
SuSE PRESS

Dr. Helmut Herold

Das HTML/XHTML-Buch

mit Cascading Style Sheets und einer Einführung in XML

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Herold, Helmut:
Das HTML/XHTML-Buch : mit cascading style sheets und einer Einführung in XML / Helmut Herold. – Nürnberg : SuSE-PRESS, 2002
ISBN 3-935922-52-3

Umschlaggestaltung: Fritz Design GmbH, Erlangen
Gesamtlektorat: Nicolaus Millin
Fachlektorat: Uwe Boris, Michael Eicks, Klaas Freitag, Bjoern Lotz, Arno Seidel, Steve Tomlin, Peter Varkoly
Satz: LaTeX
Druck: Kösel, Kempten
Printed in Germany on acid free paper.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Hier wird zunächst kurz auf die Historie von HTML und seine Standardisierung sowie auf einige andere Aspekte eingegangen, die im Hinblick auf HTML zum Grundverständnis gehören. Zudem werden einige wichtige Webadressen vorgestellt, an denen sich weitere Informationen zu HTML und den hier vorgestellten Aspekten befinden.

1.1 Die Entstehung von HTML

Nachfolgend werden die Meilensteine der verschiedenen HTML-Versionen kurz vorgestellt:

HTML 1.0
Die erste Version von HTML entstand Anfang der 1990er Jahre und war im Vergleich zu den heutigen Versionen noch sehr spartanisch ausgelegt.

HTML 2.0
Ende des Jahres 1995 wurde HTML 2.0 als offizieller Sprachstandard verabschiedet. Da der zu dieser Zeit weit verbreitete Netscape Navigator (Version 2) bereits neue Techniken anbot, wie z. B. einfache Multimedia-Unterstützung oder Mehrfenstertechnik, die in diesem HTML-Standard nicht vorhanden waren, nahm man kaum Rücksicht auf diesen Standard. Statt dessen orientierte man sich mehr an die vom Netscape Navigator unterstützten HTML-Sprachelemente.

HTML 3.2
Diese Version wurde Anfang 1997 zum offiziellen HTML-Standard und war aus heutiger Sicht ein kleines Fiasko, da man sich zu diesem Zeitpunkt nicht klar war, welchen Weg HTML gehen sollte. Mit dieser Version wurde versucht, HTML als Design-Sprache zu etablieren, was sich als Fehlschlag entpuppte. Viele Sprachelemente, die in HTML 3.2 eingeführt wurden, gelten mittlerweile wieder als veraltet und werden sehr wahrscheinlich in neueren HTML-Versionen auch wieder entfernt werden, da sie mittlerweile durch an-

dere und bessere Ansätze verwirklicht wurden, wie z. B. *Cascading Style Sheets* (siehe Kapitel 17 auf Seite 279).

HTML 4.0 und **HTML 4.01**
HTML 4.0 wurde Anfang 1998 als offizieller HTML-Standard verabschiedet. Zwischenzeitlich gilt HTML 4.01, eine etwas überarbeitete Version zu HTML 4.0, als offizieller HTML-Standard. Diese beiden Versionen standardisierten das Einbinden von *Cascading Style Sheets* und Skript-Sprachen (wie z. B. JavaScript) in HTML-Dokumente. Zudem konzentrierten sich diese beiden Versionen auf die Internationalisierung von HTML, um HTML-Dokumente in allen möglichen Sprachen für die unterschiedlichsten Kulturkreise dieser Welt zu entwerfen.

1.2 Die Browser-Problematik

Browser sind Programme, wie z. B. der Netscape Navigator, der Internet-Explorer usw. Solche Browser sind für eine entsprechende Darstellung der vorgelegten HTML-Dokumente am lokalen Rechner zuständig. Die Darstellung von HTML-Dokumenten liegt ausschließlich in der Verantwortung des eingesetzten Browsers.
Nun könnte man denken, dass mit der Existenz von HTML-Standards jeder Browser sich bei Vorlage des gleichen HTML-Dokuments gleich verhält. Leider ist das nicht so. Wie bereits im vorherigen Abschnitt erwähnt, war der Netscape Navigator in seiner Version 2 dem damaligen HTML-Standard 2.0 technisch weit voraus, was dazu führte, dass HTML-Dokumente entworfen wurden, die zwar vom Netscape Navigator 2.0 dargestellt werden konnten, aber nicht dem damaligen HTML-Standard 2.0 entsprachen.
Nun, diese Situation hat sich bis heute noch nicht grundlegend geändert. Gerade mit dem Aufkommen des Internet-Explorers der Firma Microsoft erwuchs dem Netscape Navigator ein ernst zu nehmender Konkurrent. Diese beiden Browser bekriegen sich bis heute, wobei jeder dieser beiden immer wieder neue HTML-Konstrukte einführte, die er dann nachträglich als HTML-Standard durchzusetzen versuchte.
Man sollte also grundsätzlich niemals Browser-spezifische HTML-Konstrukte verwenden, wenn man sicherstellen will, dass ein HTML-Dokument von jedem Browser so dargestellt wird, wie man sich das vorstellt. Da nicht alle im heutigen HTML-Standard enthaltenen Konstrukte und Vorgaben auch von jedem Browser unterstützt werden, liegt die Problematik sogar noch tiefer. Um sicherzustellen, dass eigene HTML-Dokumente auch wirklich in der gewünschten Darstellung bei der Vielzahl von Internet-Benutzern angezeigt werden, ist es empfehlenswert, sich eigene HTML-Dokumente mit mehreren Browsern anzeigen zu lassen, bevor man sie ins Internet stellt.
Diese Situation war und ist absolut unbefiedigend, weshalb man schon sehr früh erkannte, dass diesem Wildwuchs von Browser-spezifischen HTML-Konstrukten Einhalt geboten werden musste. Aus diesem Grund gründete man bereits Mitte der 1990er Jahre ein Standardisierungs-Komitee, dass diese Browser-Problematik

zumindest schon etwas gelindert hat und für die Zukunft doch eine Besserung erhoffen lässt.

1.3 Das W3-Konsortium (W3C)

Tim Berners-Lee, der geistige Vater von HTTP und HTML, erkannte schon sehr früh, dass ein HTML-Standard geschaffen werden musste. Dazu schuf er im Jahre 1994 mit dem *W3-Konsortium (W3C)* ein offenes Forum für Entwickler aus den unterschiedlichsten Branchen. Im Jahre 1995 traten namhafte Firmen diesem Komitee bei, die dieses Komitee auch finanzieren. Die Aufgabe des W3-Konsortiums (W3C) ist die Standardisierung von Sprachen, wie z. B. HTML, CSS, XML usw.
Während die Standards und Vorgaben dieses Konsortium am Anfang von den Browser-Herstellern nicht allzu ernst genommen wurden, scheint die Bedeutung dieses Komitees in den letzten Jahren doch zu wachsen, und es bleibt zu hoffen, dass die Browser-Hersteller endlich ihre Alleingänge einstellen und sich stattdessen an die vom W3-Konsortium (W3C) vorgegebenen Standards halten.

1.4 Cascading Style Sheets (CSS)

Cascading Style Sheets sind eine unmittelbare Ergänzung zu HTML. Mit Style-Sheets kann man die Formateigenschaften von HTML-Befehlen festlegen. So kann man z. B. mit Hilfe von Style-Sheets vorgeben, dass Überschriften der zweiten Stufe immer eingerückt und mit einer Schriftgröße von 16 Punkt in der Schriftart *Times-Roman* anzuzeigen sind. Wie wir in Kapitel 17 auf Seite 279 sehen werden, bieten Style-Sheets aber noch eine Vielzahl anderer Möglichkeiten,
Es existieren mehrere Sprachen zum Definieren von Style-Sheets. Die bekannteste Sprache ist CSS (*Cascading Style Sheets*), die vom W3-Konsortium, das auch für die Normierung von HTML zuständig ist, vorgeschlagen wird. Zu CSS existieren zwischenzeitlich mehrere Versionen: Version 1.0 (von 1996) und Version 2.0 (1998). Derzeit wird an der dritten Version von CSS beim W3C gearbeitet.
Leider unterstützen nicht alle Browser CSS vollständig. Netscape 4.x interpretiert fast den vollen Sprachumfang von CSS 1.0 und einen Teil der Befehle von CSS 2.0. Netscape 6.0 unterstützt CSS 1.0 und CSS 2.0 nahezu vollständig.
Der Microsoft Internet-Explorer unterstützt CSS 1.0 ab Priogramm-Version 3.0. In der Version 4.0 unterstützt er einen Teil von CSS 2.0 und einige spezielle, von Microsoft eingeführte Style-Sheet-Angaben. Ab Version 5.5 unterstützt der Microsoft Internet-Explorer CSS 1.0 vollständig und größtenteils auch CSS 2.0
Andere Browser unterstützen meist nur CSS 1.0 oder nur Teile von CSS 1.0 und / oder CSS 2.0.

1.5 XML und XHTML

XML (*Extensible Markup Language*) ist nicht nur einfach eine weitere Sprache, sondern ein neues Konzept für die Datenspeicherung und deshalb nicht nur auf das

Internet zugeschnitten. Die Idee zu XML hatte ebenfalls *Tim Berners-Lee*in den Anfangsjahren der Internet-Euphorie, als er erkannte, dass HTML alleine nicht ausreichen würde, die ständig steigenden Ansprüche zu erfüllen. Es wurde offensichtlich, dass ein Standard geschaffen werden musste, in den HTML, das zwischenzeitlich weitverbreitet war, integriert werden konnte.
Im Februar 1998 brachte das W3-Konsortium die erste Empfehlung zu XML (siehe Kapitel 18 auf Seite 411) heraus. Danach bemühte sich das W3-Konsortium, andere vorhandene Metasprachen mit Hilfe von XML zu standardisieren.
Eine der wichtigste Sprachen war dabei natürlich HTML, das man mit Hilfe von XML umdefinieren musste. Daraus resultierte das neue auf XML basierende HTML mit dem Namen **XHTML 1.0** (*Extensible HyperText Markup Language*). Seit dem Jahre 2000 liegt XHTML 1.0 als Empfehlung des W3-Konsortiums vor und hat damit den gleichen verbindlichen Stellenwert wie der HTML 4.0-Standard. Browser, die HTML-4.0-Dokumente darstellen können, können meist auch ohne Probleme XHTML-Dokumente richtig darstellen.
XHTML entspricht weitgehend dem HTML 4.0-Standard. Allerdings gibt es auch – bedingt durch die XML-Vorschriften – kleinere Unterschiede, die im Kapitel 19 auf Seite 451 vorgestellt werden.

1.6 Einige Internet-Adressen

Internet-Adressen zu HTML

`http://www.w3.org/TR/REC-html32.html`

HTML 3.2-Spezifikation des W3-Konsortiums

`http://www.w3.org/TR/html4`

HTML 4.x-Spezifikation des W3-Konsortiums

`http://selfhtml.teamone.de/`

SELFHTML von Stefan Münz. Diese hervorrragende deutschsprachige Online-Dokumentation zu HTML ist mehr als empfehlenswert.

Internet-Adressen zu Browsern

Internet-Adressen des W3-Konsortiums (W3C)

`http://www.w3.org/`

Offizielle Startseite (Homepage) des W3-Konsortiums

`http://www.w3.org/Consortium/Offices/Germany/`

Deutschsprachige Startseite (Homepage) des W3-Konsortiums

Internet-Adressen zu Cascading Style Sheets (CSS)

`http://www.w3.org/TR/REC-CSS1`

CSS 1.0-Spezifikation des W3-Konsortiums

`http://www.w3.org/TR/REC-CSS2`

CSS 2.0-Spezifikation des W3-Konsortiums

Internet-Adressen zu Cascading (CSS)

`http://www.w3.org/TR/REC-xml`

Aktuelle XML-Spezifikation des W3-Konsortiums

`http://www.w3.org/TR/REC-xhtml1`

XHTML 1.0-Spezifikation des W3-Konsortiums

`http://www.w3.org/TR/REC-xhtml11`

XHTML 1.1-Spezifikation des W3-Konsortiums

Internet-Adresse zum Buch

`http://www.susepress.de/de/download/index.html`

Material zum Download für dieses Buch

Kapitel 3

Grundlegendes zu HTML-Dokumenten

3.1 Struktur von HTML-Dokumenten

HTML-Dokumente haben einen ganz klar definierten Aufbau. Sie unterteilen sich in:

- *Head* (Dokumentenkopf)
  Hier befinden sich Anweisungen, die für das ganze Dokument gelten, wie z. B. der Titel des Dokuments.
- *Body* (eigentlicher Inhalt)
  Hier befindet sich die eigentliche Information in Form von HTML-Anweisungen, die der jeweilige Browser interpretieren und entsprechend darstellen muss.

3.2 Struktur von HTML-Befehlen

HTML-Befehle (auch *HTML-Tags* genannt) haben meist die folgende Syntax:

- Sie bestehen – bis auf wenige Ausnahmen – immer aus einem *Starttag* und einem *Endtag*.
- Die Schlüsselwörter von HTML-Befehlen werden in spitzen Klammern (<>) angegeben. Starttags stehen nur mit Befehlswort in den spitzen Klammern, während man den entsprechenden Endtags einen Slash (/) voranstellt. Ein HTML-Tag wird also üblicherweise – wie nachfolgend gezeigt – angegeben:

  ```
  <tagname> ... Text ... </tagname>
  ```

  Das Start- und Endtag schließen also immer den betreffenden Text ein. Bei einigen Tags kann auf die Angabe des Endtags verzichtet werden, da es aus dem Kontext ersichtlich wird, dass hier das Ende des betreffenden Tags vorliegt. Im folgenden wird auf Tags, die keine Angabe eines Endtags erfordern, gesondert hingewiesen.

