3** (externe Definition des Style-Sheets):

```
<html>
<head>
<title>Beispiel 3</title>
<link rel="stylesheet" type="text/css" href="form.css">
</head>
<body>
<h1>Überschrift</h1>
</body>
</html>
```

Mit der Style-Datei `form.css` haben Sie für (fast) alle Elemente in Ihren Dokumenten eine zentrale Stelle zur Festlegung des Layouts.

## 4.6 Literatur und andere Informationsmöglichkeiten (Auswahl)

- einige E-Mail-Diskussionslisten
- viele Usenet-News-Gruppen
- Stefan Münz: *Die Energie des Verstehens: SELFHTML – HTML-Dateien selbst erstellen* (<http://www.teamone.de/selfhtml/>)
- Stefan Münz: *HTML 4.0 Handbuch*; Franzis, Poing; 1999; ISBN 3-7723-7514-6

## 5 XML (eXtensible Markup Language)

Aus den Erfahrungen von SGML und HTML wurde XML vom W3C als eine stark vereinfachte SGML-Teilmenge entwickelt:

1996: Beginn der Entwicklung

1997: *Proposed Recommendation*

(<http://www.w3.org/TR/PR-xml-971208>)

1998: *Recommendation* (<http://www.w3.org/TR/REC-xml>)

### 5.1 Gründe für XML

Gewichtige Nachteile von [HTML](#):

- Syntax und Bedeutung von Elementen und Attributen festgeschrieben;
- keine klare Trennung zwischen Struktur, Inhalt und Darstellung;
- HTML eher auf die Präsentation des Textes am Bildschirm ausgerichtet;
- viele firmenspezifische Erweiterungen gegenüber dem Standard;
- viele Programme produzieren ungültigen HTML-Code;
- (ungültige) HTML-Dokumente, die zwar von einigen Browsern dargestellt werden, erschweren den Datenaustausch;
- kaum strukturbeschreibende oder bedeutungsbeschreibende Elemente;
- nur eingeschränkte Unterstützung von Metadaten;
- viele strukturelle Möglichkeiten zum Datenaustausch fehlen;
- fehlende moderne Features, z.B. Dokument-Objekt-Modell

Gewichtige Nachteile von [SGML](#):

- SGML ist hochkomplex und daher nur sehr schwer zu erlernen;
- SGML ist nur sehr aufwändig implementierbar;
- hoher Pflegeaufwand für SGML-Dokumente;
- Verarbeitungsprogramme meist hochspezialisiert und teuer;
- SGML-Dokumente i.allg. ungeeignet zur Darstellung im WWW

### 5.2 Entwurfsziele für XML

- XML-Dokumente sind einfach zu erstellen.
- XML-Dokumente sind übersichtlich und können ohne zusätzliche Hilfsmittel gelesen werden.
- Programme zum Verarbeiten von XML-Dateien sind einfach zu entwickeln.
- XML-Dokumente sind ähnlich wie HTML-Dokumente schnell und einfach im WWW darstellbar.

- XML ist kompatibel mit SGML.
- Minimierung des Markups ist nicht zulässig.
- XML enthält möglichst wenige optionale Features.
- XML ist für eine breite Anwendungspalette einsetzbar.
- Der XML-Standard wird schnell verabschiedet.

## 5.3 XML-Konzepte

### 5.3.1 „Wohlgeformte“ und gültige XML-Dateien

Ein XML-Dokument, für das die folgenden Regeln gelten, ist „*wohlgeformt*“ (*well-formed*):

- Der Inhalt des Dokuments muss in genau einem Wurzelement eingeschlossen werden.
- Elemente mit optionalem Ende-Tag sind nicht zulässig.
- Alle leeren Elemente müssen geschlossen werden, also z.B. `<leer />` oder `<leer></leer>`.
- Groß- und Kleinschreibung bei Element- und Attribut-Namen wird unterschieden.
- Überlappende Elemente sind nicht erlaubt, also z.B. nicht `<wichtig>Hallo <ganzwichtig>Welt</wichtig></ganzwichtig>`.
- Alle Attribut-Werte müssen in Anführungszeichen eingeschlossen werden.
- Es gibt nur noch einige wenige vordefinierte Zeichen-Entities:

<code>&lt;!ENTITY lt</code>	<code>"&amp;#60;"</code>	<code>&lt;!-- "&lt;" --&gt;</code>
<code>&lt;!ENTITY gt</code>	<code>"&amp;#62;"</code>	<code>&lt;!-- "&gt;" --&gt;</code>
<code>&lt;!ENTITY amp</code>	<code>"&amp;#38;"</code>	<code>&lt;!-- "&amp;" --&gt;</code>
<code>&lt;!ENTITY apos</code>	<code>"&amp;#39;"</code>	<code>&lt;!-- "'" --&gt;</code>
<code>&lt;!ENTITY quot</code>	<code>"&amp;#34;"</code>	<code>&lt;!-- '"' --&gt;</code>

Ein wohlgeformtes XML-Dokument, das einer [DTD](#) gehorcht, ist *gültig* (*valid*).

### 5.3.2 XML-Deklaration (*XML declaration*)

In der XML-Deklaration wird – falls vorhanden – die XML-Version und optional der zugrunde liegende Zeichensatz spezifiziert:

```
<?xml version="1.0" encoding='ISO-8859-1' ?>
```

oder

```
<?xml version="1.0" standalone="yes" ?>
```

Mit der Angabe `standalone="yes"` teilt man dem XML-Prozessor mit, dass er nicht nach externen Bestandteilen (z.B. nach einer DTD) suchen soll. Voreingestellter Zeichensatz ist übrigens [UTF-8](#) bzw. [UTF-16](#).

### 5.3.3 Dokument-Typdeklaration (*document type declaration*)

Die DTD, die einem XML-Dokument zugrunde liegt, kann am Anfang des Dokuments spezifiziert werden:

```
<?xml version="1.0" ?>
<!DOCTYPE adressen SYSTEM "adressen.dtd">
...
```

bzw.

```
<?xml version="1.0" ?>
<!DOCTYPE adressen [
  <!ELEMENT adressen ...>
  <!ENTITY ...>
  ...
  <!ATTLIST ...>
  ...
]>
...
```

#### Element-Deklaration:

Alle Elemente werden nach dem folgenden allgemeinen Schema vereinbart:

```
<!ELEMENT name inhalt>
```

mit EMPTY, ANY, *inhaltsmodell* als Möglichkeiten für *inhalt*.

- Im *inhaltsmodell* werden der Inhaltstyp, die möglichen Tochter-Elemente, ihre Reihenfolge bzw. ihre Häufigkeit aufgeführt. Beispiele:

```
<!ELEMENT sammlung (gedicht*) >
<!ELEMENT gedicht (kopf, strophe+) >
<!ELEMENT kopf (titel?, autor?, jahr?) >
<!ELEMENT titel (#PCDATA) >
...
```

- `sammlung` enthält beliebig viele (\*) Elemente der Art `gedicht`.
- Ein `gedicht` besteht aus genau einem `kopf` und mindestens einer (+) `strophe`.

- Der kopf besteht aus den optionalen (?) Teilen `titel`, `autor`, `jahr` (in dieser Reihenfolge).
- Das Element `titel` enthält Daten, die weiter untersucht werden können.

### Attribut-Deklaration:

XML-Elemente können nach folgendem Schema zusätzlich durch Attribute genauer festgelegt werden:

```
<!ATTLIST element name wertebereich voreinstellung
...>
```

- Für *wertebereich* wird am häufigsten CDATA (Zeichenkette) oder eine *aufzählung* angegeben.
- Üblich für *voreinstellung* sind #REQUIRED (zwingend), #IMPLIED (implizit) oder ein konkreter Weglasswert.

### Beispiel:

```
<!ATTLIST bild name      CDATA #REQUIRED
               groesse   CDATA #IMPLIED
               titel      CDATA "Weglasstitel"
               typ        (inline|display) 'inline'>
```

### Entity-Deklaration:

Mit Hilfe von [Entities](#) kann zusätzliches Material (Textfragmente, Spezialzeichen, Bilder, externe Dateien) in ein XML-Dokument eingebracht werden.

## 5.4 Möglichkeiten zur Darstellung von XML-Dokumenten

### 5.4.1 Transformation durch spezialisierte Programme

- Da gültige XML-Dokumente auch gültige SGML-Dokumente sind, lassen sich eigentlich alle SGML-Transformations-Tools einsetzen.
- Da Syntax und Struktur bei XML einfach sind, lassen sich mit geringem Aufwand auch eigene Programme für diesen (einen) Zweck entwickeln. [Beispiel](#)

### 5.4.2 Darstellung mittels DSSSL (*Document Style Semantics and Specification Language*) und eines DSSSL-Prozessors

#### Vorteile von DSSSL:

- sehr mächtig;
- Fast alle gewünschten Umformungen und Formatierungen lassen sich bewerkstelligen.

#### Nachteile von DSSSL:

- viel zu komplex und aufwändig
- viel zu schwierig zu erlernen (LISP-artige Programmierung!)

