FernUniversität
Gesamthochschule in Hagen

Universitätsrechenzentrum

Thomas Feuerstack
Jens Vieler
Beratung und IT-Services

Das XML-Cocktailbuch

B/016/0210

Inhaltsverzeichnis

1 Aperitif

Manhattan (Aperitif)

1dash: Angostura Bitter
4cl: Canadian Whisky
2cl: Vermouth Rosso

Dekoration
Cocktailkirsche

Mixanleitung
Alle Zutaten im Mixglas mit viel Eis verrühren. Danach in eine gut gekühlte Cocktailschale abseihen und mit der Cocktailkirsche garnieren.

1.1 Wie dieses Kapitel zu seinem Namen kam

ist eigentlich noch die einfachste der zu diesem Zeitpunkt zu beantwortenden Fragen. Eine Broschüre die den Namen *Das XML-Cocktailbuch* trägt – und sich damit in doppelter Hinsicht mit hochgeistigen Genüssen beschäftigt – kann folgerichtig nur mit einem Aperitif beginnen.

Die Frage müsste daher eigentlich lauten *Wie diese Broschüre zu ihrem Namen kam* und eine Antwort darauf dürfte ungleich schwerer zu geben sein. Bekanntermassen lässt sich der Funke der Inspiration, sofern er denn erstmal gezündet hat, schwerlich auf seinen Auslöser zurückführen – eines Morgens wacht man auf und weiß mit Sicherheit, dass man demnächst ein XML-Cocktailbuch schreiben wird.[1]

Für den Fall, dass Sie zu dem Menschenschlag gehören, der immer alles ganz genau wissen muss, können Sie sich eventuell aus den folgenden Anhaltspunkten etwas zusammenbasteln.

- Wer uns (in diesem Fall die Autoren) kennt weiß, dass uns die Erstellung einer rein technisch orientierten Broschüre viel zu dröge ist.
- Ganz nebenbei existieren für unseren Geschmack bereits viel zu viele *XML-Einführungen*, *XML-HowTo's*, *XML für Dummies* und was der Markt traditionell sonst noch für den/die vermeintlich Unbedarfte(n) hergibt.

[1] Oder haben Sie jemals eine rationale Begründung für die Entstehung von Beethovens 9. Sinfonie gehört?

- Wie bei jedem Rezeptbuch soll der Inhalt natürlich Geschmack auf mehr machen; Sie sollten daher tunlichst daran gehen, die beschriebenen Inhalte selbst auszuprobieren.
- Last not least, für die ökonomisch denkenden LeserInnen: Sollte Ihnen der Inhalt dieser Broschüre nicht zusagen, so können Sie wenigstens noch die abgedruckten Rezepte verwerten – die Anschaffung hat sich also auf alle Fälle gelohnt.

1.2 Wer sollte diese Broschüre lesen?

Prinzipiell alle, die schon immer mal wissen wollten, was es denn de facto mit der neuen(?) „Wunderwaffe“ XML so auf sich hat. Sollten Sie darüberhinaus noch ein paar nette Ideen für Ihre nächste Gartenparty suchen, so haben Sie gewissermaßen in einem Abwasch die Gelegenheit, das Angenehme mit dem Nützlichen zu verbinden.

Fans von Hochglanzwerbematerial werden dagegen eher nicht auf ihre Kosten kommen. Zum einen gestatten wir uns auf den kommenden Seiten auch durchaus kritische Anmerkungen zu XML, zweitens können wir uns die umfeldspezifischen Schlagwörter meist selbst nicht merken, und – last not least – würde eine Reproduktion dieser Broschüre auf Hochglanz unseren mageren Universitätshaushalt bei weitem übersteigen.

In diesem Sinne, der Manhattan ist angerichtet – Prost!

2 Bargeflüster

Troubleshooter (Longdrink)

4cl:	Asbach Uralt
1:	Limette
1EL	Rohrzucker, braun
1/2	Zitrone
	Fanta Pink Grapefruit
	Crushed Ice

Dekoration
Zwei Strohhalme in das Glas lehnen.

Mixanleitung
Die Limette in Achtel schneiden und in einem Longdrinkglas, beispielsweise mit Hilfe eines Caipirinha-Stößels, zerstoßen.
Den Rohrzucker und den Zitronensaft zugeben und das Glas zu knapp 3/4 mit dem gestoßenen Eis auffüllen. Mit Asbach Uralt und Pink Grapefruit bis zum Rand auffüllen.