- Bei HTML-Tags wird nicht zwischen Groß- und Kleinschreibung unterschieden, wobei es jedoch im Hinblick auf XHTML (auf XML basierendes HTML) empfehlenswert ist, die HTML-Tags immer klein zu scheiben.

3.3 Grundlegende HTML-Tags

Einige Tags sind in jedem Dokument von Notwendigkeit, da sie die entsprechenden Abschnitte (Kopf und Inhalt) kennzeichnen.

3.3.1 Das Tag <html>

Das Tag `<html>...</html>` umrahmt das gesamte Dokument und kennzeichnet es somit als HTML-Datei:

```
<html>
 .... HTML-Dokument
</html>
```

3.3.2 Das Tag <head>

Das Tag `<head>...</head>` markiert den Dokumentenkopf mit seinem Start- und Endtag:

```
<html>

<head>
 .... Dokumentenkopf
</head>

 .... Rest der HTML-Datei

</html>
```

Die Anweisungen innerhalb des Dokumentenkopfes beziehen sich immer auf das gesamte Dokument. Im Dokumentenkopf kann man z. B. den Dokumenttitel und allgemeine Information zum Dokument angeben, wie z. B. Name des Autors. Der Dokumententitel muss dabei mit den Tags `<title>...</title>` angegeben werden.

3.3.3 Das Tag <body>

Das Tag `<body>...</body>` umrahmt den eigentlichen Inhalt des Dokuments:

```
<html>

<head>
 .... Dokumentenkopf
</head>

<body>
 .... Eigentlicher Inhalt des Dokuments
```

```
</body>

</html>
```

Im Tag `<body>...</body>` werden der Inhalt des Dokuments und die Tags zur Formatierung und Gestaltung dieses Dokuments angegeben.

3.4 Ein erstes Beispiel und weitere grundlegende HTML-Konzepte

Hier wollen wir nun ein erstes HTML-Dokument erstellen, wie es nachfolgend gezeigt ist:

```
<html>

<head>
<title>Mein erstes HTML-Dokument</title>
</head>

<body>
Dies ist ein einfacher Text, der sich im
Dokumentenkoerper befindet.

1. Dateiname endet mit '.txt'
   Wenn ich dieser Datei, in der sich dieser Text befindet,
   einen Namen gebe, der mit .txt endet, so wird der gesamte Inhalt
   dieser Datei ohne jegliche Interpretation einfach vom Browser angezeigt.

2. Dateiname endet mit '.htm' oder '.html'
   Wenn ich dieser Datei, in der sich dieser Text befindet,
   einen Namen gebe, der mit .htm oder .html endet, so wird nur
   der Dokumentenkoerper angezeigt, wobei der Browser oben auch einen
   Titel anzeigt.
</body>

</html>
```

Wenn wir diese Datei unter dem Namen `htmlerst.txt` speichern und diese Datei dann im Browser laden, erhalten wir ein Erscheinungsbild, wie es in Abbildung 3.1 gezeigt ist. Interessant ist an Abbildung 3.1, dass der Browser den gesamten Inhalt der Datei `htmlerst.txt` – ohne jegliche Interpretation der HTML-Anweisungen – einfach anzeigt. Das liegt daran, dass wir der Datei einen Namen gegeben haben, der mit `.txt` endet. Dateien, die mit der Endung `.txt` enden, werden vom Browser als Textdateien identifiziert, deren Inhalt er nicht auswertet, sondern nur einfach anzeigt.
Wollen wir, dass der Browser eine Datei als HTML-Datei behandelt und die enthaltenen HTML-Anweisungen auswertet, müssen wir einer solchen Datei immer die Endung `.htm` bzw. `.html` geben. Benennen wir also die obige Datei in `htmlerst.htm` bzw. `htmlerst.html` um und laden sie dann im Browser, erhalten wir

Abbildung 3.1: Browser-Anzeige für Datei `htmlerst.txt`

ein Erscheinungsbild, wie es in Abbildung 3.2 (beim Browser Netscape Navigator) bzw. in Abbildung 3.3 (beim Browser Internet-Explorer) gezeigt ist.
Nun interpretiert der Browser zwar die HTML-Anweisungen und zeigt wirklich nur den Dokumentenkörper (aus `<body>...</body>`) sowie den Titel (aus `<title>...</title>`) in der obersten Leiste des Browserfensters an. Trotzdem läßt die Formatierung des Textes sehr zu wünschen übrig. Warum das so ist, erklären die folgenden Punkte:

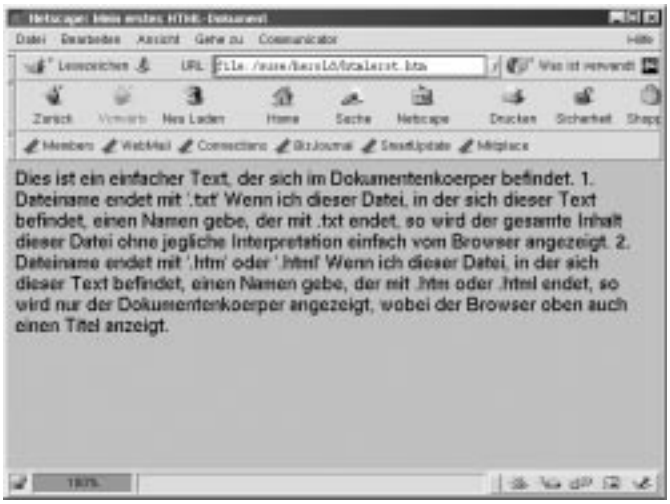

Abbildung 3.2: Netscape-Anzeige für Datei `htmlerst.htm` bzw. `htmlerst.html`

Abbildung 3.3: Internet-Explorer-Anzeige für Datei `htmlerst.htm` bzw. `htmlerst.html`

- HTML-Anzeigen haben – wenn dies nicht explizit angegeben wird –, keine festen Seitenlängen und Zeilenbreiten. Der Browser bestimmt selbst, wie lang eine Zeile sein darf, und bricht den Text entsprechend um. Ändert man z. B. die Größe des Fensters aus Abbildung 3.2, paßt der Browser den Text automatisch an die neue Fensterbreite an, wie dies in Abbildung 3.4 gezeigt ist.

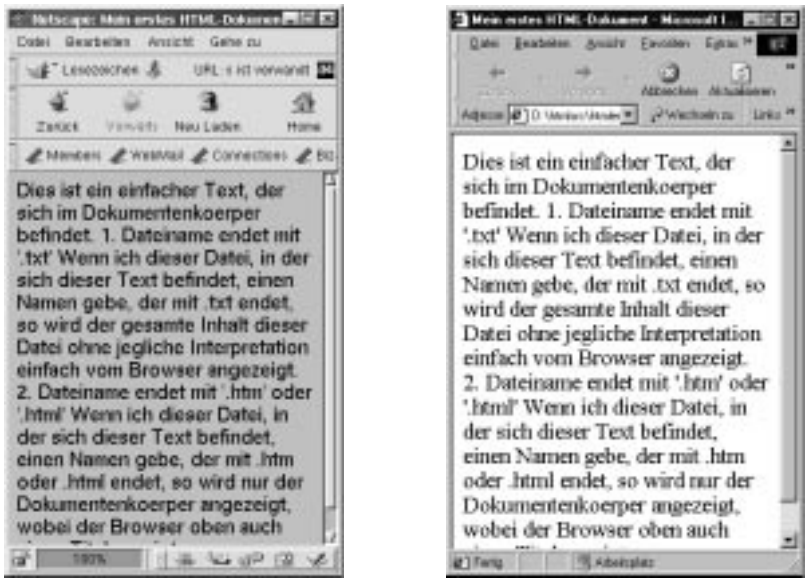

Abbildung 3.4: Browser-Anzeige für Datei `htmlerst.htm`, nachdem Fenstergröße geändert wurde (links: Netscape; rechts: Internet-Explorer)

- In einer HTML-Datei angegebene Leerzeilen, Leerzeichen und Zeilenumbrüche werden vom Browser einfach ignoriert; sie werden jeweils vom Browser durch ein Leerzeichen ersetzt.

3.5 Sonderzeichen in HTML-Dokumenten

Da HTML ursprünglich nur für das englischsprachige Alphabet ausgelegt wurde, mussten für Sonderzeichen, wie z. B. die deutschen Umlaute oder aber für HTML reservierte Zeichen (wie z. B. die spitzen Klammern), eigene Kodierungen eingeführt werden. Für die Kodierung solcher Sonderzeichen gelten die folgenden Konventionen:

- Die Sonderzeichen sind entweder als ASCII-Code (z. B. `&#148;`) oder als Name (z. B. `&ouml;`) im HTML-Dokument anzugeben.
- Ein solcher „Sonderzeichen-Ausdruck“ muss immer mit einem &-Zeichen beginnen und mit einem Semikolon abgeschlossen werden.

Tabelle 3.1 zeigt die Kodierungen der wichtigsten Sonderzeichen in HTML.
Lädt man die HTML-Datei 3.1 in einem Browser, wird das in Abbildung 3.5 gezeigte Fenster angezeigt.

HTML-Datei 3.1 – `sonderzeich.htm`:
Beispiel zu Sonderzeichen in HTML

```
<html>
<head>
<title>Sonderzeichen</title>
</head>
<body>
&Auml;=&amp;Auml;
&Ouml;=&amp;Ouml;
&Uuml;=&amp;Uuml;
&auml;=&amp;auml;
```

Tabelle 3.1: Darstellung von Sonderzeichen in HTML-Dokumenten

Zeichen	Code
Ä	Ä
Ö	Ö
Ü	Ü
ä	ä
ö	ö
ü	ü
ß	ß
“	"
<	<
>	>
Leerzeichen	(*non-breakable space*)

Kapitel 10

Frames

Mit Frames läßt sich der Anzeigebereich des Browsers in mehrere Fenster unterteilen, in denen sich jeweils unterschiedliche HTML-Dokumente anzeigen lassen.

10.1 Tags zum Erzeugen von Frames

Um Frames zu erzeugen, werden die beiden folgenden Tags zur Verfügung gestellt:

- `<frameset>...</frameset>`
  legt die Aufteilung des Browserfensters in einzelne Frames fest. Die einzelnen Frames können dann mit dem folgenden Tag darin eingebettet angegeben werden.
- `<frame>`
  spezifiziert jeweils einen einzelnen Frame; so wird hier z. B. über das Attribut `src=` der Pfadname des HTML-Dokuments festgelegt, das hier anzuzeigen ist.

Die typische Angabe für Frames sieht somit wie folgt aus:

```
<frameset ...>
  <frame src="HTML-Datei1">
  <frame src="HTML-Datei2">
  <frame src="HTML-Datei3">
  ...
</frameset>
```

Es ist erkennbar, dass die üblichen Tags `<html>`, `<head>`, `<title>` und `<body>` fehlen. Diese Tags (bis auf `<body>`) können zwar, müssen aber nicht angegeben werden. Es wäre z. B. auch die folgende Angabe erlaubt:

```
<html>
<head>
<title>Frames</title>
</head>
<frameset ...>
  <frame src="HTML-Datei1">
  <frame src="HTML-Datei2">
  <frame src="HTML-Datei3">
```

```
  ...
</frameset>
</html>
```

Hier würde zusätzlich noch ein Titel in der Kopfzeile des Browsers angezeigt. Auf keinen Fall darf nach dem Tag `</head>` das Tag `<body>` angegeben werden, um die mit `<frameset>...</frameset>` definierten Frames als Dokumentenkörper zu kennzeichnen. Das Tag `<body>` darf in einer Frame-Datei nur innerhalb des Tags `<noframes>...</noframes>` (siehe auch Kapitel 10.9 auf Seite 172) angegeben werden.

10.2 Vertikal angeordnete Frames

Zur vertikalen Aufteilung des Browserfensters in Frames steht das Attribut `rows=` im Tag `<frameset>` zur Verfügung. Nach `rows=` muss dabei in Anführungszeichen für jedes Frame festgelegt werden, wie hoch es sein soll. Die gewünschte Höhe eines Frames kann dabei als Pixel- oder als Prozentwert angegeben werden. Die Angabe eines Sternchens (`*`) legt fest, dass dieser Frame den Rest des Browserfensters vertikal ausfüllen soll. Wird `*` mehrmals angegeben, wird der restliche vertikale

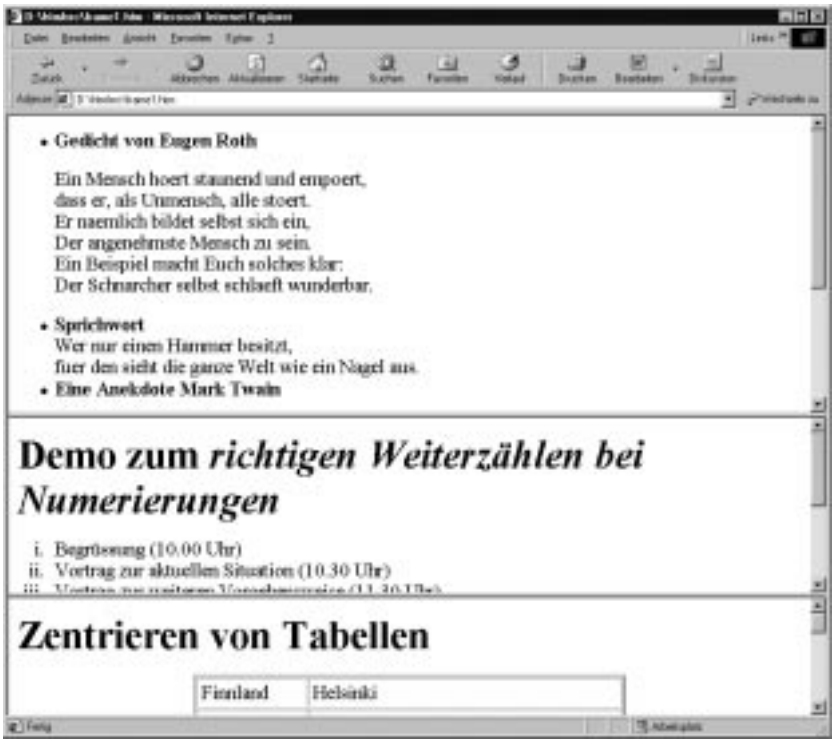

Abbildung 10.1: Browser-Anzeige (Internet-Explorer) für Datei 10.1 (`frame1.htm`)

Platz gleich unter den Sternchen verteilt. Alle Höhenangaben sind mit Kommas voneinander zu trennen.
Beispiele für `rows`-Angaben sind nachfolgend gezeigt:

- `<frameset rows="30%,70%">`
  legt fest, dass das erste Frame 30 Prozent und das zweite Frame 70 Prozent der Höhe des Browserfensters belegen soll.
- `<frameset rows="150,200,80">`
  legt fest, dass das erste Frame 150 Pixel, das zweite 200 Pixel und das dritte 80 Pixel hoch sein soll.
- `<frameset rows="20%,40%,*">`
  legt fest, dass das erste Frame 20 Prozent, das zweite 40 Prozent und das dritte den Rest des Browserfensters (in der Höhe) belegen soll.
- `<frameset rows="*,40%,*">`
  legt fest, dass das zweite Frame 40 Prozent, das erste und das dritte Frame jeweils 30 Prozent des Browserfensters (in der Höhe) belegen soll.