### 5.4.3 Darstellung mittels CSS (*Cascading Style Sheets*) und Browser

Mit Hilfe von CSS ([CSS1](#), [CSS2](#)) lässt sich eine XML-Datei relativ einfach in einem XML-fähigen Browser darstellen ([Beispiel 1](#), [Beispiel 2](#)):

```
<?xml version="1.0" encoding='ISO-8859-1' ?>
<?xml-stylesheet href="adr.css" type="text/css" ?>
<!DOCTYPE adressen SYSTEM "adressen.dtd">
...
```

#### Nachteile:

- Auswertung der Attribut-Werte nicht generell möglich (insbesondere nicht in CSS1)
- Ausgabe zusätzlicher Texte nur eingeschränkt möglich
- Änderung der vorgegebenen Reihenfolge nicht möglich
- Wiederverwendung von Elementinhalten nicht möglich
- CSS uneinheitlich bzw. unvollständig in den Browsern realisiert

### 5.4.4 Transformation mittels XSL (*eXtensible Style Language*) und Browser

Mit Hilfe einer [XSL](#)-Style-Datei lässt sich eine XML-Datei relativ einfach in einem XML-fähigen Browser umformen bzw. darstellen ([Beispiel 1](#), [Beispiel 2](#)):

```
<?xml version="1.0" encoding='ISO-8859-1' ?>
<?xml-stylesheet href="adr.xsl" type="text/xsl" ?>
<!DOCTYPE adressen SYSTEM "adressen.dtd">
...
```

**Vorteile:**

- sehr mächtige, XML-artige Programmiersprache
- (Fast) alle gewünschten Umformungen und Formatierungen lassen sich bewerkstelligen.
- leicht zu erlernen
- alle oben genannte [Nachteile von CSS](#) behoben

**Nachteile:**

- existiert erst seit dem 15.10.2001 als vollständiger Standard (vorher lediglich der Teil, der sich mit [Transformationen](#) beschäftigt)
- bisher nur wenige Browser verfügbar, die XSL/XSLT einsetzen können

**5.4.5 Transformation mittels XSL (eXtensible Style Language) und eines XSL(T)-Prozessors**

Mit Hilfe von XSL-Prozessoren kann aus einem XML-Dokument ein Dokument in einem anderen Format erzeugt werden. Mögliche Ausgabeformate sind z.B. ASCII, XML, TeX/LaTeX oder HTML; gesteuert wird der Prozess zumeist durch ein XSL-Style-Sheet.

Eine ausgezeichnete Übersicht über solche Programme liefert <http://www.garshol.priv.no/download/xmltools/>.

**Beispiel:** Mit dem folgenden Aufruf wird die XML-Datei `adr5.xml` – gesteuert durch die XSL-Style-Datei `adr-tex.xsl` – mittels des XSL-Prozessors **saxon** in die LaTeX-Datei `adr-tex.tex` transformiert:

```
saxon adr5.xml adr-tex.xsl > adr-tex.tex
```

Der Anfang der Stylesheet-Datei sieht dabei wie folgt aus (siehe [Beispiel](#)):

```
<?xml version='1.0' encoding="ISO-8859-1" ?>
<!-- adr-tex.xsl -->
<!-- XSL-Datei (passend zu adr5.xml -->
<!-- kann mit SAXON verarbeitet werden -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="ISO-8859-1" />
  <xsl:template match="/">
    \documentclass[10pt,a4paper]{article}
    \usepackage{adr-tex}

    \title{Adressen}
    \author{Emil Mayer}
```



```
\date{\today}

\begin{document}
\maketitle
\section*{Adressen}

...
```

#### 5.4.6 Auslieferung durch den Server als HTML-Dokument

In den vorangegangenen Abschnitten wurde gezeigt, wie Sie XML-Dokumente auf **Ihrem** Rechner darstellen können. Viel häufiger wird jedoch in der Praxis ein anderer Weg beschritten:

1. Sie fordern bei einem WWW-Server ein XML-Dokument an.
2. Der WWW-Server liefert jedoch nicht das XML-Dokument unmittelbar aus, sondern wandelt dieses „on the fly“ in ein HTML-Dokument um.
3. Dieses HTML-Dokument lässt sich mit großer Wahrscheinlichkeit auf Ihrem WWW-Browser darstellen.

Die dazu notwendigen Einstellungen und Vorkehrungen gehören zu den Aufgaben eines Web-Administrators und sollen daher hier nicht weiter beschrieben werden.

### 5.5 Begleitende Konzepte

#### Auswahl:

DOM *Document Object Model (DOM) Level 1 Specification, Version 1.0, W3C Recommendation 1 October, 1998*

(<http://www.w3.org/TR/REC-DOM-Level-1/>):

*This document contains the requirements for the Document Object Model, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents.*

XSL *Extensible Stylesheet Language (XSL) Version 1.0 W3C Recommendation 15 October 2001* (<http://www.w3.org/TR/xsl/>)

XSLT *XSL Transformations (XSLT), Version 1.0, W3C Recommendation 16 November 1999* (<http://www.w3.org/TR/xslt>)

XPath *XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999* (<http://www.w3.org/TR/xpath>):

*XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer.*

## 5.6 Einige standardisierte XML-Applikationen

Aufbauend auf XML wurden zahlreiche Applikationen als Standards entwickelt. Hier eine Auswahl der vom W3C betreuten Entwicklungen:

- 15.6.1998 *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification* (Recommendation), <http://www.w3.org/TR/REC-smil/>
- 22.2.1999 *Resource Description Framework (RDF) Model and Syntax Specification* (Recommendation),  
<http://www.w3.org/TR/REC-rdf-syntax>
- 3.3.2000 *Resource Description Framework (RDF) Schemas* (Candidate Recommendation), <http://www.w3.org/TR/rdf-schema/>
- 2.11.2000 *Scalable Vector Graphics (SVG) 1.0 Specification* (Candidate Recommendation), <http://www.w3.org/TR/SVG/>
- 21.2.2001 *Mathematical Markup Language (MathML) Version 2.0* (Recommendation), <http://www.w3.org/TR/MathML2>

### 5.6.1 Mathematical Markup Language (MathML)

W3C Recommendation, revision of 7 July 1999  
(<http://www.w3.org/TR/REC-MathML/>)

MathML ist eine Auszeichnungssprache, die speziell für die Beschreibung mathematischer Ausdrücke gedacht ist.

**Beispiel ( $x^2 + 4x + 2 = 0$ ):**

```
<mrow>
  <mrow>
    <msup><mi>x</mi><mn>2</mn></msup>
    <mo>+</mo>
    <mrow>
      <mn>4</mn>
      <mo>&InvisibleTimes;</mo>
      <mi>x</mi>
    </mrow>
    <mo>+</mo><mn>4</mn>
  </mrow>
  <mo>=</mo><mn>0</mn>
</mrow>
```

## 5.7 Literatur und andere Informationsmöglichkeiten (Auswahl)

- [comp.text.xml](#) (Usenet-News-Gruppe, englisch)
- [xml-de@LISTSERV.GMD.DE](#) (E-Mail-Diskussionsliste, deutsch)
- [xml-l@LISTSERV.HEANET.IE](#) (E-Mail-Diskussionsliste, englisch)
- Goldfarb, Charles F.; Prescod, Paul: *XML Handbuch*; Prentice Hall, München; 1999; ISBN 3-8272-9575-0
- Kay, Michael: *XSLT – Programmers's Reference*; Wrox Press, Birmingham; 2000; ISBN 1-861003-12-9
- Behme, Henning; Mintert, Stefan: *XML in der Praxis. Professionelles Web-Publishing mit der Extensible Markup Language*; Addison-Wesley, München; 2000; ISBN 3-82731-636-7
- Ray, Erik T.: *Einführung in XML*; O'Reilly, Köln; 2001; ISBN 3-89721-286-2
- Microsoft: *XML Tutorial*; <http://msdn.microsoft.com/library/en-us/xmlsdk30/htm/xmtutxmletutorial.asp>
- Microsoft: *XSL Developer's Guide*; <http://msdn.microsoft.com/library/en-us/xmlsdk30/htm/xmconxsldevelopersguide.asp>

## 6 Zukünftige Entwicklungen

- Laut Auskunft des W3C ist die Entwicklung von HTML abgeschlossen; nur noch geringfügige Fehlerkorrekturen werden vorgenommen (HTML 4.0, HTML 4.01). Siehe dazu (<http://www.w3.org/MarkUp/>).
- Auf der Basis von XML wird HTML als XHTML reformuliert.

Einige Meilensteine:

- 24.12.1999 *HTML 4.01 Specification* (Recommendation),  
<http://www.w3.org/TR/html401>
- 5.1.2000 *XHTML™ 1.1 - Module-based XHTML* (Working Draft),  
<http://www.w3.org/TR/xhtml11>
- 26.1.2000 *XHTML™ 1.0: The Extensible HyperText Markup Language – A Reformulation of HTML 4 in XML 1.0* (Recommendation),  
<http://www.w3.org/TR/xhtml1>
- 9.12.2000 *XHTML™ Basic* (Recommendation),  
<http://www.w3.org/TR/xhtml-basic>
- 23.2.2001 *Modularization of XHTML* (Proposed Recommendation),  
<http://www.w3.org/TR/xhtml-modularization/>
- XML wird vervollständigt und stärker modularisiert.