2.1 Aufgeschnappte Satzfragmente

Je nach baulicher Beschaffenheit der von Ihnen gewählten Bar und meistens auch abhängig von dem Umstand, ob Sie allein oder in Begleitung gekommen sind, ist es beinahe unvermeidlich, dass Sie an den Gesprächen der anderen Gäste, nun sagen wir einmal „teilhaben“.

Meist sind es nur Satzfragmente, die wellenartig zu Ihnen durchdringen; dennoch kann der Inhalt umso interessanter sein. So erzählte neulich, meiner ungewollten „Teilhaberschaft“ überhaupt nicht gewahr, eine Hagenerin ihrer Begleitung, dass ihr Schwager (?) seinen diesjährigen Urlaub „irgendwo auf dem Balkan in Reykjavik“ verbracht habe.[1]

In Bezug auf XML sind die im Umlauf befindlichen Gerüchte kaum weniger abenteuerlich. Aussagen die von *XML als neuer Internetsprache* künden, häufig gekoppelt mit dem Zusatz, dass *es mittlerweile unnötig ist, HTML zu lernen*, sind dabei die eher harmloseren Vertreter ihrer Art.

Daneben finden sich auch gern Zitate, die von der beinahe *grenzenlosen Formatierbarkeit eigener Dokumente* berichten, bis hin zu der Tatsache, dass auch das leidige *Austauschen verschiedenster Dokumentformate* durch XML endlich das ersehnte positive Ende gefunden hat.

[1] Zu ihrer Ehrenrettung sei gesagt, dass der schwägerliche Urlaubsort später dann doch glasweise Stück für Stück weiter nach Norden verlegt wurde.

Als Fazit lässt sich feststellen, dass *ohne XML zukünftig wohl nichts mehr geht*, oder plakativer:

XML ist die Lösung für alle Probleme überhaupt.

Erstaunlich, nicht wahr? Denn als Kunde einer konsumorientierten Umwelt sollten Sie eigentlich genug Erfahrung gesammelt haben um zu wissen, dass bei extrem angepriesenen Super-Angeboten der Pferdefuß schon irgendwo versteckt sein wird. Mit knapp zwanzigjähriger Berufserfahrung können wir zudem beisteuern, dass wir in jedem dieser Jahre mindestens eine „Lösung für alle Probleme“ haben kommen (und auch wieder gehen) sehen.

Deshalb erstmal keine Panik, gemeinsam werden wir den Cocktail schon schütteln. Nippen wir also nochmal am Troubleshooter und raisonieren dabei: Was ist nun dran an/drin im XML?

2.2 XML — die Zutatenliste

Schaut man sich das Konglomerat XML etwas genauer an, so wird man schnell feststellen, dass das Rad bei weitem nicht neu erfunden wurde, im Gegenteil: das meiste wird Ihnen wahrscheinlich ziemlich bekannt vorkommen. Konkret ausgeführt bedeutet dies:

- Die Abkürzung *XML* steht für E*X*tensible *M*arkup *L*anguage, also auf Deutsch: erweiterbare Auszeichnungssprache.
- Durch XML sollen ASCII-, also textzeichenbasierte Dokumente, mit einer Struktur versehen werden. Dies geschieht (Stichwort: *Markup Language*) durch Auszeichnungen in Form von Start- und Ende-Tags – und sollte bereits jetzt zu Deja-vu-Effekten führen.

  Erstes Beispiel: `Wir nehmen den <Cocktail>Troubleshooter</Cocktail> gerührt und nicht geschüttelt.`

→Kapitel 2.3 auf der nächsten Seite

- XML ist eine Meta-Auszeichnungssprache, über die Elemente anderer Auszeichnungssprachen definiert werden können. Hierzu werden die benötigten Elementnamen, deren Inhaltsstruktur, Attribute, etc. festgelegt und können darüberhinaus wahlweise in einer Regeldatei gespeichert werden.

! →

- Die Darstellung, Verarbeitung und Verknüpfung von Daten *gehört dagegen nicht zur Aufgabe* von XML – und ist damit auch nicht Inhalt dieser Broschüre!

  Entsprechende Umsetzungen werden durch Co-Standards, die unmittelbar auf XML aufbauen, wie XSL, XPath, XLink oder im simpelsten Fall CSS realisiert.