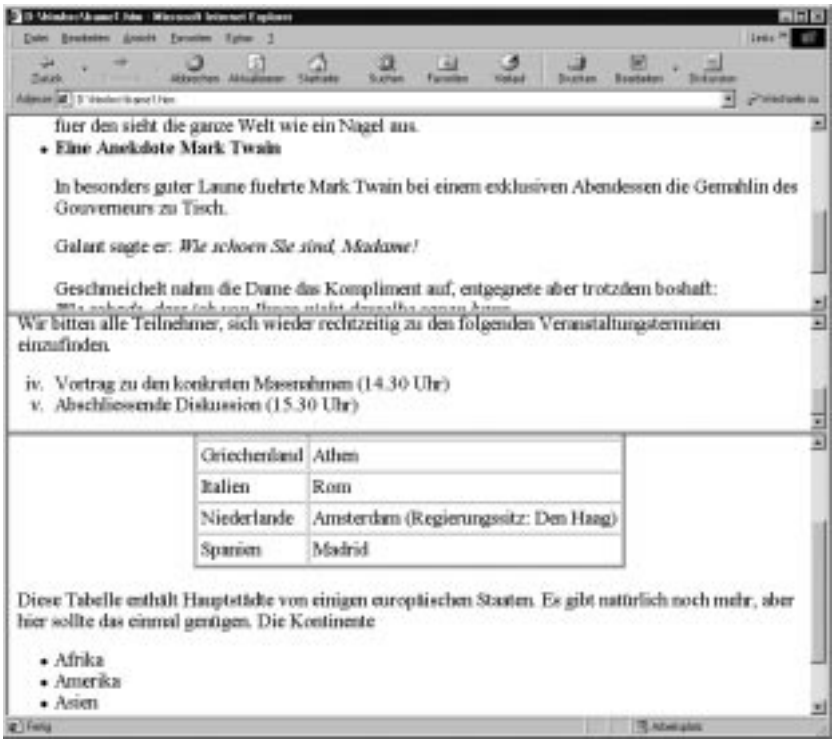

Abbildung 10.2: Browser-Anzeige (Internet-Explorer) für Datei 10.1 (`frame1.htm`), nachdem Benutzer in den Frames geblättert und Größe der einzelnen Frames verändert hat

- `<frameset rows="3*,10%,*,2*">`
  legt fest, dass das zweite Frame 10 Prozent, das erste Frame 45 Prozent, das dritte Frame 15 Prozent und das vierte Frame 30 Prozent des Browserfensters (in der Höhe) belegen soll.

Lädt man im Browser die HTML-Datei 10.1, wird z. B. das in Abbildung 10.1 gezeigte Fenster eingeblendet, in dem der Benutzer zum einen mit den eingeblendeten Bildlaufleisten in den Inhalten der einzelnen Frames vor- und zurückblättern kann, und zum anderen auch die Höhe der einzelnen Frames durch Verschieben der Fensterteiler mit der Maus verändern kann; siehe dazu auch Abbildung 10.2.

HTML-Datei 10.1 – `frame1.htm`:
Beispiel zu vertikalen Frames

```
<frameset rows="50%,30%,*">
  <frame src="aufzaehl5.htm">
  <frame src="aufzaehl11.htm">
  <frame src="table10.htm">
</frameset>
```

Die in der HTML-Datei 10.1 (`frame1.htm`) als Frames eingeblendeten HTML-Dokumente finden Sie auf folgenden Seiten: `aufzaehl5.htm` (auf Seite 47), `aufzaehl11.htm` (auf Seite 54) und `table10.htm` (auf Seite 83).
Ist bei einem `<frame>`-Tag kein `src=` angegeben, zeigt der Browser hierfür einen leeren Frame an. Sind weniger `<frame>`-Tags vorhanden als im `<frameset>`-Tag vorgegeben, zeigt der Browser für die fehlenden Angaben leere Frames an.

10.3 Horizontal angeordnete Frames

Zur horizontalen Aufteilung des Browserfensters in Frames steht das Attribut `cols=` im Tag `<frameset>` zur Verfügung. Nach `cols=` muss dabei in Anführungszeichen für jedes Frame festgelegt werden, wie breit es sein soll. Die gewünschte Breite eines Frames kann dabei als Pixel- oder als Prozentwert angegeben werden. Die Angabe eines Sternchens (`*`) legt fest, dass dieses Frame den Rest des Browserfensters horizontal ausfüllen soll. Wird `*` mehrmals angegeben, wird der restliche horizontale Platz gleich unter den Sternchen verteilt. Alle Breiteangaben sind mit Kommas voneinander zu trennen.
Lädt man im Browser die HTML-Datei 10.2, wird z. B. das in Abbildung 10.3 gezeigte Fenster eingeblendet, in dem der Benutzer zum einen mit den eingeblendeten Bildlaufleisten in den Inhalten der einzelnen Frames blättern kann, und zum anderen auch die Breite der einzelnen Frames durch Verschieben der Fensterteiler mit der Maus verändern kann; siehe dazu auch Abbildung 10.4. In dieser HTML-Datei `frame2.htm` ist erkennbar, dass man auch das gleiche HTML-Dokument (hier `table12.htm`) in unterschiedlichen Frames anzeigen lassen kann. Ebenso zeigt diese Datei, dass man nicht nur HTML-Dokumente in Frames anzeigen lassen kann, sondern z. B. auch Graphikbilder (hier `lausbub.jpg`).

Abbildung 10.3: Browser-Anzeige (Netscape) für Datei 10.2 (`frame2.htm`)

Abbildung 10.4: Browser-Anzeige (Netscape) für Datei 10.2 (`frame2.htm`), nachdem Benutzer in den Frames geblättert und die Breite der einzelnen Frames verändert hat

HTML-Datei 10.2 – `frame2.htm`:
Beispiel zu horizontalen Frames

```
<html>
<frameset cols="50%,*,200">
  <frame src="table12.htm">
  <frame src="lausbub.jpg">
  <frame src="table12.htm">
</frameset>
</html>
```

Das in der HTML-Datei 10.2 (`frame2.htm`) als Frame eingeblendete HTML-Dokument `table12.htm` finden Sie auf Seite 86.

10.4 Geschachtelte Frames

Frames lassen sich auch beliebig schachteln. Lädt man im Browser die HTML-Datei 10.3, wird z. B. das in Abbildung 10.5 gezeigte Fenster eingeblendet.

HTML-Datei 10.3 – `frame3.htm`:
Beispiel zu geschachtelten Frames

```
<html>
<head>
<title>Geschachtelte Frames</title>
</head>
<frameset cols="300,*,40%">
  <frameset rows="40%,*">
     <frame src="table10.htm">
     <frame src="lausbub.jpg">
  </frameset>
  <frameset rows="*,200,20%">
     <frame src="aufzaehl5.htm">
     <frame src="table12.htm">
     <frame src="aufzaehl11.htm">
  </frameset>
  <frameset>
     <frame src="lausbub.jpg">
  </frameset>
</frameset>
</html>
```

Die beiden Attribute `rows=` und `cols=` können auch gleichzeitig im Tag `<frameset>` angegeben werden. Fehlt – wie in den vorherigen Beispielen – eines der beiden Attribute, nimmt der Browser jeweils `rows="100%"` bzw. `cols="100%"` an.
Mit einer gleichzeitigen Angabe der beiden Attribute `rows=` und `cols=` ist ein entsprechendes Layout mit verschieden großen Frames möglich, wie dies in der HTML-Datei 10.4 gezeigt ist. Lädt man diese Datei im Browser, wird z. B. das in Abbildung 10.6 gezeigte Fenster eingeblendet. Verkleinert man dieses Browserfenster, werden die Frames auch entsprechend verkleinert, wie dies in Abbildung 10.7 gezeigt ist.

Abbildung 10.5: Browser-Anzeige (Internet-Explorer) für Datei 10.3 (`frame3.htm`)

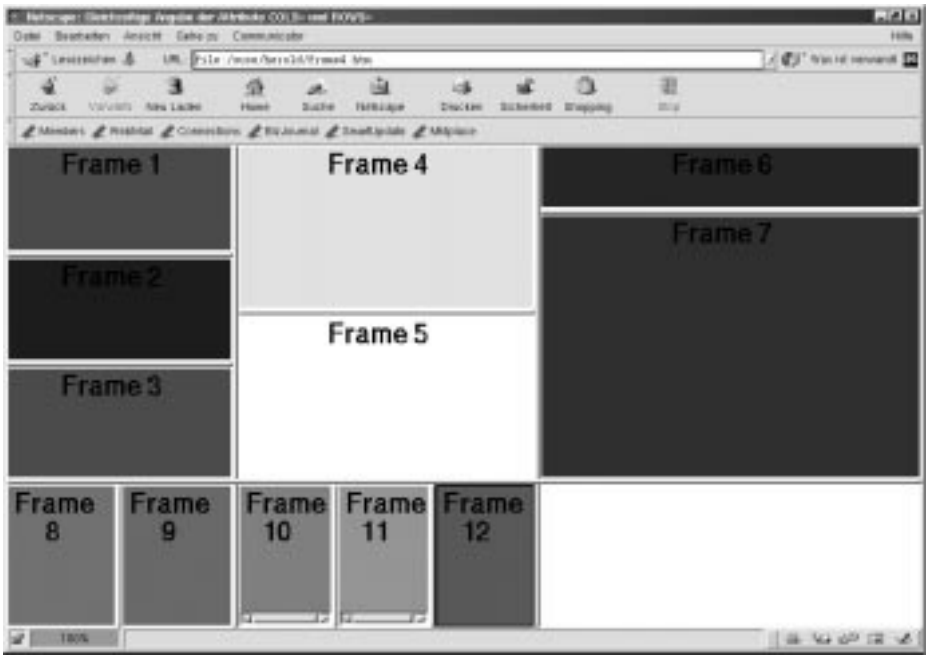

Abbildung 10.6: Browser-Anzeige (Netscape) für Datei 10.4 (`frame4.htm`)

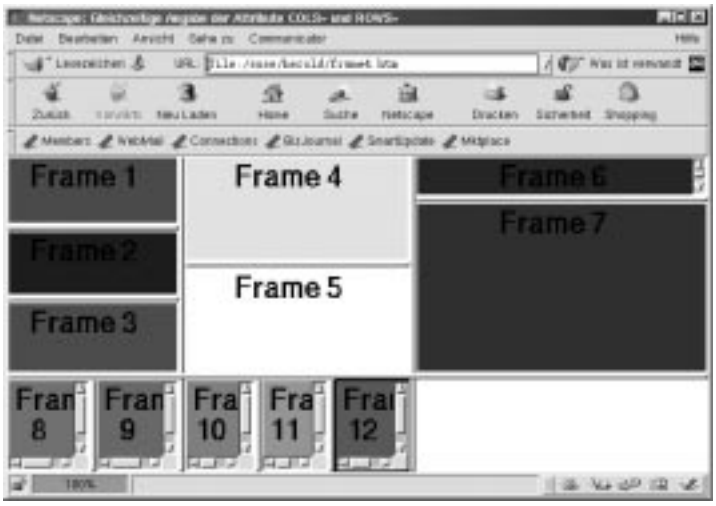

Abbildung 10.7: Browser-Anzeige (Netscape) für Datei 10.4 (`frame4.htm`), nachdem Browserfenster verkleinert wurde

HTML-Datei 10.4 – `frame4.htm`:
Weiteres Beispiel zu geschachtelten Frames

```
<html>
<head>
<title>Gleichzeitige Angabe der Attribute COLS= und ROWS=</title>
</head>
<frameset rows="70%,*" COLS="25%,33%,*"> <!-- Gesamtfenster:
                                              2 Zeilen (70%, *) und
                                              3 Spalten (25%, 33%, *) -->
 <!-- ......... Festlegen der 1. Zeile -->
     <frameset rows="33%,33%,*"> <!-- Frame11: 3 gleiche Zeilen -->
        <frame src="dat1.htm">
        <frame src="dat2.htm">
        <frame src="dat3.htm">
     </frameset>
     <frameset rows="50%,*">     <!-- Frame12: 2 gleiche Zeilen -->
        <frame src="dat4.htm">
        <frame src="dat5.htm">
     </frameset>
     <frameset rows="20%,*">     <!-- Frame13: 2 Zeilen (Relation 1:4) -->
        <frame src="dat6.htm">
        <frame src="dat7.htm">
     </frameset>
 <!-- ......... Festlegen der 2. Zeile -->
     <frameset cols="50%,*">     <!-- Frame21: 2 gleiche Spalten -->
        <frame src="dat8.htm">
        <frame src="dat9.htm">
     </frameset>
```

```
    <frameset cols=33%,33%,*>   <!-- Frame22: 3 gleiche Spalten -->
       <frame src="dat10.htm">
       <frame src="dat11.htm">
       <frame src="dat12.htm">
    </frameset>
                                <!-- Frame23: keinerlei Zuordnung -->
</frameset>
</html>
```

In den Frames der Datei 10.4 (`frame4.htm`) werden HTML-Dateien mit dem Namen `datx.htm` eingeblendet. Diese Dateien haben alle den folgenden Inhalt, wobei jeweils nur eine andere Farbe und eine andere Nummer angegeben ist:

```
<html>
<body bgcolor=green>
<h1 align=center>Frame 1</h1>
</body>
</html>
```

10.5 Bildlaufleisten bei Frames

Mit dem Attribut `scrolling=` kann im Tag `<frame>` festgelegt werden, ob bei einem Frame eine Scrollbar (Bildlaufleiste) einzublenden ist oder nicht. Tabelle 10.1 zeigt die möglichen Werte für das Attribut `scrolling=`.