Einige Etappen:

- 14.1.1999 *Namespaces in XML* (Recommendation),  
<http://www.w3.org/TR/REC-xml-names>
- 16.11.1999 *XSL Transformations (XSLT) Version 1.0* (Recommendation),  
<http://www.w3.org/TR/xslt>
- 16.11.1999 *XML Path Language (XPath) Version 1.0* (Recommendation),  
<http://www.w3.org/TR/xpath>
- 6.10.2000 *Extensible Markup Language (XML) 1.0 (Second Edition)* (Recommendation), <http://www.w3.org/TR/REC-xml>
- 13.11.2000 *Document Object Model (DOM) Level 2 Specifications* (Recommendation) mit 5 Dokumenten
- 21.11.2000 *Extensible Stylesheet Language (XSL) Version 1.0* (Candidate Recommendation), <http://www.w3.org/TR/xsl/>
- 20.12.2000 *XML Linking Language (XLink) Version 1.0* (Proposed Recommendation), <http://www.w3.org/TR/xlink/>

- 20.12.2000 *XML Base* (Proposed Recommendation),  
<http://www.w3.org/TR/xmlbase/>
- 8.1.2001 *XML Pointer Language (XPointer) Version 1.0* (Working Draft),  
<http://www.w3.org/TR/xptr>
- 19.1.2001 *Canonical XML Version 1.0* (Proposed Recommendation),  
<http://www.w3.org/TR/xml-c14n>
- 2.2.2001 *XML Information Set* (Working Draft),  
<http://www.w3.org/TR/xml-infoset>

## 7 Zwei kleine XML-Beispiele

### 7.1 Gedichtsammlung

Eine Gedichtsammlung, die z.Zt. nur aus einem Gedicht besteht, wird [wohlgeformt](#) mit XML beschrieben.

#### 7.1.1 Erstellen der XML-Datei

Eine XML-Datei lässt sich mit einem einfachen Texteditor erstellen und bearbeiten:

```
<sammlung>
  <gedicht>
    <kopf>
      <titel>The SICK ROSE</titel>
      <autor>Unknown Poet</autor>
      <jahr/>
    </kopf>
    <strophe>
      <zeile>O Rose thou art sick.</zeile>
      <zeile>The invisible worm,</zeile>
      <zeile>That flies in the night</zeile>
      <zeile>In the howling storm:</zeile>
    </strophe>
    <strophe>
      <zeile>Has found out thy bed</zeile>
      <zeile>Of crimson joy:</zeile>
      <zeile>And his dark secret love</zeile>
      <zeile>Does thy life destroy.</zeile>
    </strophe>
    <strophe>
      <zeile>Has found out thy bed</zeile>
      <zeile>Of crimson joy:</zeile>
      <zeile>And his dark secret love</zeile>
      <zeile>Does thy life destroy.</zeile>
    </strophe>
  </gedicht>
</sammlung>
```

Dieser Darstellung liegt eine vorher entwickelte [DTD](#) zugrunde:

```
<!ELEMENT sammlung (gedicht*) >
<!ELEMENT gedicht (kopf?, strophe+) >
<!ELEMENT kopf (titel?, autor?, jahr?) >
<!ELEMENT titel (#PCDATA) >
<!ELEMENT autor (#PCDATA) >
<!ELEMENT jahr (#PCDATA) >
<!ELEMENT strophe (zeile+) >
<!ELEMENT zeile (#PCDATA) >
```

Beide Teile lassen sich zu der vollständigen XML-Datei `gedicht0.xml` zusammenfassen:

```
<?xml version="1.0"?>
<!-- gedicht0.xml -->
<!DOCTYPE sammlung [
<!ELEMENT sammlung (gedicht*) >
...
]>
<sammlung>
  <gedicht>
...
  </gedicht>
</sammlung>
```

### 7.1.2 Darstellen der XML-Datei

Auf nicht-XML-fähigen Browsern (z.B. Netscape 4.7, Internet Explorer 4) lässt sich diese XML-Datei nur unzureichend darstellen. Im Internet Explorer 5 wird wenigstens die Struktur der XML-Datei dargestellt:

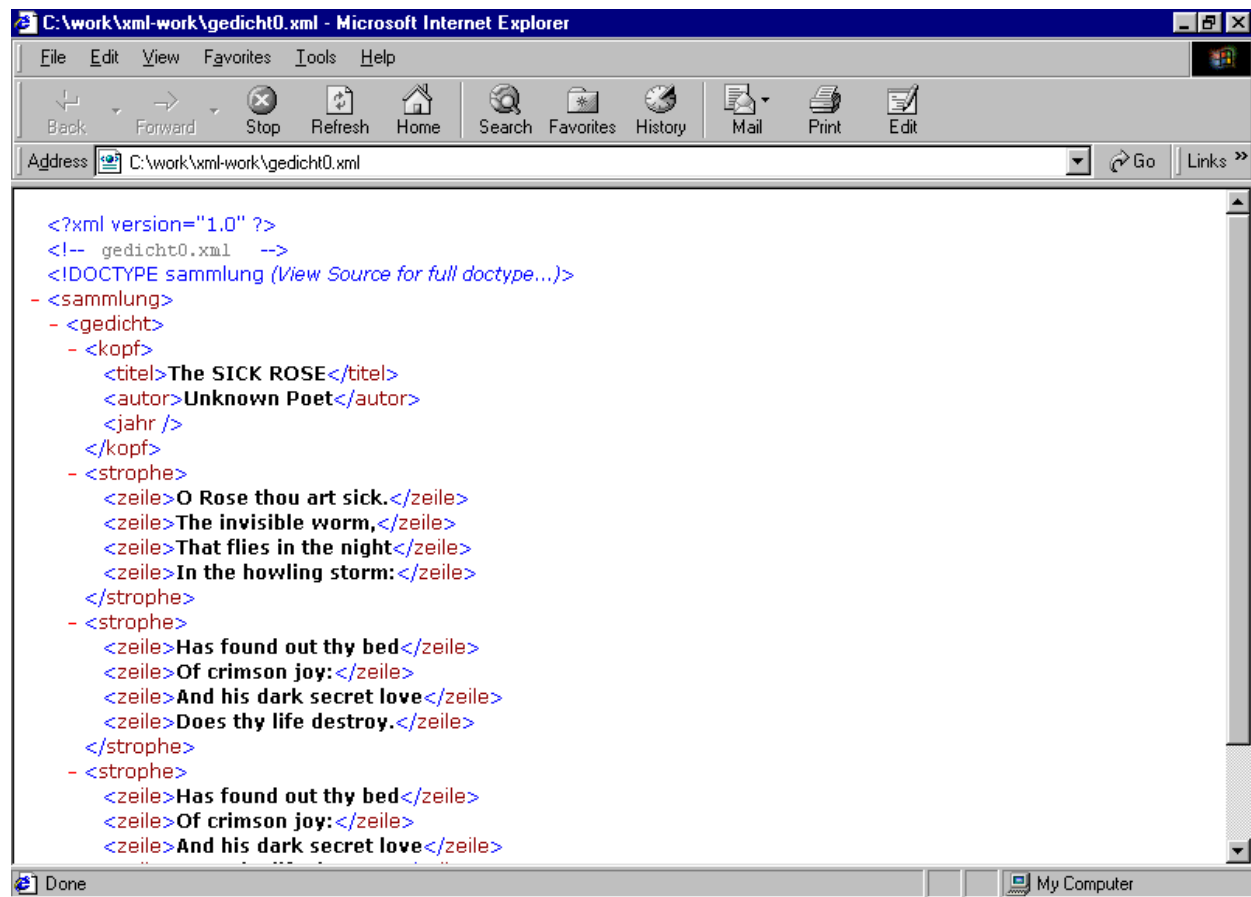


Abbildung 1: Darstellung der XML-Datei `gedicht0.xml` im Internet Explorer mittels des Default-Style-Sheets

Entwickelt man für diese XML-Datei die dazu passende [CSS-Datei](#) `gedicht.css`:

```

/* gedicht.css */
sammlung { color:#CC0000;
           font-size:110%; }
gedicht  { display:block;
           margin-top:2.0ex; }
titel    { display:block;
           font-weight:bold;
           margin-bottom:0.5ex; }
autor    { display:block;
           font-size: 80%; }
strophe  { display:block;
           margin-top:1.0ex;
           margin-left:0.5cm; }
zeile    { display:block; }

```

und ergänzt die XML-Datei entsprechend:



```
<?xml version="1.0"?>
<!-- gedicht.xml -->
<?xml-stylesheet href="gedicht.css" type="text/css" ?>
<!DOCTYPE sammlung [
<!ELEMENT sammlung (gedicht*) >
...
]>
<sammlung>
...
</sammlung>
```

so erhält man in XML-fähigen Browsern beispielsweise die folgende Darstellung:

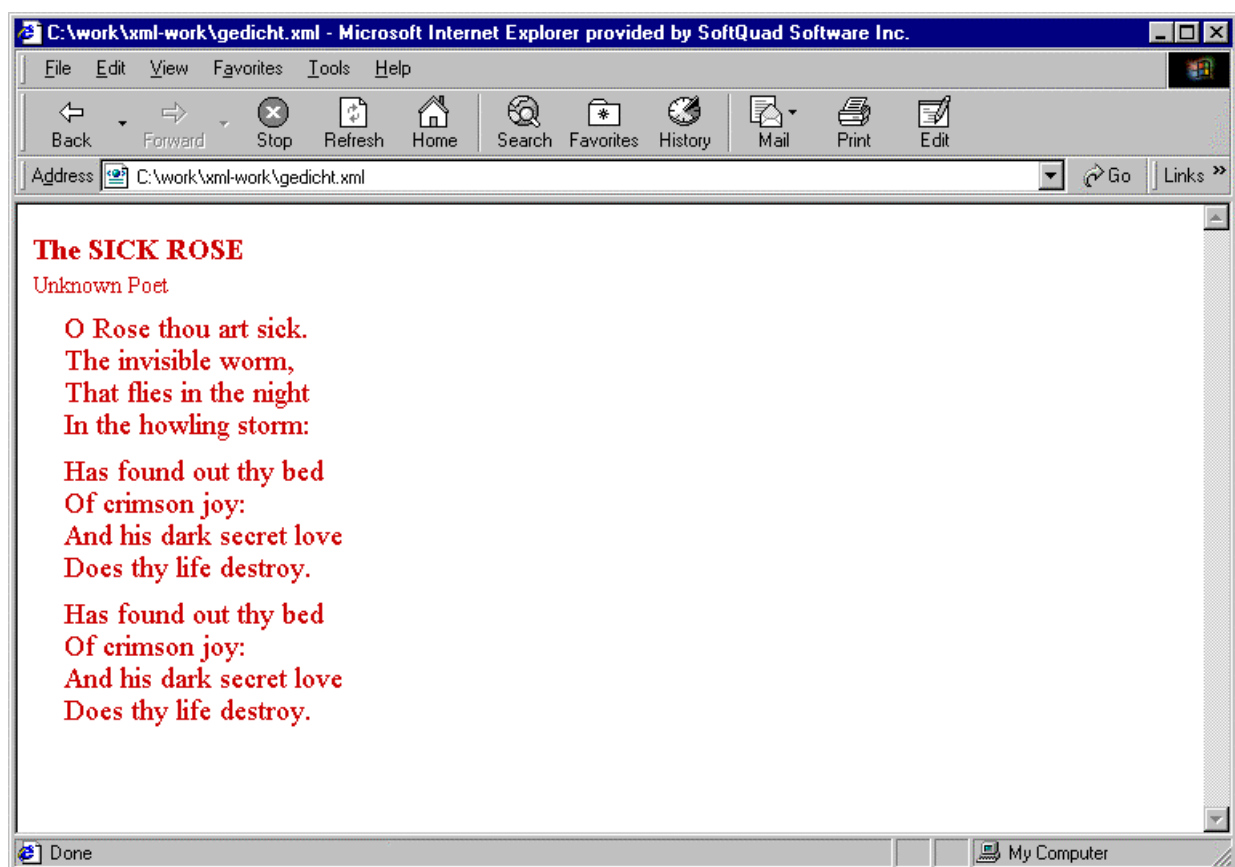
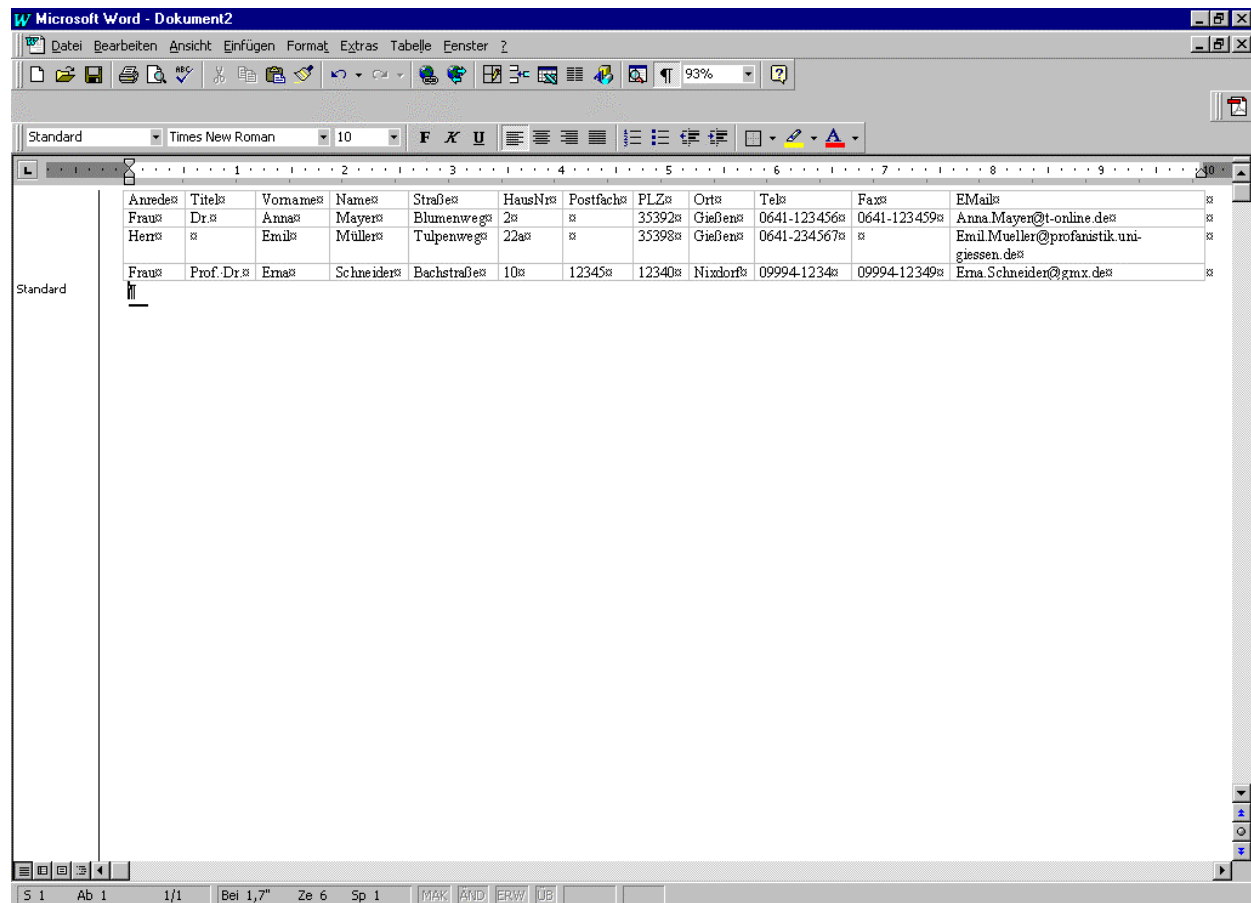


Abbildung 2: Darstellung der XML-Datei `gedicht.xml` im Internet Explorer mittels der CSS-Datei `gedicht.css`

## 7.2 Tabelle

### 7.2.1 Erstellen der XML-Datei

Ausgangspunkt könnte eine WinWord-Tabelle sein:



The screenshot shows the Microsoft Word interface with a table containing contact data. The table has 12 columns: Anrede, Titel, Vorname, Name, Straße, HausNr, Postfach, PLZ, Ort, Tel, Fax, and EMail. There are three rows of data. The status bar at the bottom indicates 'S 1', 'Ab 1', '1/1', 'Bei 1,7"', 'Ze 6', 'Sp 1', and a list of fields: MAX, AND, ERW, UB.

Anrede	Titel	Vorname	Name	Straße	HausNr	Postfach	PLZ	Ort	Tel	Fax	EMail
Frau	Dr.	Anna	Mayer	Blumenweg	2a		35392	Gießen	0641-123456	0641-123459	Anna.Mayer@t-online.de
Herr		Emil	Müller	Tulpenweg	22a		35398	Gießen	0641-234567		Emil.Mueller@profanistik.uni-giessen.de
Frau	Prof. Dr.	Ema	Schneider	Bachstraße	10	12345	12340	Nixdorf	09994-1234	09994-12349	Ema.Schneider@gmx.de

Abbildung 3: Ausgangstabelle in WinWord

oder eine daraus abgeleitete ASCII-Rohfassung (adr.txt):