Tabelle 10.1: Werte für das Attribut `scrolling=` beim Tag `<frame>`

scrolling=	**Auswirkung**
auto	Browser entscheidet selbst, ob Scrollbars notwendig sind oder nicht
yes	Es werden immer – unabhängig von der Größe des Inhalts – Scrollbars angezeigt
no	Es werden keine Scrollbars angezeigt, selbst wenn der Inhalt zu groß für den Frame ist

Lädt man im Browser die HTML-Datei 10.5, wird z. B. das in Abbildung 10.8 gezeigte Fenster eingeblendet.

HTML-Datei 10.5 – `frame5.htm`:
Beispiel zu Bildlaufleisten in Frames

```
<html>
<head>
<title>Scrollbars bei Frames</title>
</head>
<frameset cols="33%,33%,*">
  <frameset rows="50%,*">
     <frame scrolling=auto src="lausbub.jpg">
     <frame scrolling=auto src="bubclown.gif">
  </frameset>
  <frameset rows="50%,*">
     <frame scrolling=yes src="lausbub.jpg">
```

Abbildung 10.8: Browser-Anzeige (Internet-Explorer) für Datei 10.5 (`frame5.htm`)

```
    <frame scrolling=yes src="bubclown.gif">
  </frameset>
  <frameset rows="50%,*">
    <frame scrolling=no src="lausbub.jpg">
    <frame scrolling=no src="bubclown.gif">
  </frameset>
</frameset>
</html>
```

10.6 Frames und Hyperlinks

10.6.1 Hyperlinks auf bestimmte Frames

Um HTML-Dateien oder Graphikbilder über Hyperlinks in Frames einblenden zu lassen, müssen den entsprechenden Frames Namen gegeben werden. Dies ist mit den Attributen `name=` oder `id=` im Tag `<frame>` möglich:

```
<frame src="datei" name="NameDesFrames">  oder
<frame src="datei" id="NameDesFrames">
```

Um einen Hyperlink auf eine HTML-Datei oder ein Graphikbild zu erzeugen, muss im Tag `<a>` beim Attribut `href=` der Name der entsprechenden HTML- bzw. Graphikdatei angegeben werden. In welchem Frame die betreffende HTML-Datei bzw das Graphikbild anzuzeigen ist, muss mit dem Attribut `target=` festgelegt wer-

den, wobei hier als Wert der Name des entsprechenden Frames anzugeben ist, der mit dem Attribut `name=` bzw. `id=` im Tag `<frame>` festgelegt wurde.
Ein Hyperlink für das Einblenden einer HTML-Datei bzw. einer Graphik aus einer Datei in einem bestimmten Frame sieht somit wie folgt aus:

```
<a href="nameDerDatei target=nameDesFrames">Text des Hyperlinks</a>
```

Die HTML-Datei 10.6 erzeugt z. B. zwei horizontal angeordnete Frames, wobei sie dem linken Frame den Namen `links` und dem rechten Frame den Namen `rechts` gibt.

HTML-Datei 10.6 – `frame6.htm`:
Beispiel zu Hyperlinks und Frames

```
<html>
<head>
<title>Hyperlinks und Frames</title>
</head>
<frameset cols="30%,*">
    <frame src="verweise.htm" name="links">
    <frame src="splogo.gif" name="rechts">
</frameset>
</html>
```

Im linken Frame soll der Inhalt der HTML-Datei 10.7 (`verweise.htm`) und im rechten ein Graphikbild (Logo von SuSE PRESS) angezeigt werden.

HTML-Datei 10.7 – `verweise.htm`:
HTML-Datei mit Verweisen

```
<html>
<head>
<title>Verweise</title>
</head>
<body>
 <b>Inhaltsverzeichnis</b><br><br>
 <a href="table12.htm" target="rechts">Tabelle mit Hintergrundfarbe</a><br><br>
 <a href="aufzaehl5.htm" target="rechts">Ein Gedicht von Eugen Roth</a><br><br>
 <a href="lausbub.jpg" target="rechts">Ein Bild von einem Lausbub</a>
</body>
</html>
```

Die in der HTML-Datei 10.7 (`verweise.htm`) angegebenen HTML-Dokumente finden Sie auf folgenden Seiten: `table12.htm` auf Seite 86 und `aufzaehl5.htm` auf Seite 47.
Lädt man im Browser nun die HTML-Datei 10.6 (`frame6.htm`), wird das in Abbildung 10.9 gezeigte Fenster eingeblendet.
Klickt der Benutzer nun z. B. im linken Frame auf den Hyperlink „`Ein Gedicht von Eugen Roth`“, blendet der Browser ihm im rechten Frame die HTML-Datei 10.5 (`aufzaehl5.htm`) ein; siehe auch Abbildung 10.10.
Klickt der Benutzer nun z. B. im linken Frame auf den Hyperlink „`Ein Bild von einem Lausbub`“, blendet der Browser ihm das Graphikbild aus der Datei `lausbub.jpg` ein; siehe auch Abbildung 10.11.

Kapitel 11

Formulare

HTML bietet auch die Möglichkeit, so genannte Formulare zu erstellen. Formulare enthalten Elemente wie Eingabefelder, Auswahllisten usw., in denen der Benutzer Eingaben vornehmen kann oder bestimmte Auswahlmöglichkeiten hat. Die eingegebenen bzw. ausgewählten Daten können dann natürlich auch über das Internet verschickt werden.

11.1 Allgemeines zu Formularen

11.1.1 Formulardefinition (Tag <form>)

Formulare müssen immer mit dem Tag `<form>...</form>` geklammert sein:

```
<form>

... Formularanweisungen, die das Layout des Formulars festlegen ...

</form>
```

Allgemein gilt folgendes für ein Formular:

- In einem Formular können neben Steuerelementen (Eingabefelder, Auswahllisten usw.), die wir in diesem Kapitel noch kennenlernen werden, auch einfache Texte (zur Beschriftung von Elementen) enthalten sein.
- Ein Formular kann auch Tags wie z. B. `<p>`, `<b>`, `<img>`, `<br>` usw. zur Formulargestaltung enthalten.
- Ein HTML-Dokument kann mehrere `<form>`-Abschnitte enthalten.
- Eine Schachtelung von `<form>`-Tags ist nicht erlaubt.

Zur vollständigen Formulardefinition müssen dem Server nun noch weitere Informationen gegeben werden. Diese sind zum einen die Internetadresse (URL), an welche die vom Benutzer eingegebenen Daten gesendet werden sollen, zum anderen muss auch die Übertragungsart definiert werden.

11.1.2 Festlegen der Versandadresse (Attribut `action=`)

Die Versandadresse legt den Ort fest, an den die Benutzereingaben im Formular zu schicken sind. Das Festlegen der Versandadresse ist mit dem Attribut `action=` im Tag `<form>` möglich. Es sind verschiedene Arten von Versandadressen möglich, die durch das entsprechende Protokoll festgelegt werden:

- `http:`
  Der URL adressiert hierbei ein so genanntes CGI-Skript, dem die eingegebenen Benutzerdaten zur Verarbeitung geschickt werden. *CGI* (*Common Gateway Interface*) ist ein Standard für CGI-Skripte, die in Sprachen wie PHP, Perl, JavaScript usw. geschrieben sind, um Informationen von HTTP-Servern zu verarbeiten. Später wird dazu in Kapitel 11.5.2 auf Seite 196 ein Beispiel gegeben.
- `mailto:`
  Hier wird eine E-Mail-Adresse angegeben, an die die eingegebenen Benutzerdaten geschickt werden.

11.1.3 Festlegen der Übertragungsmethode (Attribut `method=`)

Die Übertragungsmethode beschreibt die Art und Weise, wie die Daten zum Server übertragen werden. Das Festlegen der Übertragungsmethode ist mit dem Attribut `method=` im Tag `<form>` möglich. Man unterscheidet zwischen zwei verschiedenen Übertragungsmethoden:

- `method=post`
  Bei dieser Übertragungsmethode werden die Formulardaten in die HTTP-Anforderung eingebettet, die dem Server geschickt wird. Dadurch werden die Daten versteckt und nicht für jedermann lesbar im Internet übertragen. Diese Übertragungsmethode wird vom W3C-Standard bevorzugt. Bei Benutzereingaben, die über E-Mail übertragen werden, sollte man diese Methode wählen.
- `method=get`
  Bei dieser Übertragungsmethode werden die Formulardaten getrennt durch ein Fragezeichen an den URL (von `action=`) angehängt, bevor der so erweiterte URL an den Server geschickt wird.

Eine Formulardefinition könnte z. B. folgendermaßen aussehen:

```
<form action=mailto:herold@suse.de method=post>

... Formularanweisungen ...

</form>
```

Diese Anweisungen definieren ein Formular, das die eingegebenen Benutzerdaten mittels E-Mail an die Adresse `herold@suse.de` schickt.

11.2 Eingabefelder

Eingabefelder sind Formularfelder, in die der Benutzer Text eingeben kann. Man unterscheidet zwischen einzeiligen und mehrzeiligen Eingabefeldern.

11.2.1 Einzeilige Eingabefelder

Einzeilige Eingabefelder werden durch das Tag `<input>` definiert. Dieses Tag kann ausschliesslich innerhalb einer Formulardefinition angegeben werden und besitzt unter anderem die folgenden Attribute:

- `type=`**`text`**
  legt fest, dass ein Eingabefeld einzublenden ist, in dem jeder vom Benutzer eingegebener Text anzuzeigen ist.
- `type=`**`password`**
  legt fest, dass ein Eingabefeld einzublenden ist, in dem der vom Benutzer eingegebene Text nicht anzuzeigen ist. Jedes eingegebene Zeichen wird hier z. B. als Sternchen (*) angezeigt. Diese Form von Eingabefelder dient zur Eingabe von Paßwörtern.
- `name=`
  legt den Namen des entsprechenden Eingabefelds fest.
- `size=x`
  legt die Länge des Eingabefelds fest; `x` gibt dabei die Länge des Eingabefelds in Zeichen an. Ist die Eingabe länger, so wird der bereits eingegebene Text links im Eingabefeld weggeschoben. Mit dem Cursor-Steuerzeichen ← kann man den weggeschobenen Text wieder im Eingabefeld sichtbar machen.
- `maxlength=x`
  legt die maximale Länge fest, die ein Text im Eingabefeld haben kann; `x` gibt dabei die maximale Anzahl von Zeichen an, die im Eingabefeld eingegeben werden können.
- `VALUE=`
  legt einen Vorgabewert fest, der beim Einblenden des Formulars im Eingabefeld oder bei einem Klick auf den RESET-Button (siehe Kapitel 11.5 auf Seite 200) anzuzeigen ist.

Lädt man im Browser die HTML-Datei 11.1, wird das in Abbildung 11.1 gezeigte Fenster eingeblendet, in dem schon einige Eingaben vorgenommen wurden.

Abbildung 11.1: Browser-Anzeige für Datei 11.1 (`formular1.htm`) (links: Netscape, rechts: Internet-Explorer)

HTML-Datei 11.1 – `formular1.htm`:
Beispiel zu einem Formular mit einzeiligen Eingabefeldern

```
<html>
<head>
<title>Formular mit einzeiligen Eingabefeldern</title>
</head>
<body>
<h1>Formular mit einzeiligen Eingabefeldern</h1>
<form method=post action="mailto:herold@suse.de">
   Kennwort: <input type=text name="loginname"><br>
   Passwort: <input type=password name="geheimwort" SIZE=6><br>
   Alter:    <input type=text name="alter" maxlength=2><br>
   Land:     <input type=text name="country" value="Deutschland"><br>
   Farbe:    <input type=text name="farbe" size=8 maxlength=9><br>
</form>
</body>
</html>
```

11.2.2 Mehrzeilige Eingabefelder

Mehrzeilige Eingabefelder (oft auch mehrzeilige Textfelder genannt) kann man mit dem Tag `<textarea>...</textarea>` einblenden lassen. Sie können zur Eingabe von Bemerkungen oder sonstigem beliebigen Text dienen. Mit dem Attribut `name=` wird wieder dem mehrzeiligen Textfeld ein Name zugewiesen. Des weiteren muss natürlich auch noch die Größe des Textfeldes festgelegt werden. Dazu stehen die beiden folgende Attribute zur Verfügung:

- `cols=x`: `x` legt die Breite des Textfelds (in Zeichen) fest.
- `rows=x`: `x` legt die Höhe des Textfelds (in Zeilen) fest.

Lädt man im Browser die HTML-Datei 11.2, wird z. B. das in Abbildung 11.2 gezeigte Fenster eingeblendet, in dem der vorgegebene Text schon angezeigt wird, aber der Benutzer nun noch weiteren Text eintragen kann.

Abbildung 11.2: Browser-Anzeige für Datei 11.2 (`formular2.htm`) (links: Netscape, rechts: Internet-Explorer)

Kapitel 17

Cascading Style Sheets

Cascading Style Sheets sind eine unmittelbare Ergänzung zu HTML. Mit Style-Sheets kann man die Formateigenschaften von HTML-Befehlen festlegen. So kann man z. B. mit Hilfe von Style-Sheets vorgeben, dass Überschriften der zweiten Stufe immer eingerückt und mit einer Schriftgröße von 16 Punkt in der Schriftart *Times-Roman* anzuzeigen sind. Wie wir in diesem Kapitel sehen werden, bieten Style-Sheets aber noch eine Vielzahl von anderen Möglichkeiten,
Es existieren mehrere Sprachen zum Definieren von Style-Sheets. Die bekannteste Sprache ist CSS (*Cascading Style Sheets*), die vom W3-Konsortium, das auch für die Normierung von HTML zuständig ist, vorgeschlagen wird. Diese Sprache CSS wird hier vorgestellt. Zu CSS existieren zwischenzeitlich mehrere Versionen:

- Version 1.0 (von 1996) und
- Version 2.0 (1998).

An den entsprechenden Stellen wird auf Versions-Spezifika hingewiesen. Derzeit wird an der 3. Version von CSS beim W3C gearbeitet.
Leider unterstützen nicht alle Browser CSS vollständig. Netscape 4.x interpretiert fast den vollen Sprachumfang von CSS 1.0 und einen Teil der Befehle von CSS 2.0. Netscape 6.0 unterstützt CSS 1.0 und CSS 2.0 nahezu vollständig.
Der Microsoft Internet-Explorer unterstützt CSS 1.0 ab seiner Version 3.0. In der Version 4.0 unterstützt er einen Teil von CSS 2.0 und einige spezielle, von Microsoft eingeführte Style-Sheet-Angaben. Ab Version 5.5 unterstützt der Microsoft Internet-Explorer CSS 1.0 vollständig und größtenteils auch CSS 2.0
Andere Browser unterstützen meist nur CSS 1.0 oder nur Teile von CSS 1.0 und / oder CSS 2.0.

17.1 Allgemeines zu Style-Sheets

17.1.1 Definieren von Style-Sheets für eine HTML-Datei

Innerhalb eines HTML-Dokuments kann man sich einen Bereich für Style-Sheet-Angaben definieren. Dazu steht das Tag `<style...>...</style>` zur Verfügung, das man in den Kopfteil des HTML-Dokuments einbetten muss:

```
<html>
<head>
..........
<style type="text/css">
<!--
 ... Style-Sheet-Angaben ...
//-->
</style>
..........
</head>
<body>
..........
</body>
</html>
```

Im Tag `<style...>` muss man dabei den Typ der Formatdefinition angeben, was bei allen hier beschriebenen Formatiermöglichkeiten mit der Angabe `type="text/css"` möglich ist. Damit Browser, die keine Style-Sheets unterstützen, die Style-Sheet-Angaben nicht als Text ausgeben, gibt man den Bereich der eigentlichen Style-Sheet-Angaben üblicherweise in einem mehrzeiligen HTML-Kommentar `<!-- ...//-->` an. Style-Sheet-Angaben, die im Kopf einer HTML-Datei definiert werden, gelten nur für diese eine HTML-Datei.
In XHTML müssen Style-Sheets in einen CDATA-Abschnitt eingebettet werden; siehe auch Kapitel 19.13 auf Seite 457.

17.1.2 Globale Style-Sheets für mehrere Dateien

Möchte man einmal definierte Style-Sheets nicht nur in einer Datei benutzen, muss man die betreffenden Style-Sheets in einer eigenen Datei definieren. Jede HTML-Datei, die diese Style-Sheets benötigt, muss dann nur diese „Style-Sheet-Datei“ (CSS-Datei) einbinden. Ändert man in diesem Fall die Angaben in der „Style-Sheet-Datei“, wirken sich die Änderungen einheitlich auf alle Dateien aus, in denen diese CSS-Datei eingebunden ist.
Die CSS-Datei muss eine reine Textdatei sein, deren Name die Endung `.css` haben sollte. Diese Datei sollte nur Formatdefinitionen und keine HTML-Befehle enthalten.
Um eine CSS-Datei in ein HTML-Dokument einzubinden, muss folgende Zeile im Kopfteil angegeben werden:

```
<link rel=stylesheet type="text/css" href="css-datei">
```

Befindet sich die CSS-Datei in einem anderen Directory bzw. auf einem anderen Server, muss man für *`css-datei`* den relativen Pfad bzw. die entsprechende URL-Adresse angeben.
Nach der `<link>`-Zeile können noch weitere Style-Sheets definiert werden:

```
<head>
.......
<link rel=stylesheet type="text/css" href="css-datei">
<style type="text/css">
<!--
```

```
 ... Weitere Style-Sheet-Angaben ...
//-->
</style>
.......
</head>
```

Definiert man in einer HTML-Datei Style-Sheet-Formate, die auch in der eingebundenen CSS-Datei definiert sind, hat die Formatdefinition in der HTML-Datei höhere Priorität als die gleichnamige in der CSS-Datei. So kann man immer wieder verwendete Formate importieren und einige davon mit dateispezifischen Formaten überschreiben.
Neben der HTML-Syntax zum Einbinden von CSS-Dateien gibt es auch noch die folgende CSS-Syntax, die das gleiche bewirkt:

```
<html>
<head>
.......
<style type="text/css">
<!--
    @import url(css-datei);  oder auch:
    @import "css-datei";
     .....
     ... Weitere Style-Sheet-Angaben ...
//-->
</style>
.......
</head>
```

Es gibt also zwei Möglichkeiten, CSS-Dateien in einer HTML-Datei einzubinden, entweder über das Tag `<link>` oder mit `@import`. CSS 2 legt fest, dass die Browser die eingebundenen Style-Sheets – abhängig von der Art der Einbindung – unterschiedlich behandeln:

1. Bei mehreren mit dem Tag `<link>` eingebundenen CSS-Dateien sollte der Browser den Benutzer wählen lassen, welche der CSS-Dateien für die Anzeige zu verwenden ist. Bei einer Angabe, wie z. B.:

   ```
   <link rel=stylesheet type="text/css" href=farbig.css title="Anzeige farbig">
   <link rel=stylesheet type="text/css" href=sw.css title="Anzeige schwarzweiss">
   ```

   könnte der Browser z. B. zuerst ein Fenster einblenden, in dem der Benutzer zwischen den beiden Alternativen „`Anzeige farbig`“ oder „`Anzeige schwarzweiss`“ auswählen kann, bevor dann die entsprechende CSS-Datei (`farbig.css` oder `sw.css`) zur Darstellung des HTML-Dokuments verwendet wird.
2. Bei mehreren mit `@import` eingebundenen Dateien sollte der Browser ohne Rückfrage beim Benutzer alle angegebenen Style-Sheet-Angaben aus den unterschiedlichen CSS-Dateien zusammenmischen und dann auf das HTML-Dokument anwenden. Sind in den CSS-Dateien Style-Sheet-Angaben enthalten, die für eine Eigenschaft unterschiedliche Formatierungen festlegen, überschreibt die zuletzt importierte Formatierungs-Angabe eine zuvor importierte.

Es ist jedoch zu erwähnen, dass viele Browser sich zwar an den Punkt 2 halten, aber CSS-Dateien, die mit dem Tag `<link>` eingebunden werden, genauso behandeln, als ob dies mit `@import` geschehen sei.

17.1.3 Kaskadieren bei gleichen Style-Sheet-Angaben

Cascading Style Sheets sind *kaskadierend*. Dies bedeutet, dass bei unterschiedlichen Style-Sheet-Angaben für ein und dasselbe Tag die unterschiedlichen Eigenschaften sich ergänzen, während bei gleichen Eigenschaften die zuletzt definierte Eigenschaft eine zuvor definierte Eigenschaft überschreibt.
Hat man z. B. die beiden folgenden CSS-Dateien:

- CSS-Datei `ueber1.htm`:

```
h1 { background-color:red; color:white; font-style:bold; font-size:36pt; }
```

- CSS-Datei `ueber2.htm`:

```
h1 { font-style:italic; }
```

und man lädt die HTML-Datei 17.1 im Browser, wird z. B. das in Abbildung 17.1 gezeigte Fenster eingeblendet, in dem eine Überschrift erster Ordnung kursiv und unterstrichen in weisser Schrift der Größe 36 auf blauem Hintergrund angezeigt wird.

HTML-Datei 17.1 – `cascad1.htm`:
Beispiel zur Kaskadierung von CSS-Formaten

```
<html>
<head><title>Cascading Style Sheets</title>
 <style type="text/css">
   <!--
     @import url(ueber1.css);
     @import url(ueber2.css);
     h1 { background-color:yellow; text-decoration:underline; }
   //-->
 </style>
</head>
<body>
<h1 style="background-color:blue;">Überschrift 1. Ordnung</h1>
</body>
</html>
```

Abbildung 17.1: Browser-Anzeige für HTML-Datei 17.1 (`cascad1.htm`)

17.1.4 Kommentare innerhalb von Style-Sheets

Um einen Kommentar in einen Style-Sheet einzufügen, steht eine spezielle an die Programmiersprache C angelehnte Syntax zur Verfügung:

```
/* Kommentar */
```

Der Kommentar erstreckt sich dabei vom ersten `/*` bis zum nächsten `*/`, so dass sich ein Kommentar auch über mehrere Zeilen erstrecken kann, wie z. B.:

```
   p  { color:yellow; } /* Format für gelben Text; neu definiert: 24.7.2002 */

     /* Nachfolgend etwas größere Überschrift in weiss auf blauem Hintergrund;
        sieht ganz nett aus, aber sollte nochmals von unserem
        Designer begutachtet werden;
                                                  neu definiert: 12.8.2002
      */
   h1 {  background-color:blue; color:white; font-style:bold; font-size:36pt; }
```

Eine Schachtelung von Kommentaren ist nicht erlaubt. So wäre z. B. die folgende Angabe nicht erlaubt:

```
     /* Nachfolgend etwas größere Überschrift in weiss auf blauem Hintergrund;
            /* neu definiert: 12.8.2002 */ <-- Schachtelung nicht möglich
      */
   h1 {  background-color:blue; color:white; font-style:bold; font-size:36pt; }
```

17.1.5 Style-Sheet-Sprache festlegen

Man kann in einer HTML-Datei im Tag `<meta...>` angeben, welche Definitionssprache bei den Style-Sheet-Definitionen verwendet wird, wie z. B.:

```
<head>
  .....
  <meta http-equiv="Content-Style-Type" content="text/css">
  .....
</head>
```

Diese Angabe ist z. Z. bei Verwendung der CSS-Sprache nicht unbedingt erforderlich. Mit der Angabe `<meta http-equiv="Content-Style-Type" content="text/css">` legt man fest, dass die CSS-Syntax bei den Style-Sheet-Definitionen in dieser Datei verwendet wird. Verwendet man eine andere Style-Sheet-Sprache, muss man anstelle von `"text/css"` den MIME-Typ der entsprechenden Definitionssprache angeben.