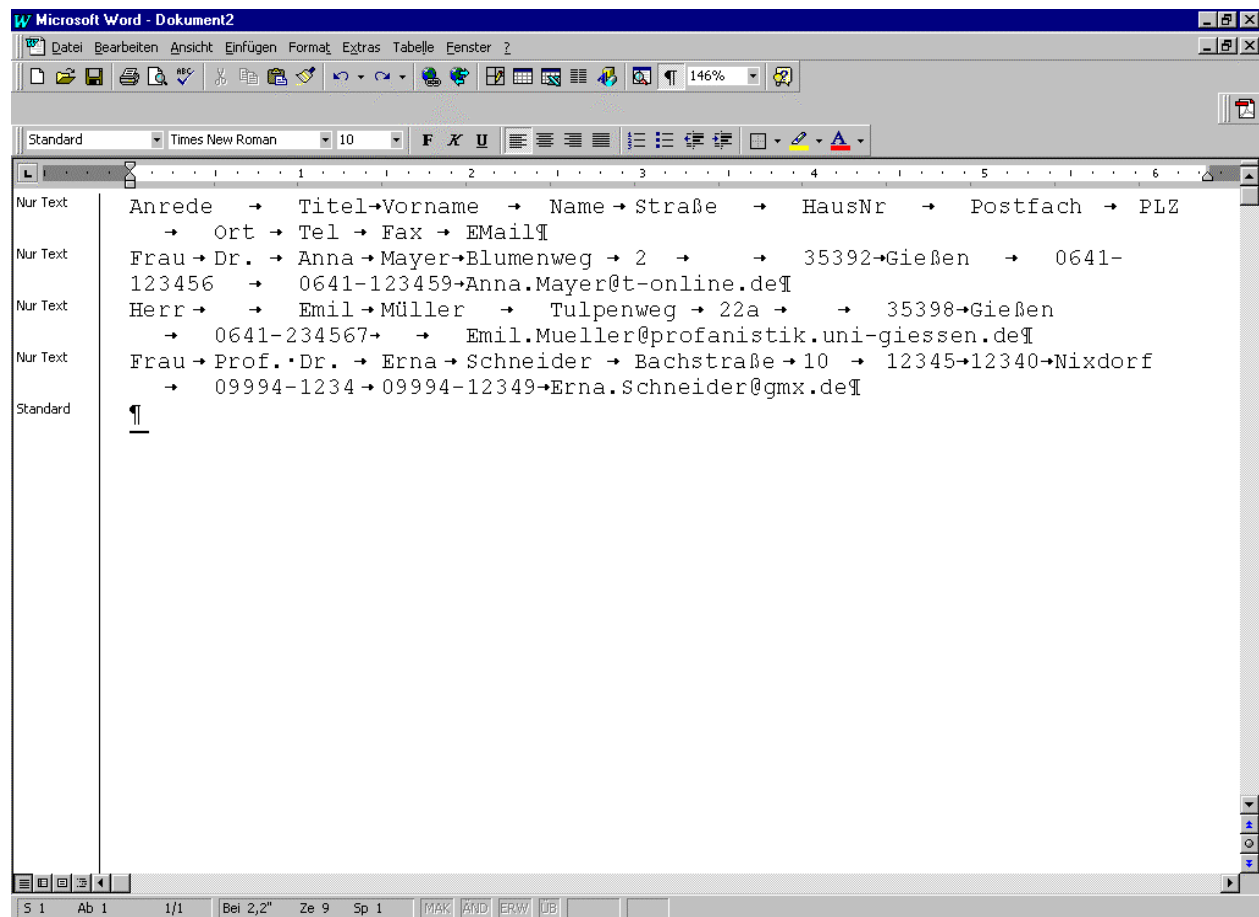


Abbildung 4: ASCII-Rohfassung `adr.txt` der Tabelle in WinWord

Diese Datei lässt sich mit geringem Aufwand mittels des spezialisierten AWK-Programms `xml-adr.awk` in eine XML-Datei (`adr.xml`) umwandeln:

```
gawk -f xml-adr.awk adr.txt > adr.xml
```

mit `adr.xml`:

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE adressen [
<!ELEMENT adressen (adresse*) >
<!ELEMENT adresse (anrede?, titel?,
                    name, strasse?, postfach?,
<!ATTLIST adresse nr CDATA #REQUIRED >
<!ELEMENT titel (#PCDATA) >
<!ELEMENT anrede (#PCDATA) >
<!ELEMENT name (vorname?, nachname) >
<!ELEMENT vorname (#PCDATA) >
<!ELEMENT nachname (#PCDATA) >
<!ELEMENT strasse (sname?, hnr?) >
<!ELEMENT sname (#PCDATA) >
<!ELEMENT hnr (#PCDATA) >

```

```
<!ELEMENT postfach (#PCDATA) >
<!ELEMENT ort (plz?,ortsname?) >
<!ELEMENT plz (#PCDATA) >
<!ELEMENT ortsname (#PCDATA) >
<!ELEMENT telefon (#PCDATA) >
<!ELEMENT fax (#PCDATA) >
<!ELEMENT email (#PCDATA) >
]>
<adressen>
<adresse nr="1">
  <anrede>Frau</anrede>
  <titel>Dr.</titel>
  <name>
    <vorname>Anna</vorname>
    <nachname>Mayer</nachname>
  </name>
  <strasse>
    <sname>Blumenweg</sname>
    <hnr>2</hnr>
  </strasse>
  <ort>
    <plz>35392</plz>
    <ortsname>Gießen</ortsname>
  </ort>
  <telefon>0641-123456</telefon>
  <fax>0641-123459</fax>
  <email>Anna.Mayer@t-online.de</email>
</adresse>
<adresse nr="2">
  ...
</adresse>
</adressen>
```

oder alternativ als XML-Datei adr2.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- adr2.xml -->
<!DOCTYPE adressen SYSTEM "adressen.dtd" >
<adressen>
  ...
</adressen>
```

mit externer [DTD](#) adressen.dtd:

```
<!-- adressen.dtd -->
<!ELEMENT adressen (adresse*) >
<!ELEMENT adresse (anrede?, titel?, name, strasse?,
                  postfach?, ort?, telefon?, fax?,
                  email?) >
<!ATTLIST adresse nr CDATA #REQUIRED >
<!ELEMENT titel (#PCDATA) >
<!ELEMENT anrede (#PCDATA) >
<!ELEMENT name (vorname?, nachname) >
<!ELEMENT vorname (#PCDATA) >
<!ELEMENT nachname (#PCDATA) >
<!ELEMENT strasse (sname?, hnr?) >
<!ELEMENT sname (#PCDATA) >
<!ELEMENT hnr (#PCDATA) >
<!ELEMENT postfach (#PCDATA) >
<!ELEMENT ort (plz?, ortsname?) >
<!ELEMENT plz (#PCDATA) >
<!ELEMENT ortsname (#PCDATA) >
<!ELEMENT telefon (#PCDATA) >
<!ELEMENT fax (#PCDATA) >
<!ELEMENT email (#PCDATA) >
```

### 7.2.2 Darstellen der XML-Datei

Diese XML-Datei `adr2.xml` lässt sich zwar nicht im Internet Explorer 4 oder Netscape 4.7 darstellen, wohl aber ihre Struktur ohne weiteren Aufwand im Internet Explorer 5 oder XML Notepad:

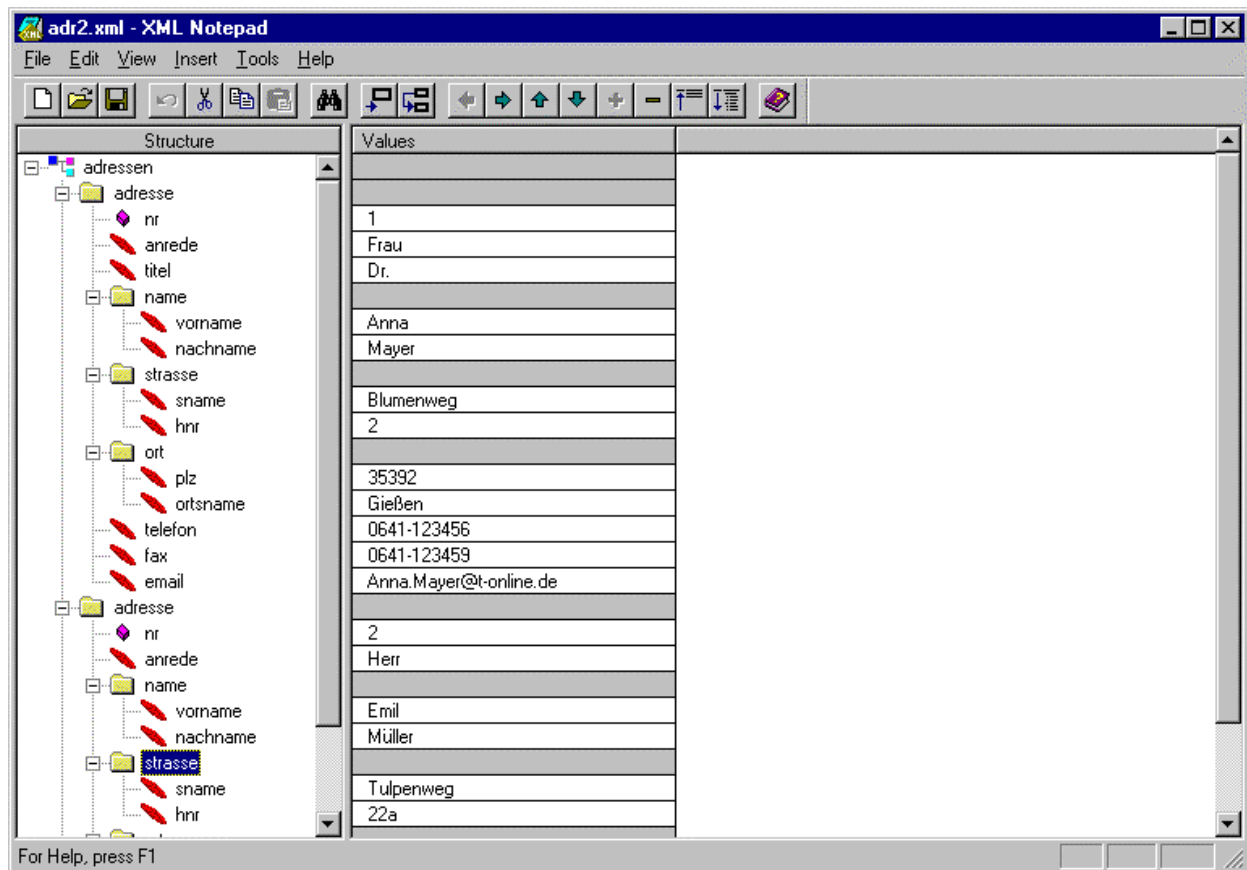


Abbildung 5: Darstellung der XML-Datei adr2.xml im XML Notepad

### 7.2.3 Umformen der XML-Datei durch ein spezialisiertes Programm

Die XML-Datei adr2.xml kann durch ein spezialisiertes Programm, wie z.B.

```
gawk -f pr0.awk adr2.xml | gawk -f pr1.awk >adr.html
```