17.1.6 Style-Sheets für unterschiedliche Ausgabemedien (CSS 2.0)

Einbinden der CSS-Dateien (CSS 2.0)

Hier wird vorgestellt, wie man unterschiedliche Style-Sheets für verschiedene Ausgabemedien benutzen kann, indem man für die jeweiligen Ausgabemedien eigene CSS-Dateien erstellt, die man dann einbindet. CSS hat für die unterschiedlichen Ausgabemedien eigene Kennwörter eingeführt, die man beim Attribut `media=` im Tag `<link...>` angeben muss, wie z. B.:

Tabelle 17.1: Angaben für das Attribut `media=`

`media=`	Style-Sheets, die zu verwenden sind für die Ausgabe
`screen`	am Bildschirm (Voreinstellung)
`print`	am Drucker
`aural`	am Lautsprecher
`projection`	am Overhead-Projektor
`tv`	am Fernseher
`handheld`	an Handys, Palmtops oder ähnliche Geräte mit kleinem Display
`braille`	an taktilen Braille-Medien
`all`	an allen Medientypen

```
<head>
.......
<link rel=stylesheet media="screen" href="bildschirm.css">
<link rel=stylesheet media="print"  href="drucker.css">
<link rel=stylesheet media="tv"     href="fernseher.css">
.......
</head>
```

Tabelle 17.1 zeigt mögliche Angaben beim Attribut `media=`.
Die Syntax zum Einbinden eigener CSS-Dateien für unterschiedliche Ausgabemedien mit dem Tag `<link...>` ist HTML-Syntax.
Daneben gibt es auch noch eine CSS-Syntax, die das gleiche bewirkt, wie z. B.:

```
<head>
.......
<style type="text/css">
<!--
   @import url(drucker.css) print;
   @import url(multimedia.css) tv, projection;
      oder auch:
   @import "drucker.css" print;
   @import "multimedia.css" tv, projection;
//-->
</style>
.......
</head>
```

Eine CSS-spezifische Einbindung muss also innerhalb des Tags `<style>...</style>` folgendermassen angegeben werden:

```
  ....
  @import url(css-datei) medium1, medium2, ...;
      oder auch:
  ....
  @import "css-datei" medium1, medium2, ...;
```

CSS-Definitionen in der HTML-Datei (CSS 2.0)

Neben der Möglichkeit, Style-Sheet-Dateien für unterschiedliche Ausgabemedien in HTML einzubinden, kann man auch direkt innerhalb des Tags `<sty-`

le>...</style> Style-Sheets für die unterschiedlichen Ausgabemedien definieren, wie z. B.:

```
<head>
.......
<style type="text/css">
<!--
   @media print
   {
     ... Style-Sheet-Angaben zum Drucken ...
   }
   @media screen, tv
   {
     ... Style-Sheet-Angaben zur Ausgabe am Bildschirm und Fernseher ...
   }
//-->
</style>
.......
</head>
```

Die folgende Angabe würde z. B. bewirken, dass am Bildschirm für das HTML-Dokument ein weisser Hintergrund gewählt wird, während bei einer Anzeige dieses HTML-Dokuments auf einem Fernseher oder einem Overhead-Projektor ein blauer Hintergrund verwendet wird:

```
<style type="text/css">
<!--
   body  { background-color:white; }

   @media tv, projection {
       body  { background-color:blue; }
   }
//-->
</style>
```

17.2 Definieren von Formaten (Style-Sheets)

17.2.1 Eigene Formate für HTML-Tags

Man kann für HTML-Tags, wie z. B. für <h1>, <h2>, <td>, <p> usw. mit Hilfe von Style-Sheet-Angaben eigene Formate definieren. Verwendet man dann das entsprechende HTML-Tag, werden die selbstdefinierten Formate hierfür verwendet.
Lädt man die HTML-Datei 17.2 im Browser, wird z. B. das in Abbildung 17.2 gezeigte Fenster eingeblendet, in dem nun alle Überschriften erster Ordnung kursiv in weisser Schrift der Größe 36 auf blauem Hintergrund angezeigt werden. Absätze und Tabellenzellen werden bei diesem HTML-Dokument in gelber Schrift der Größe 24 auf schwarzem Hintergrund angezeigt.

Kapitel 18

Eine kurze Einführung in XML

Hier wird eine kurze Einführung in XML gegeben. XML (*Extensible Markup Language*) ist nicht nur einfach eine weitere Sprache, sondern ein neues Konzept für die Datenspeicherung und deshalb nicht nur auf das Internet zugeschnitten. Die Idee zu XML hatte *Tim Berners-Lee*, der Vater von HTTP und HTML, in den Anfangsjahren der Internet-Euphorie, als er erkannte, dass HTML alleine nicht ausreichen würde, die ständig steigenden Ansprüche zu erfüllen. Es wurde offensichtlich, dass ein Standard geschaffen werden musste, in den HTML, das zwischenzeitlich weitverbreitet war, integriert werden konnte. Dazu schuf er im Jahre 1994 ein offenes Forum für Entwickler aus den unterschiedlichsten Branchen, in dem dann auch die verschiedensten Ansätze diskutiert wurden.
Im Jahre 1996 wurde dann der erste Entwurf zu XML präsentiert. Die Standardisierung von XML unterliegt nun – wie auch bei HTML – dem W3-Konsortium. Im Februar 1998 brachte das W3-Konsortium schließlich die erste Empfehlung zu XML heraus. Danach bemühte sich das W3-Konsortium, andere vorhandene Metasprachen mit Hilfe von XML zu standardisieren. Eine der wichtigste Sprachen war dabei natürlich HTML, das dann auch mit Hilfe von XML zu dem neuen Standard XHTML (siehe Kapitel 19) umdefiniert wurde und auf dieser Basis weiterentwickelt wird. Auch andere Sprachen wie z. B. SVG (Vektorgrafikformat), MathML (Mathematical Markup Language, RDF (Resource Description Framework) und WML (Wireless Markup Language) für Handys wurden mit Hilfe von XML umdefiniert, um sie als software-unabhängige Beschreibungssprachen einsetzen zu können.
XML ist ein völlig neuer Ansatz, der die ganze Software-Industrie revolutionieren könnte. Das Ziel von XML ist es nämlich, unabhängige Dateiformate bereitzustellen, so dass sie beliebig zwischen den unterschiedlichen Programmen austauschbar sind. Setzt sich XML z. B. als Dateiformat für Office-Pakete durch, kann ein auf XML basierender Brief mit jedem beliebigen Office-Programm, ob nun mit Microsoft-Word oder mit StarOffice gelesen und editiert werden. Das einzige, was sich dabei noch unterscheiden wird, ist die Bedienoberfläche, die sehr wahrscheinlich auch weiterhin in den unterschiedlichen Programmen verschieden sein wird. Da XML von Anfang universell entworfen wurde, könnte es sich als Standard für die unterschiedlichsten Anwendungen durchsetzen, so dass man je nach Anwendungstyp,

wie z. B. Graphik, Datenbanken, Computerspiele, CAD usw., jeweils nur noch ein unabhängiges Dateiformat kennt, das auf XML basiert.
Entscheidend bei XML ist, dass Daten und Layout grundsätzlich voneinander getrennt sind. Das gleiche XML-Dokument kann sowohl von einem Browser als Webseite eingeblendet werden, oder aber an einem Drucker ausgegeben werden, da die Daten selbst vom Layout getrennt sind. Die Darstellung der Daten, also das Layout, erfolgt mit Hilfe von Konvertern, die XML-Daten in andere medienabhängige Formate (wie z. B. in HTML oder in Postscript) übersetzen.

18.1 Allgemeines zu XML

18.1.1 Die Metasprache XML

XML (*Extensible Markup Language*) ist eine Metasprache, mit der man Dokumenttypen definieren kann. XML gibt allerdings nur die Regeln vor, die beim Definieren von Dokumenttypen einzuhalten sind. Anders als HTML schreibt XML keine bestimmten Schlüsselwörter mit einer zugeordneten Funktionsweise vor, sondern legt lediglich den Aufbau und die Struktur fest, an die sich jedes XML-Dokument halten muss.
Nachfolgend ist ein mögliches Aussehen eines XML-Dokuments gezeigt:

```
<adresse nr="1">
   <titel />
   <name>Hans Meier</name>
   <postfach>352</postfach>
   <wohnort>12345 Musterstadt</wohnort>
</adresse>
```

An diesem Beispiel kann man schon die grundsätzliche Struktur von XML-Dokumenten sowie die dahinterliegende XML-Philosophie erkennen:

- In XML existieren so genannte Tags (Elemente). Diese Tags in spitzen Klammern (wie z. B. `<adresse>`) treten immer paarweise auf, wobei das Endtag (wie z. B. `</adresse>`) den jeweiligen Block beendet.
- Tags können beliebig geschachtelt werden. Die Schachtelung entspricht dabei einer Klammerung wie man sie z. B. von der Mathematik her kennt. Dies bedeutet, dass jedes Element auf der Ebene abgeschlossen werden muss, auf der es auch geöffnet wurde.
- Leere Elemente können das Endtag gleich mitenthalten (`<titel />`).
- Ein Tag kann Attribute besitzen, wie z. B. eine Kennzeichnung (`<adresse nr="1">`).
- Die Information selbst steht innerhalb des Start- und Endtags, wie z. B. innerhalb von `<wohnort>...</wohnort>`.

Optional kann auch ein Kopfteil angegeben werden, der die XML-Version und den Zeichensatz festlegt, wie das nachfolgend gezeigt ist:

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

```
    <name>Hans Meier</name>
    <postfach>352</postfach>
    <wohnort>12345 Musterstadt</wohnort>
  </adresse>
  <adresse>
    <title>Dr.</title>
    <name>Kati Wilhelm</name>
    <strasse>Waldstr. 17</strasse>
    <wohnort>54342 Hallenfurt</wohnort>
  </adresse>
</adressen>
```

18.1.2 Dokumenttyp-Definitionen (DTDs)

Die Regeln für erlaubte Elemente, Attribute und Verschachtelungsmöglichkeiten einer XML-Metasprache werden unabhängig von den eigentlichen Daten definiert. Die Vorgabe dieser Regeln nennt man Dokumenttyp-Definition (*document type definition*, Abkürzung *DTD*). Ein Dokument, das alle Vorgaben einer DTD erfüllt, bezeichnet man als ein *gültiges* Dokument. Das Verfahren, um zu überprüfen, ob eine XML-Datei nach den Regeln ihrer zugehörigen DTD fehlerfrei ist, heißt *Validierung*. Für das vorherige Beispiel könnte eine DTD wie folgt aussehen:

```
<!ELEMENT adressen (adresse)*>

<!-- Der Titel kann, muss aber nicht angegeben werden und
     es ist entweder das Postfach oder die Strasse, aber
     nicht beides anzugeben                                -->
<!ELEMENT adresse (titel?, name, (postfach | strasse), wohnort)>

<!ELEMENT titel (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT postfach (#PCDATA)>
<!ELEMENT strasse (#PCDATA)>
<!ELEMENT wohnort (#PCDATA)>
```

Die Angabe

```
<!ELEMENT adressen (adresse)*>
```

legt fest, dass das Tag `<adresse>` nur innerhalb von `<adressen>...</adressen>` angegeben werden darf. Der Stern (`*`) legt dabei fest, dass innerhalb von `<adressen>...</adressen>` das Tag `<adresse>` entweder gar nicht angegeben sein muss, oder aber auch einmal oder mehrmals vorkommen darf.
Die Regel

```
<!ELEMENT adresse (titel?, name, (postfach | strasse), wohnort)>
```

legt fest, dass die Tags `<titel>`, `<name>`, `<postfach>`, `<strasse>` und `<wohnort>` nur innerhalb von `<adresse>...</adresse>` angegeben werden dürfen. Der Titel wird in diesem Beispiel als optional festgelegt, was man mit der Angabe des Fragezeichens `?` hinter `titel` erreicht. Die anderen Tags `<name>`,

<postfach>, <strasse> und <wohnort> dagegen müssen immer innerhalb von <adresse>...</adresse> vorhanden sein, wobei allerdings nur eines der beiden Tags <postfach> oder <strasse> angegeben sein darf. Dazu wurden diese beiden alternativen Elementtypen geklammert und mit einem Senkrechtstrich | voneinander getrennt.
Eine Definition wie z. B.

```
<!ELEMENT name (#PCDATA)>
```

legt fest, dass es ein Element name gibt, das mit den Tags <name>...</name> eingegrenzt und dessen Inhalt als normaler Text interpretiert wird.
Kommentare lassen sich mit <!-- ... --> in der DTD einfügen.

18.1.3 Formatierung für XML-Elemente mit Style-Sprachen

XML-Dokumente enthalten lediglich so genannte semantische Markierungen. Eine Markierung wie z. B. <cite>...</cite> legt dabei nicht fest, wie der darin enthaltene Text anzuzeigen ist. In HTML dagegen würde der Browser bei einer Angabe wie <cite>...</cite> den darin angegebenen Text entsprechend hervorgehoben, wie z. B. kursiv, darstellen. In XML dagegen muss man mit Hilfe einer Style-Sprache festlegen, wie die Daten in den entsprechenden Tags zu formatieren sind.
Dazu stehen heute zwei Style-Sprachen zur Verfügung: CSS und XSL. CSS (Cascading Style-Sheets), die in Kapitel 17 ausführlich behandelt wurden, dienen lediglich dazu, festzulegen, wie der Browser die Elemente einer XML-Datei darstellen soll.
XSL (*Extensible Stylesheet Language*) auf der anderen Seite ist dagegen wesentlich mächtiger und direkt auf XML zugeschnitten.

18.2 Aufbau und Inhalt von XML-Dateien

In diesem Abschnitt wird der typische Aufbau von XML-Dateien gezeigt, und es werden wichtige Regeln vorgestellt, die in XML-Dateien einzuhalten sind.