in eine HTML-Datei (adr.html) umgesetzt werden:

```
<html>
<head><title>Adressen</title></head>
<body>
<h1>Adressen</h1>

<table border=1>
<tr><!-- Tabellenkopf -->
  <td><b>Nr.</b></td>
  <td><b>Anrede</b></td>
  <td><b>Titel</b></td>
  <td><b>Vorname</b></td>
  <td><b>Nachname</b></td>
  <td><b>PLZ</b></td>
  <td><b>Ortsname</b></td>
```

```

    <td><b>Straßenname</b></td>
    <td><b>Hausnummer</b></td>
    <td><b>Postfach</b></td>
    <td><b>Telefon</b></td>
    <td><b>Fax</b></td>
    <td><b>EMail</b></td>
  </tr>
<tr> <!-- Tabelleneintrag 1 -->
  <td>1</td>
  <td><anrede>Frau</anrede></td>
  <td><titel>Dr.</titel></td>
  <td><vorname>Anna</vorname></td>
  <td><nachname>Mayer</nachname></td>
  <td><plz>35392</plz></td>
  <td><ortsname>Gießen</ortsname></td>
  <td><sname>Blumenweg</sname></td>
  <td><hnr>2</hnr></td>
  ...
</tr>
<tr> <!-- Tabelleneintrag 2 -->
  ...
<tr> <!-- Tabelleneintrag 3 -->
  ...
</table>
</body>
</html>

```

### 7.2.4 Darstellen der XML-Datei mit Hilfe einer CSS-Datei und eines Browsers

Entwickelt man für die XML-Datei adr2.xml die [spezielle CSS-Datei](#) adr.css

```

/* adr.css */
postfach:before { content:"Postfach "; }
telefon:before  { content:"Tel.: "; }
fax:before      { content:"Fax: "; }
email:before    { content:"E-Mail: "; }
adressen:before { content:"Adressen "; }
adressen:after  { content:"-----"; }
adressen        { color:#CC0000;
                  font-size:130%; }
adresse        { display:block;
                  margin-top:1.0ex;
                  font-family:sans-serif; }

```

```
/* keine Spez. fuer anrede,titel,name
(vorname,nachname) */
ort          { display:block;
              font-weight:bold;
              margin-bottom:0.5ex; }
strasse      { display:block; }
telefon      { display:block; }
fax          { display:block; }
email        { display:block;
              font-family:monospace; }
postfach     { display:block; }
```

und ändert die XML-Datei entsprechend ab (adr3.xml):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- adr3.xml -->
<?xml-stylesheet href="adr.css" type="text/css" ?>
<!DOCTYPE adressen SYSTEM "adressen.dtd" >
<adressen>
...
</adressen>
```

so erhält man bei Mozilla und Netscape 6 die folgende Darstellung:



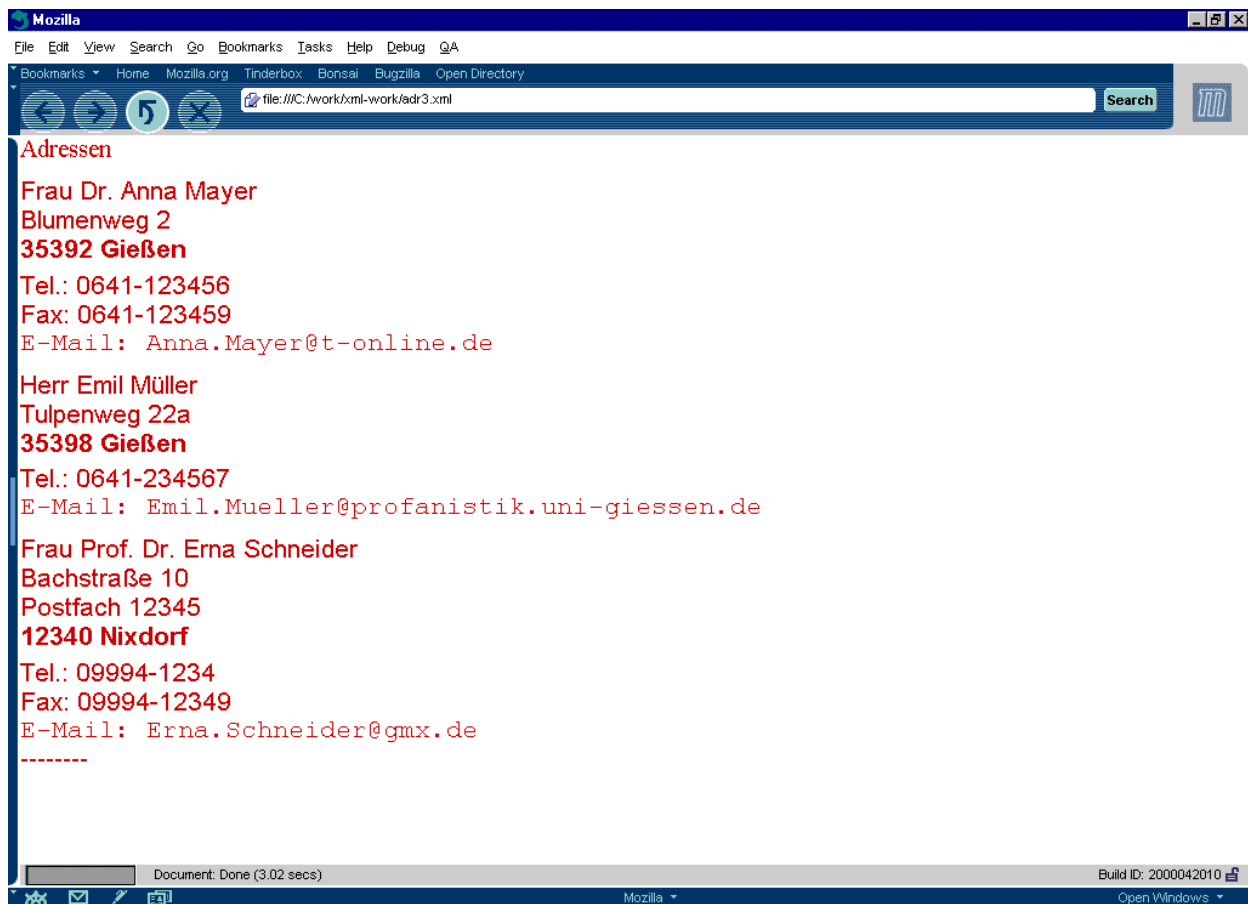


Abbildung 6: Darstellung der XML-Datei `adr3.xml` im Mozilla mittels der CSS-Datei `adr.css`

## 7.2.5 Transformation in eine HTML-Datei mit Hilfe von XSL und eines Browsers

Weitaus mehr Möglichkeiten bietet die Verbindung der XML-Datei (`adr5.xml`) mit einer [speziellen XSL-Datei](#):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- adr5.xml -->
<?xml-stylesheet href="adr.xsl" type="text/xsl" ?>
<!DOCTYPE adressen SYSTEM "adressen.dtd" >
<adressen>
...
</adressen>
```