18.2.1 Typischer Aufbau von XML-Dateien

Angaben am Anfang einer XML-Datei

Jede XML-Datei sollte sich am Anfang als XML-Datei identifizieren. Dies ist mit folgender einfacher XML-Deklaration am Beginn der Datei möglich:

```
<?xml version="1.0"?>
.....
```

Neben der Versionsangabe kann eine solche XML-Deklaration zwei weitere Attribute enthalten:

- encoding=
  legt verwendeten Zeichensatz fest.
- standalone=
  legt fest, ob die zugehörige DTD sich in dieser ("yes") oder aber in einer anderen Datei ("no") befindet

Tabelle 18.1: Angaben für das Attribut encoding=

encoding=	Zeichensatz
UTF-8	internationaler Zeichensatz auf Basis der ISO / IEC-10646-Norm mit 8 Bit Zeichenbreite
UTF-16	internationaler Zeichensatz auf Basis der ISO / IEC-10646-Norm mit 16 Bit Zeichenbreite (Voreinstellung)
ISO-8859-1	ISO-Zeichensatz für westeuropäische Sprachen
ISO-8859-2	ISO-Zeichensatz für osteuropäische Sprachen
ISO-8859-3	ISO-Zeichensatz für südeuropäische Sprachen
ISO-8859-4	ISO-Zeichensatz für nordeuropäische Sprachen
ISO-8859-5	ISO-Zeichensatz für kyrillische Sprachen
ISO-8859-6	ISO-Zeichensatz für arabische Sprachen
ISO-8859-7	ISO-Zeichensatz für griechische Sprache
ISO-8859-8	ISO-Zeichensatz für hebräische Sprache
ISO-8859-9	ISO-Zeichensatz für türkische Sprache
ISO-8859-10	ISO-Zeichensatz für nordische Sprache

Nachfolgend ist eine XML-Deklaration gezeigt, in der diese beiden Attribute angegeben sind:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
.....
```

Tabelle 18.1 zeigt mögliche Angaben beim Attribut encoding=.
Die beiden Attribute standalone= und encoding= sind zwar optional, müssen aber, wenn sie vorhanden sind, in folgender Reihenfolge angegeben werden:

```
version= encoding= standalone=
```

Verarbeitungsanweisungen in einer XML-Datei

Manchmal benötigt man innerhalb einer XML-Datei so genannte Verarbeitungsanweisungen (*processing instructions*) für die Programme, die die XML-Datei verarbeiten, wie z. B.:

```
<?xml version="1.0"?>
....
<?xml-stylesheet type="text/css" href="meinstil.css"?>
....
```

18.2.2 Dokumenttyp-Deklaration

Mit einer Dokumenttyp-Deklaration kann man zu einer DTD (*Document Type Definition*) den Bezug herstellen. Dabei gibt es zwei Möglichkeiten, die hier vorgestellt werden.

Dokumenttyp-Deklaration mit interner DTD

In diesem Fall befinden sich die DTD-Definitionen innerhalb der Dokumenttyp-Deklaration, wie z. B.:

```
<?xml version="1.0"?>
<!DOCTYPE adressen [
    <!ELEMENT adressen (adresse)*>

    <!ELEMENT adresse (titel?, name, (postfach | strasse), wohnort)>

    <!ELEMENT titel (#PCDATA)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT postfach (#PCDATA)>
    <!ELEMENT strasse (#PCDATA)>
    <!ELEMENT wohnort (#PCDATA)>
]>
```

Die Dokumenttyp-Deklaration beginnt mit `<!DOCTYPE...`. Danach folgt, durch ein Leerzeichen getrennt, der Name des Dokumenttyps (hier `adressen`). Der Name des Dokumenttyps muss dabei mit dem Namen des Dokument-Elements, also dem obersten Tag in der XML-Datei, identisch sein. Die eigentlichen DTD-Definitionen müssen dann nach dem Namen des Dokumenttyps in eckigen Klammern angegeben werden. Am Ende muss eine schließende eckige Klammer gefolgt von der abschließenden spitzen Klammer `>` der Dokumenttyp-Deklaration angegeben werden. Danach können dann die Daten entsprechend der DTD-Definitionen angegeben werden. Definitionen in der DTD werden in Kapitel 18.3 auf Seite 425 näher erläutert.

Dokumenttyp-Deklaration mit externer DTD

In diesem Fall befinden sich die DTD-Definitionen in einer anderen Datei, deren Pfadnamen (lokal oder im Web) man hier angeben muss, wie z. B.:

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

oder

```
<?xml version="1.0"?>
<!DOCTYPE adressen SYSTEM "http://www.xxx.de/meinxml/adressen.dtd">
....
```

oder

```
<?xml version="1.0"?>
<!DOCTYPE adressen PUBLIC "-//Vorlagen Uni//DTD adressen V 1.0//DE"
  "http://www.xxx.de/vorlage/adressen.dtd">
....
```

DTD-Dateien haben üblicherweise die Endung `.dtd`.
Die Dokumenttyp-Deklaration beginnt mit `<!DOCTYPE...`. Danach folgt, durch ein Leerzeichen getrennt, der Name des Dokumenttyps (hier `adressen`). Die nächste Angabe muss dabei entweder `SYSTEM` oder `PUBLIC` sein. Das Wort `SYSTEM`

muss man angeben, wenn man den Pfadnamen der DTD explizit angibt. Das Wort `PUBLIC` muss man angeben, wenn man den Pfadnamen der DTD nicht explizit, sondern durch einen so genannten öffentlichen Bezeichner (*public identifier*) angibt. Solche Angaben zu einem öffentlichen Bezeichner bestehen aus drei Teilen:

1. Angabe der veröffentlichenden Person oder Institution
2. Bezeichnung der DTD, beginnend mit DTD, gefolgt vom Namen der DTD und einer Versionsangabe
3. Sprache, in der die Namen der Elemente, Attribute usw. in der DTD definiert sind.

Hinter der Angabe zum öffentlichen Bezeichner kann man zusätzlich noch – wie im vorherigen Beispiel – die echte Webadresse der DTD angeben.

Kommentare

Kommentare lassen sich in DTDs – genauso wie in HTML-Dateien – mit `<!--....-->` angeben. Ein Kommentartext wird also mit der Zeichenkombination `<!--` eingeleitet und mit `-->` beendet. Ein Kommentartext darf sich dabei auch über mehrere Zeilen erstrecken, wie z. B.:

```
<!-- Der Titel kann, muss aber nicht angegeben werden und
     es ist entweder das Postfach oder die Strasse, aber
     nicht beides anzugeben                                  -->
<!ELEMENT adresse (titel?, name, (postfach | strasse), wohnort)>
```

Nach `<!--` und vor `-->` muss sich jeweils mindestens ein Leerzeichen oder ein Zeilenumbruch befinden. Innerhalb von Kommentaren sollte man nicht zwei Bindestriche `--` hintereinander angeben.

18.2.3 CDATA-Abschnitte

XML-Dokumente dürfen so genannte CDATA-Abschnitte enthalten, die nicht als eine XML-Angabe interpretiert werden. Anders als Kommentare werden solche Bereiche jedoch als eine normale Zeichenfolge ausgegeben. So würden z. B. die beiden folgenden Angaben:

```
<![CDATA[Dieser Text wird einfach als normale Zeichenfolge ausgegeben]]>
<![CDATA[<cite>Tags werden <b>nicht</b> ausgewertet]]>
```

zu folgender Ausgabe führen:

```
Dieser Text wird einfach als normale Zeichenfolge ausgegeben
<cite>Tags werden <b>nicht</b> ausgewertet
```

18.2.4 Allgemeine Regeln für Tags, Attribute und Wertzuweisungen

Für Tags, Attribute und ihre Wertzuweisungen gelten in XML folgende Regeln:

- **Namen dürfen nur bestimmte Zeichen enthalten**
  Namen für Tags, Attribute und Entities (siehe Kapitel 18.3.4 auf Seite 440) dürfen folgende Zeichen enthalten:

- Groß- und Kleinbuchstaben
- Ziffern von 0 bis 9
- Unterstriche (_)
- Bindestriche (-)
- Punkte (.)
- Doppelpunkt (:); der Doppelpunkt ist jedoch auch für Namensräume (siehe Kapitel 18.2.7 auf Seite 422) reserviert.

Weitere Regeln für Namen sind:

- Das erste Zeichen eines Namens darf keine Ziffer sein, sondern muss ein Buchstabe, ein Unterstrich oder eines der erlaubten Interpunktionszeichen sein. Grundsätzlich empfiehlt es sich aber, dass das erste Zeichen möglichst immer ein Buchstabe, eventuell auch ein Unterstrich ist.
- Namen dürfen keine Leerzeichen enthalten.
- Namen dürfen nicht mit der Zeichenfolge `xml` oder `XML` beginnen, da diese Zeichenfolgen für spätere XML-Versionen reserviert ist.
- Namen müssen mindestens einen Buchstaben bzw. Unterstrich enthalten.
- Die Länge von Namen ist praktisch unbegrenzt.

Nachfolgend sind einige erlaubte und einige unerlaubte Namen gezeigt:

```
<name2-xml> <!-- richtig -->
<z1>        <!-- richtig -->
<haus-hof>  <!-- richtig -->
<neu-zaehl> <!-- richtig -->
<neu zaehl> <!-- falsch (keine Leerzeichen erlaubt)      -->
<vor=nach>  <!-- falsch (Zeichen = nicht erlaubt)        -->
<1horn>     <!-- falsch (Ziffer am Anfang nicht erlaubt) -->
<xml-name>  <!-- falsch (xml am Anfang nicht erlaubt)    -->
```

- **XML unterscheidet zwischen Groß-/Kleinschreibung**
  Anders als HTML unterscheidet XML zwischen Groß- und Kleinschreibung. So handelt es sich z. B. bei den beiden folgenden Angaben um zwei verschiedene Tags:

```
<!ELEMENT zitat (#PCDATA)>
<!ELEMENT Zitat (#PCDATA)>
```

  Dasselbe gilt auch für Attribute, wo man ebenso genau darauf achten muss, wie der Attributname geschrieben ist. So spricht man z. B. bei `<adresse land="de">` ein anderes Attribut an als bei `<adresse Land="de">`. Ebenso wird bei Wertzuweisungen zwischen Groß- und Kleinschreibung unterschieden. So handelt es sich z. B. bei `<adresse groesse="riesig">` um eine andere Wertzuweisung als bei `<adresse groesse="Riesig">`.

- **Werte müssen bei Zuweisungen immer in "..." angegeben sein**
  Bei XML gilt grundsätzlich, dass alle Werte bei Zuweisungen an Attribute in Anführungszeichen stehen müssen, wie z. B.:

```
<adresse groesse="klein"> <!-- so muss es sein -->
<adresse groesse=klein>   <!-- falsch, da Wert nicht in Anführungszeichen -->
```

Benötigt man innerhalb einer Wertzuweisung an ein Attribut den Apostroph oder das Anführungszeichen, so müssen diese entsprechend kodiert angegeben werden, wie z. B. `&apos;` für Apostroph und `&quot;` für Anführungszeichen (siehe auch Tabelle 18.2 auf Seite 424).

- **Endtags dürfen niemals weggelassen werden**
  In XML darf niemals, wie z. B. bei den Tags `<div>` oder `<td>` in HTML, das zugehörige Endtag weggelassen werden.
  Ausnahmen sind so genannte leere Tags ohne Inhalt. Solche Elemente können ohne zugehöriges Endtag angegeben werden, müssen jedoch explizit als leer gekennzeichnet werden, wie z. B.:

```
<hr/>
```

  oder aber:

```
<hr />
```

  oder aber auch:

```
<hr></hr>  <!-- zwischen Anfangstag und Endtag darf kein Leerzeichen stehen -->
```

  Eine alleinige Angabe wie

```
<hr>
```

  ist nicht erlaubt.

18.2.5 Wohlgeformte und gültige XML-Dokumente

Wohlgeformtheit eines XML-Dokuments

„Wohlgeformtheit“ ist ein wichtiger Begriff in XML. Ein XML-Dokument ist *wohlgeformt*, wenn es die Regeln von XML genau einhält. So ist z. B. das folgende XML-Dokument wohlgeformt, da für es folgendes gilt:

- Es enthält am Anfang eine XML-Deklaration, die den Bezug zu XML herstellt.
- Es gibt ein äußerstes Element (auch als Dokument-Element bezeichnet), das alle anderen Datenelemente enthält (`<Maerchen>...</Maerchen>`).
- Es gibt mindestens ein Datenelement (im Beispiel sind es drei: `<Maerchen>...</Maerchen>`, `<Rotkappe>...</Rotkappe>` und `<Wolf>...</Wolf>`)

```
<?xml version="1.0"?>
<Maerchen>
   <Rotkappe stimme="hoch">Warum hast du einen so grossen Mund</Rotkappe>
   <Wolf stimme="tief">Damit ich dich bessser fressen kann</Wolf>
</Maerchen>
```

Dieses XML-Dokument ist zwar wohlgeformt, aber ungültig, da es keine DTD selbst enthält und auch keinen Bezug auf eine Datei angibt, in der sich eine DTD befindet.

Gültigkeit und Vollständigkeit eines XML-Dokuments

Das folgende XML-Dokument ist gültig:

```
<?xml version="1.0"?>
<!DOCTYPE adresse [
```

Kapitel 20

HTML/XHTML-Kurzreferenz

20.1 Alphabetische Übersicht der HTML-Tags

[1] Zur Erklärung der Nomenklatur siehe Kapitel 20.4 auf Seite 463

20.2 Struktur von HTML-Dokumenten

```
<html>
<head>
 .... Dokumentenkopf
</head>
<body>
 .... Eigentlicher Inhalt des Dokuments
</body>
</html>
```

20.3 Kommentare in HTML-Dokumenten

```
<!-- Kommentartext -->
```

Nach `<!--` und vor `-->` muss sich mindestens ein Leerzeichen oder ein Zeilenumbruch befinden. Der Kommentartext kann sich über mehrere Zeilen erstrecken. Der Internet-Explorer kennt noch das Tag `<comment>...</comment>`, um Kommentare in einem HTML-Dokument anzugeben.

20.4 Tags und Attribute in HTML/XHTML

Nomenklatur

In dieser Kurzreferenz wurde die folgende Nomenklatur gewählt:

(I) bedeutet, dass dieses Konstrukt nur im Internet-Explorer verfügbar ist
(N) bedeutet, dass dieses Konstrukt nur im Netscape Navigator verfügbar ist
(I,N) bedeutet, dass Konstrukt im Internet-Explorer und Netscape Navigator verfügbar ist

<tag>...</tag> zeigt an, dass hier das Endtag `</tag>` in HTML nicht angegeben werden muss
<tag>...</tag> zeigt an, dass hier das Endtag `</tag>` angegeben werden muss bzw. sollte
<tag>...<tag/> zeigt an, dass hier in HTML kein Endtag existiert; in XHTML ist hier entweder `</tag>` oder `<tag ... />` anzugeben

Bei Verweisen auf nähere Beschreibungen zu bestimmten Attributen ist die entsprechende Seitenzahl, auf der sich die genauere Beschreibung befindet, in Klammern angegeben.

Tags für die Struktur eines HTML-Dokuments

<html>...</html> *umrahmt ein HTML-Dokument*: `dir=` `lang=` `version=` (482)

<head>...</head> *umrahmt den Kopfteil eines HTML-Dokuments:* `dir=` `lang=` `profile=` (482)

<body>...</body> *beinhaltet den Dokumenteninhalt*

Farben (484): `bgcolor=` (Hintergrundfarbe), `text=` (Farbe für Text)
Farben für Links (484): `link=` (nicht besucht), `alink=` (aktiv), `vlink=` (besucht)
Hintergrundbild: `background=`*`url`*
`bgproperties="fixed"` (I) Feststehender Hintergrund beim Scrollen
`leftmargin=`*n* (I), `topmargin=`*n* (I) Rand links bzw. oben in Pixel
`class=` `style=` `id=` `title=` `dir=` `lang=` (482)
`onClick=` `onDblClick=` `onKeyDown=` `onKeyPress=` `onKeyUp=`
`onMouseDown=` `onMouseMove=` `onMouseOut=` `onMouseOver=` `onMouseUp=`
`onload` `onUnload=` (483)

<bdo>...</bdo> *Andere Sprache oder Schreibrichtung*:
`class=` `style=` `id=` `title=` `dir=` `lang=` (482)

```
<bdo dir=rtl>Von rechts nach links.</bdo> Ab hier wieder von links nach rechts
```

würde bei einem Browser, der das Tag `<bdo>` unterstützt, wie folgt ausgegeben:

```
.sknil hcan sthcer onV Ab hier wieder von links nach rechts
```

Tags für den Kopfteil eines HTML-Dokuments (in <head>...</head>)

<title>...</title> *Titel eines Dokuments*: dir=, lang= (482)

<base>...<base/> *Voreingestellte Pfad- und Framenamen für Dokument*

- href=*url* — Basis-URL für alle relativen URLs im HTML-Dokument
- target=*string* — Bei Hyperlinks / Formularen, bei denen Attribut frame= fehlt, wird Zieldokument bzw. Formular automatsch im Frame mit den hier angegebenen Namen angezeigt

<style>...</style> *definiert Style-Sheets für Dokument*: dir=, lang=, title= (482)

- type= — text/css (*Cascading Style Sheet*-Angaben); text/javascript (*Javascript Style Sheet*-Angaben)
- media= — Style-Sheets für unterschiedliche Ausgabemedien: screen (Bildschirm), print (Drucker), aural (Lautsprecher) projection (Overhead-Projektor), tv (Fernseher), handheld (Handys, Palmtops usw.), braille (taktile Braille-Medien) oder all (alle Medientypen)

<meta>...<meta/> *Zusatzinformationen zum HTML-Dokument*: dir=, lang= (482)

- name=*string* — Typ der Information; Typische Namen hier sind z. B.:
  - author, copyright — Name des Autors der HTML-Datei bzw. Copyright-Hinweise
  - keywords — Stichworte (mit Komma getrennt bei content=) für Suchmaschinen
  - description — Kurze Beschreibung des Dokuments; von Suchmaschinen angezeigt
  - date, language — Datum / Zeit der Erstellung des Dokuments bzw. verwendete Sprache
  - generator — Information zu dem Tool, mit dem Dokument erstellt wurde
  - classification — Klassifikation des Dokuments; auch als Kommentar interpretiert
  - expires — Angabe eines Datums, an dem Dokument ungültig wird
- http-equiv=*string* — Informationen, die Server im HTTP-Kopfteil beim Versenden einfügt
- content=*string* — eigentliche Information zu name= bzw. http-equiv=
- scheme=*string* — wählt Vorgabe von <meta>-Angaben aus. Diese Vorgabe sollte sich in Profile-Datei von <head profile=...> (siehe Seite 482) befinden
- charset=*string* (I) — Zeichensatz für Dokument; besser ist es, den Zeichensatz mit http-equiv= und content= festzulegen

<meta http-equiv="refresh" content="5">: aktuelle Dokument alle 5 Sekunden neu laden
<meta http-equiv="refresh" content="10; url=*webpage*">: nach 10 Sek. *webpage* laden

<link>...<link/> *Bezüge zu anderen Dokumenten*

- href= — legt Bezugsdokument fest
- rel=, rev= — rel= legt Beziehung zwischen aktuellen Dokument und Zieldokument fest. rev= legt Beziehung zwischen Zieldokument und aktuellen Dokument fest.
  - next, prev — nächstes bzw. vorheriges Dokument in einer Dokumentsammlung
  - head, toc — oberstes Dokument bzw. Inhaltsverzeichnis
  - parent, child — über- bzw. untergeordnetes Dokument
  - index, glossar — Stichwortverzeichnis bzw. Glossar dieses Dokuments
- title=*string* — Beschriftung, die Browser im entspr. Button anzeigen soll
- type=*string* — MIME-Typ des Zieldokuments; z. B. bei Bezügen auf Style-Dateien: <link href=meinstil.css rel=stylesheet type=text/css>
- target=*name* — Name des Fensters bzw. Frames, in dem Zieldokument anzuzeigen ist. Neben eigenen Namen existieren vordefinierte Namen, um Zieldokument anzuzeigen...
  - _self — im selben Fenster bzw. Frame (Voreinstellung)
  - _blank — in neuem, unbenamten Fenster
  - _parent — im übergeordneten Frame; entspricht _self, wenn Frame nicht geschachtelt
  - _top — im obersten Fenster und dort alten Inhalt ersetzen
- charset=*code* — Zeichensatzcode (z. B. iso-8859-1) des Zieldokuments
- hreflang= — Sprache des Zieldokuments (482)
- media=*liste* — Darstellung (mit Kommas getrennt) für Zieldokument: screen Bildschirm (Voreinst.), print (Drucker), aural (Lautsprecher) projection (Overhead-Projektor), tv (Fernseher), handheld (Handys, Palmtops oder ähnliche Geräte mit kleinem Display), braille (taktile Braille-Medien), all (alle Medientypen)

class= style= id= title= dir= lang= (482)
onClick= onDblClick= onKeyDown= onKeyPress= onKeyUp=
onMouseDown= onMouseMove= onMouseOut= onMouseOver= onMouseUp= (483)

❏ Bei <link>-Angaben, die im Kopfteil stehen müssen, sollte Browser Buttons einblenden, die Benutzer leicht in der Dokumentensammlung blättern lassen; in vielen Browsern nicht realisiert.

<isindex>...<isindex/> *Suchfunktion für ein Dokument (Veraltet)*

- prompt=*string* — Text vor Eingabefeld für Suchbegriffe statt „...Enter search keywords:“
- action=*url* (I) — URL für Suchindex; besser <base href=*url*> im Kopfteil

class= style= id= title= dir= lang= (482)

Tags zur Dokumentgestaltung und -formatierung

Absätze

** ... ** *division (Schachtelung möglich)*

** ... ** *paragraph*

align= Text ausrichten: left (linksbündig; Voreinst.); right (rechtsbündig); center (zentriert); justify (justiert); von vielen Browsern nicht unterstützt
class= style= id= title= dir= lang= (482)
onClick= onDblClick= onKeyDown= onKeyPress= onKeyUp=
onMouseDown= onMouseMove= onMouseOut= onMouseOver= onMouseUp= (483)

- Eine Schachtelung von -Tags ist erlaubt.
- Internet-Explorer kennt noch Attribut <nowrap>. Bei <div nowrap> finden nur dort Zeilenumbrüche statt, wo auch im HTML-Dokument Zeilenvorschübe (*Carriage Return*) angegeben sind.
- Innerhalb von ... sind folgende Tags erlaubt: Verweise (<a>), Bilder (<img>), Zeilenvorschübe (<br>) und Texthervorhebungen (, , <cite>, <var> usw.; siehe *Texthervorhebungen*). Werden andere Tags angegeben, beendet der Browser den Absatz.

Überschriften

<h1>,<h2>,<h3>,<h4>,<h5>,<h6>...</h1>,</h2>,</h3>,</h4>,</h5>,</h6>

align= Überschrift ausrichten: left (linksbündig; Voreinst.); right (rechtsbündig); center (zentriert); justify (justiert); von vielen Browsern nicht unterstützt
class= style= id= title= dir= lang= (482)
onClick= onDblClick= onKeyDown= onKeyPress= onKeyUp=
onMouseDown= onMouseMove= onMouseOut= onMouseOver= onMouseUp= (483)

- In Überschriften können u.a. folgende Tags angegeben werden: Verweisziele (<a>), Bilder (<img>), Zeilenvorschübe (<br>) und Texthervorhebungen (, , <cite>, <var> usw.

Formatieren einzelner Textabschnitte

- **Feste Formatierung** (Endtags müssen immer angegeben werden)

, , <u>	Fetter, kursiver bzw. unterstrichener Text	**, , </u>**
<big>, <small>, <tt>	Größerer, kleinerer bzw. nicht proport. Text	**</big>, </small>, </tt>**
_{, ^{	Tiefer bzw. höher gestellter Text	**},}**
<blink> (N)	Blinkender Text	**</blink>**
<s>, <strike>	Durchgestrichener Text (Veraltet)	**</s>, </strike>**

- **Logische Formatierung** (überlassen Browser die Darstellung; Endtags müssen immer angegeben werden)

<abbr>, <acronym>	Abkürzung bzw. Akronym	**</abbr>, </acronym>**
<cite>	Zitat innerhalb einer Textpassage	**</cite>**
<code>, <samp>	Darstellung von Code (nicht proportionale Schrift)	**</code>, </samp>**
<dfn>, <kbd>	Begriffe (*definition*) bzw. Tastatureingaben (*keyboard*)	**</dfn>, </kbd>**
**, **	Einfache (*emphasis*) bzw. starke Betonung	**, **
<var>	Variablennamen	**</var>**

- **Benutzerdefinierte Formatierung**

... *Benutzerdefinierte Formatierung*

class= style= id= title= dir= lang= (482)
onClick= onDblClick= onKeyDown= onKeyPress= onKeyUp=
onMouseDown= onMouseMove= onMouseOut= onMouseOver= onMouseUp= (483)

Zeilenumbrüche

**
...
** *Zeilenumbruch*: class=, style=, id=, title= (482)
clear=*richtung* [left|right|all] (482)

<nobr>...</nobr> *Zeilenumbruch verhindern (I, N)*
Bei Text, der sich in <nobr>...</nobr> befindet, findet kein automatischer Zeilenumbruch statt.

<wbr>...<wbr/> *Zeilenumbruch nur bei überlangen Zeilen (I, N)*

- Bei Texten in <nobr>...</nobr> können Zeilenumbrüche immer mit Tag <br> veranlaßt werden. <wbr> bewirkt – anders als
 – in <nobr>...</nobr> nicht unbedingt einen Zeilenumbruch, sondern nur, wenn die Zeile länger als die Breite des Browserfensters ist.

20.5 Farben in HTML-Dokumenten

Farben können in HTML-Dokumenten auf zwei verschiedene Arten angegeben werden: als Farbname (wie z. B. `red`, `yellow`, `green` usw.) oder als sechsstellige Hexadezimalzahl, die den so genannten RGB-Wert festlegt.

RGB-Farben

Um eine Farbe in einem HTML-Dokument anzugeben, kann man statt eines vordefinierten Farbnamens (wie z. B. `red`, `blue` oder `white`) auch den Wert der Mischfarbe explizit angeben. Ein solcher Farbwert ist als Hexadezimalzahl `#rrggbb` (mit sechs Hexadezimalziffern für den 24-Bit Wert) anzugeben. Für die beiden Ziffern `rr` ist also der Beitrag der Farbe rot zur Mischfarbe im Bereich von 0 bis 255 (entspricht 00 bis FF im Hexadezimalsystem) anzugeben. Dasselbe gilt für `gg` (Beitrag von Grün) und `bb` (Beitrag von Blau). So legt z. B. der Wert `#00FF00` die Farbe Grün und der Wert `#0000FF` die Farbe Blau fest.

Vordefinierte Farbnamen

Abbildung 20.1 zeigt die 16 in HTML / XHTML vordefinierten Farbennamen.

black	#000000	gray	#808080
maroon	#800000	red	#FF0000
green	#008000	lime	#00FF00
olive	#808000	yellow	#FFFF00
navy	#000080	blue	#0000FF
purple	#800080	fuchsia	#FF00FF
teal	#008080	aqua	#00FFFF
silver	#C0C0C0	white	#FFFFFF

Abbildung 20.1: Vordefinierte Farbnamen mit ihren RGB-Werten

Standard-Farben

Auf Systemen, die nur 256 Farben anzeigen können, werden Farben, die nicht darstellbar sind, entweder in eine auf dem System darstellbare Farbe konvertiert oder es wird ein Verfahren namens *dithering* angewendet, bei dem die angeforderte Farbe durch nebeneinanderliegende Pixeln mit vorhandenen Farben versucht wird, nachzubilden. Beim *dithering* muss man natürlich Qualitätsverluste in Kauf nehmen. Bei der Konvertierung dagegen wird eine existierende Farbe genommen, die der angeforderten Farbe am nächsten liegt.

Die Standard-Farbpalette umfasst 216 Farben, wobei jeweils sechs Varianten für Rot, Grün und Blau angeboten werden, die beliebig kombiniert werden können, so dass sich die Zahl 216 (6 x 6 x 6) ergibt. Abbildung 20.2 zeigt die Standard-Farbpalette der 216 Farben. Die sechs Varianten jeder RGB-Komponente haben dabei die hexadezimalen Werte `00`, `33`, `66`, `99`, `CC` und `FF`, was den dezimalen Werten `0`, `51`, `102`, `153`, `204` und `255` entspricht. Farben wie `006666` (dunkelgrün) und `FF66FF` (lila) existieren direkt in der Standard-Farbpalette und müssen weder konvertiert noch mit dem *dithering*-Verfahren nachgebildet werden. Es ist also empfehlenswert, Farben aus dieser Standard-Farbpalette zu benutzen.

Abbildung 20.2: Standard-Farbpalette der 216 Farben

Index

Z