mit `adr.xsl`:

```
<?xml version='1.0'?>
<!-- adr.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <html>
      <head><title>Adressen</title></head>

      <body>"
      <h1>Adressen</h1>

      <xsl:for-each select="adressen/adresse">
        <p>
          <xsl:value-of select="anrede" />
          <xsl:value-of select="titel"/>
          <xsl:value-of select="name/vorname"/>
          <xsl:value-of select="name/nachname"/><br />
          <xsl:value-of select="strasse/sname"/>
          <xsl:value-of select="strasse/hnr"/><br />
          <xsl:if test="postfach">
            Postfach: <xsl:value-of select="postfach"/><br />
          </xsl:if>
          <xsl:value-of select="ort/plz"/>
          <xsl:value-of select="ort/ortsname"/><br />
          <xsl:if test="telefon">
            Telefon: <xsl:value-of select="telefon"/><br />
          </xsl:if>
          <xsl:if test="fax">
            Fax: <xsl:value-of select="fax"/><br />
          </xsl:if>
          E-Mail: <tt><xsl:value-of select="email"/></tt>
        </p>
      </xsl:for-each>

    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

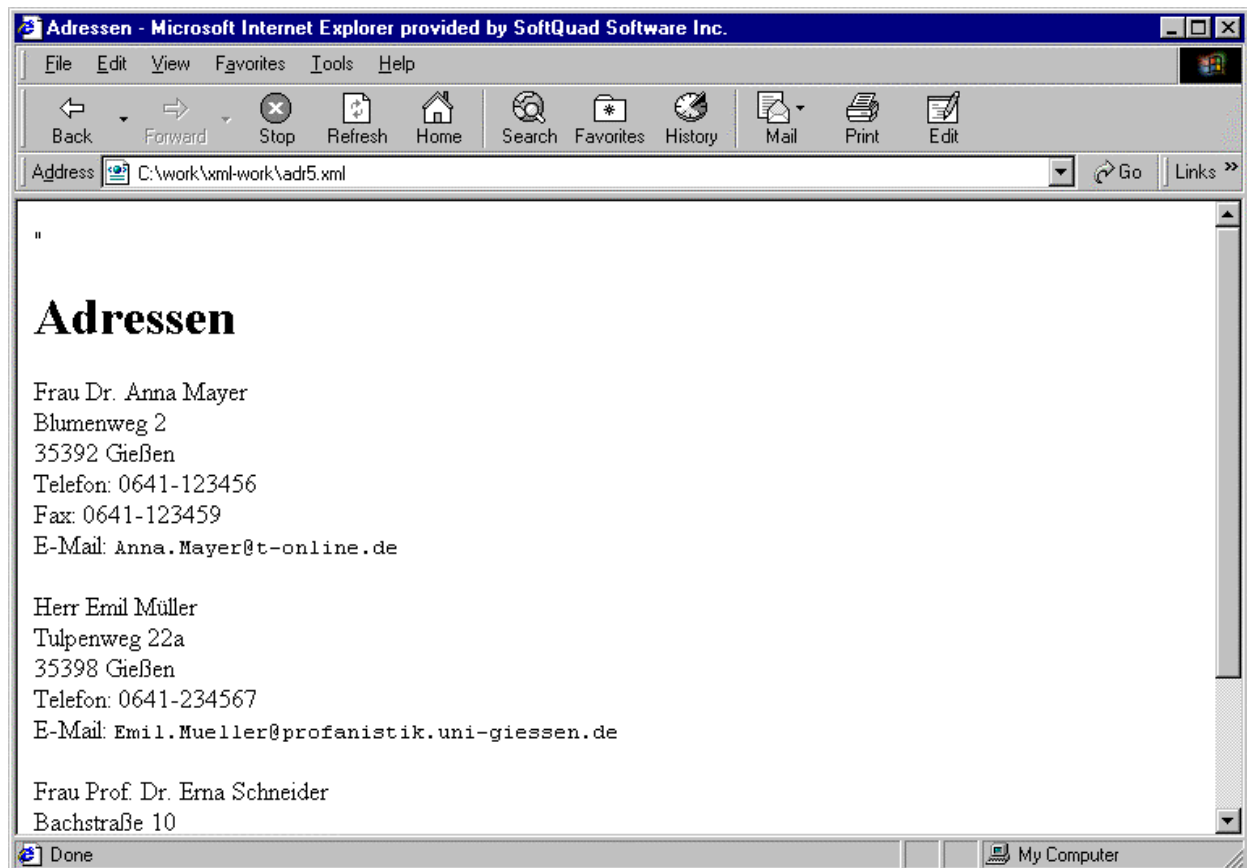


Abbildung 7: Darstellung der XML-Datei `adr5.xml` im Internet Explorer mittels der XSL-Datei `adr.xsl`

## 7.2.6 Transformation der XML-Datei in eine HTML-Datei (mit Sortierung)

Mit Hilfe der [XSL-Datei](#) `adr2.xsl`

```
<?xml version='1.0' encoding="ISO-8859-1" ?>
<!-- adr2.xsl -->
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <html>
      <head><title>Adressen</title></head>

      <body>
        <h1>Adressen</h1>

        <table border="1">
          <tr>
            <td><b>Nr.</b></td>
            <td><b>Anrede</b></td>
            <td><b>Titel</b></td>
```

```
<td><b>Vorname</b></td>
<td><b>Name</b></td>
<td><b>Straße</b></td>
<td><b>Nr.</b></td>
<td><b>Postfach</b></td>
<td><b>PLZ</b></td>
<td><b>Ort</b></td>
<td><b>Telefon</b></td>
<td><b>Fax</b></td>
<td><b>E-Mail</b></td>
</tr>

<xsl:for-each select="adressen/adresse"
order-by="ort/plz">
  <tr>
    <td><xsl:value-of select="@nr" /></td>
    <td><xsl:value-of select="anrede" /></td>
    <td><xsl:value-of select="titel"/></td>
    <td><xsl:value-of select="name/vorname"/></td>
    <td><xsl:value-of select="name/nachname"/></td>
    <td><xsl:value-of select="strasse/sname"/></td>
    <td><xsl:value-of select="strasse/hnr"/></td>
    <td><xsl:value-of select="postfach"/></td>
    <td><xsl:value-of select="ort/plz"/></td>
    <td><xsl:value-of select="ort/ortsname"/></td>
    <td><xsl:value-of select="telefon"/></td>
    <td><xsl:value-of select="fax"/></td>
    <td><TT><xsl:value-of select="email"/></TT></td>
  </tr>
</xsl:for-each>

</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

lässt sich die XML-Datei adr6.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- adr6.xml -->
<?xml-stylesheet href="adr2.xsl" type="text/xsl" ?>
<!DOCTYPE adressen SYSTEM "adressen.dtd" >
<adressen>
...
</adressen>
```

in einer sortierten Tabelle ausgeben:

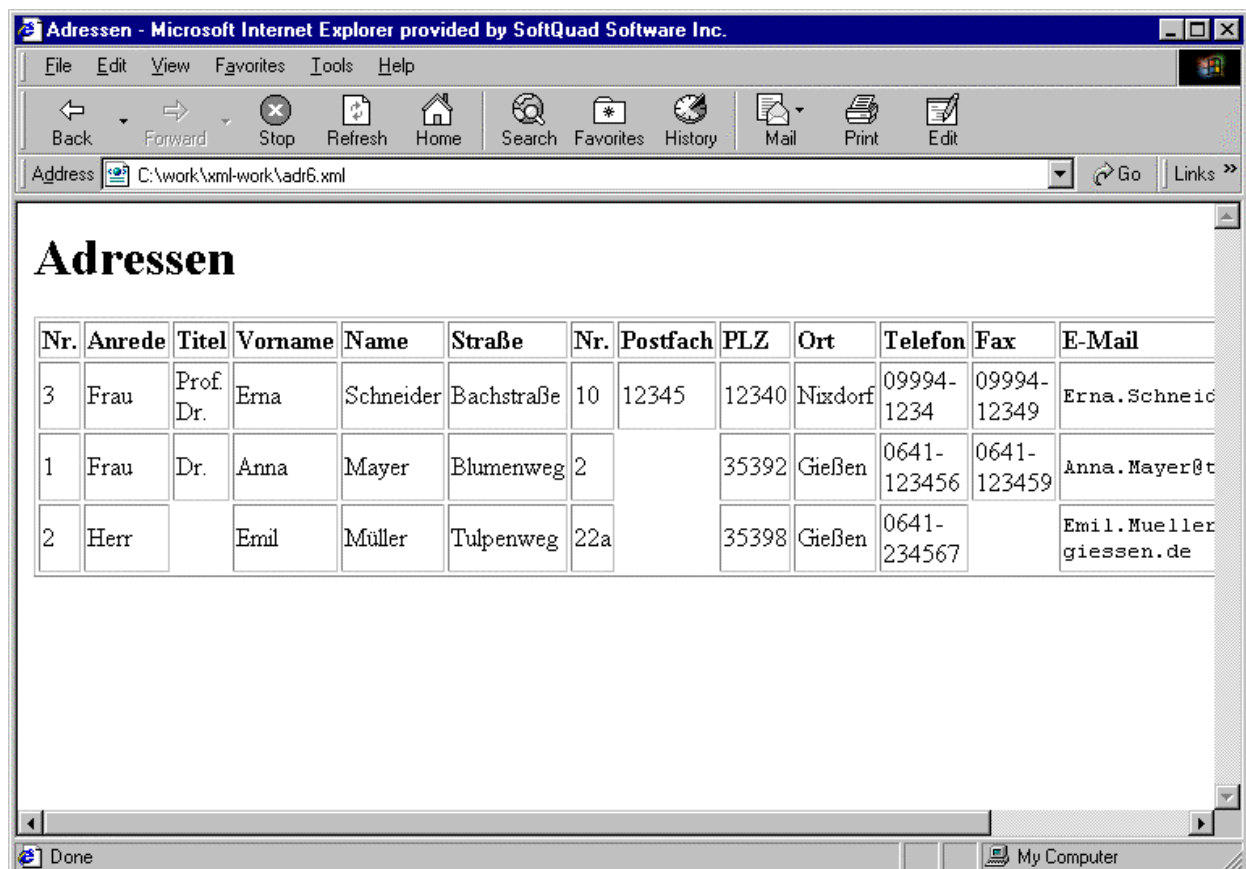


Abbildung 8: Darstellung der XML-Datei `adr6.xml` im Internet Explorer mittels der XSL-Datei `adr2.xsl`

## 7.2.7 Transformation der XML-Datei in eine LaTeX-Datei

Mit Hilfe des XSLT-Prozessors **saxon** wird das XML-Dokument `adr5.xml` in die LaTeX-Datei `adr-tex.tex` transferiert. Gesteuert wird dabei der Vorgang durch die XSL-Style-Datei `adr-tex.xsl` (hier mit zusätzlichen Einrückungen und Leerzeichen):

```
<?xml version='1.0' encoding="ISO-8859-1" ?>
<!-- adr-tex.xsl -->
<!-- XSL-Datei (passend zu adr5.xml -->
<!-- kann mit SAXON verarbeitet werden -->
<!-- 28.2.2001, GP (HRZ Gießen) -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="ISO-8859-1" />
  <xsl:template match="/">
    \documentclass[10pt,a4paper]{article}
    \usepackage{adr-tex}

    \title{Adressen}
    \author{Emil Mayer}
    \date{\today}

    \begin{document}
    \maketitle
    \section*{Adressen}

    <xsl:for-each select="adressen/adresse">
      \begin{adresse}{<xsl:value-of select="@nr" />}
      <xsl:text> </xsl:text>
      \anrede{<xsl:value-of select="anrede" />}
      <xsl:text> </xsl:text>
      <xsl:if test="titel">
        \titel{<xsl:value-of select="titel"/>}
        <xsl:text> </xsl:text>
      </xsl:if>
      \vorname{<xsl:value-of select="name/vorname"/>}
      <xsl:text> </xsl:text>
      \nachname{<xsl:value-of select="name/nachname"/>}
      <xsl:text> </xsl:text>
      \sname{<xsl:value-of select="strasse/sname"/>}
      <xsl:text> </xsl:text>
      \hnr{<xsl:value-of select="strasse/hnr"/>}
      <xsl:text> </xsl:text>
      <xsl:if test="postfach">
        \postfach{<xsl:value-of select="postfach"/>}
        <xsl:text> </xsl:text>
      </xsl:if>
      \plz{<xsl:value-of select="ort/plz"/>}
      <xsl:text> </xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

```

\ortsname{<xsl:value-of select="ort/ortsname"/>}
<xsl:text> </xsl:text>
<xsl:if test="telefon">
  \telefon{<xsl:value-of select="telefon"/>}
  <xsl:text> </xsl:text>
</xsl:if>
<xsl:if test="fax">
  \fax{<xsl:value-of select="fax"/>}
  <xsl:text> </xsl:text>
</xsl:if>
<xsl:if test="email">
  \email{<xsl:value-of select="email"/>}
  <xsl:text> </xsl:text>
</xsl:if>
\end{adresse}
</xsl:for-each>

\end{document}
</xsl:template>
</xsl:stylesheet>

```

### Der Aufruf

```
saxon adr5.xml adr-tex.xsl > adr-tex.tex
```

führt zur LaTeX-Datei adr-tex.tex:

```

\documentclass[10pt,a4paper]{article}
\usepackage{adr-tex}

\title{Adressen}
\author{Emil Mayer}
\date{\today}

\begin{document}
\maketitle
\section*{Adressen}

\begin{adresse}{1}
\anrede{Frau}
\titel{Dr.}
\vorname{Anna}
\nachname{Mayer}
\sname{Blumenweg}
\hnr{2}

```

```
\plz{35392}
\ortsname{Gießen}
  \telefon{0641-123456}
  \fax{0641-123459}
  \email{Anna.Mayer@t-online.de}
\end{adresse}

\begin{adresse}{2}
...
\end{adresse}

\begin{adresse}{3}
...
\end{adresse}

\end{document}
```

Diese LaTeX-Datei wird mittels des Aufrufs

```
latex adr-tex.tex
```

weiter verarbeitet; die Ergebnisdatei `adr5.dvi` kann durch einen DVI-Previewer am Bildschirm betrachtet werden:



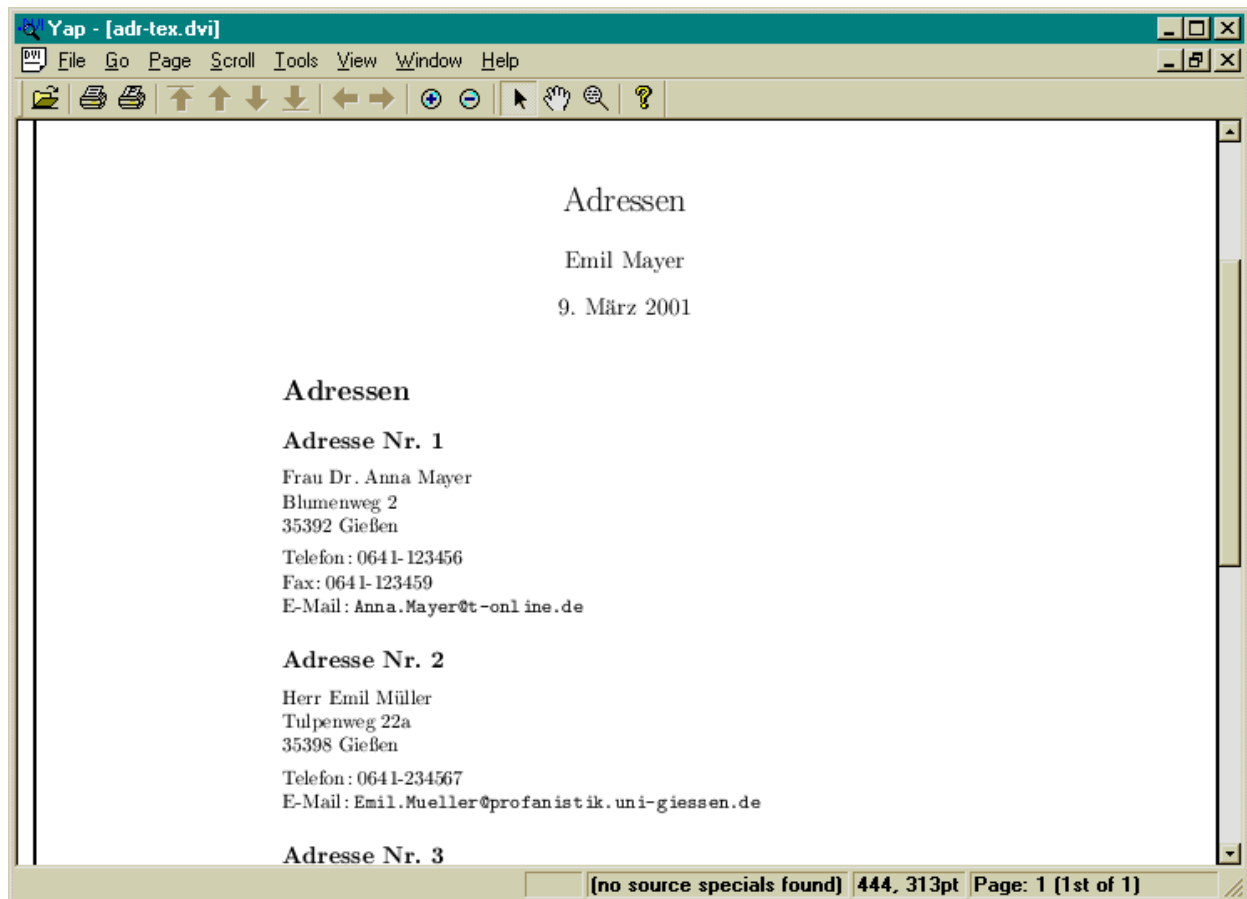


Abbildung 9: Darstellung der DVI-Datei `adr-tex.dvi` im DVI-Previewer Yap

Die konkrete Formatierung wird dabei durch das LaTeX-Package `adr-tex.sty` festgelegt:

```
% adr-tex.sty
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{adr-tex}[2001/02/28 adr-tex.sty]
\immediate\write16{This is adr-tex.sty, Version 1.00
<2001/02/28>}

\RequirePackage[latin1]{inputenc}
\RequirePackage{ngerman}

\newenvironment{adresse}[1]{%
    {\subsection*{Adresse~Nr.~#1}}{}
\newcommand{\anrede}[1]{#1}
\newcommand{\titel}[1]{#1}
\newcommand{\vorname}[1]{#1}
\newcommand{\nachname}[1]{#1\\}
\newcommand{\sname}[1]{#1}
\newcommand{\hnr}[1]{#1\\}
\newcommand{\postfach}[1]{Postfach:~#1\\}
\newcommand{\plz}[1]{#1}
```

```
\newcommand{\ortsname}[1]{#1\\[1.0ex]}  
\newcommand{\telefon}[1]{Telefon:~#1\\}  
\newcommand{\fax}[1]{Fax:~#1\\}  
\newcommand{\email}[1]{E-Mail:~\texttt{#1}}  
\endinput
```