- Strukturierung und Auszeichnung von Daten ist beileibe nichts Neues. Im Gegensatz zu proprietären Lösungen, beispielsweise Microsofts Word-Format oder Pagemaker, ist XML ein offener,

gut dokumentierter Standard, der von jedem eingesehen werden kann.

! → Fazit: XML ist daher gut für einen Datenaustausch *geeignet* – was aber auf keinen Fall damit gleichbedeutend ist, dass für diesen Zweck auch geeignete Werkzeuge oder Anwendungen existieren!

2.3 Verwandtschaften: HTML, SGML, XHTML, . . .

Eines der am weitest verbreitetsten Vorurteile besagt, XML wäre der letztendliche Nachfolger von HTML und würde dieses in naher Zukunft konsequenterweise ersetzen. Was ist davon zu halten?

Der Blick auf die Verwandschaftsverhältnisse zeigt, dass HTML eine speziell für das WWW konzipierte *Anwendung* von SGML[2] ist, das heißt, es wurde auf Grundlage der für SGML gültigen Auszeichnungsregeln definiert – in Analogie dazu können Sie auch von Ihren später erzeugten XML-Dokumenten als *Anwendungen* sprechen.

Im Gegensatz dazu ist XML hingegen eine Seiten-(Weiter-?)Entwicklung von SGML und zu dieser voll kompatibel. Ein Vergleich zwischen XML und HTML dürfte daher ausgehen wie die klassische Vorlage mit den Äpfeln und Birnen.

Wenn also etwas mit HTML verglichen werden kann, so muss dies auf der Ebene der Anwendungen geschehen. Eine Anwendung die sich dafür anbieten würde, also eine von XML, ist beispielsweise XHTML – Abbildung 2.1 zeigt dies noch einmal übersichtlicher.

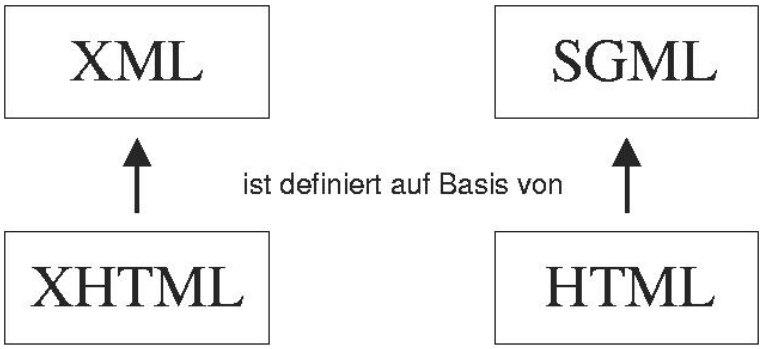


Abbildung 2.1: XML und seine „Verwandtschaftsgrade“.

In Analogie dazu muss sich XML also mit SGML messen. Prinzipiell sind sich beide Sprachen sehr ähnlich, jedoch sind die Möglichkeiten von XML bewusst eingeschränkt worden – ohne dabei allerdings wesentliche Vorteile zu verlieren; zumindest sagt man so. XML ist durch diese Maßnahme robuster als SGML; darüberhinaus sind Programme zur Verarbeitung von XML-Dokumenten, beziehungsweise XML-Anwendungen, einfacher zu erstellen.

[2] SGML = *S*tandard *G*eneralized *M*arkup *L*anguage

Dazu noch eine Bemerkung am Rande: Gerade dieser letzte Punkt sorgt unter XML-Autoren häufig für Unmut. Prinzipiell soll es ja der Computer sein, der die Arbeit des Anwenders erleichtert. Bei der Erstellung von XML-Dokumenten hat jedoch, aufgrund der im Vergleich zu SGML eingeschränkten XML-Strukturen, in jedem Fall der Autor mehr (Schreib-)Arbeit zu leisten.

2.4 Aufgaben zur Selbstkontrolle

Versuchen Sie, möglichst ohne Zuhilfenahme der Unterlagen, die folgenden Fragen zu beantworten, bzw. die Sätze zu vervollständigen:

1. XML ist die Abkürzung für:
   - (a) ☐ Standard Generalized Markup Language
   - (b) ☐ Extensible Markup Language
   - (c) ☐ Hypertext Markup Language
   - (d) ☐ nichts von allem
2. Welche der folgenden Aufgaben wird nicht durch XML gelöst:
   - (a) ☐ Strukturierung von Datenmengen
   - (b) ☐ Auszeichnung von Dokumenten
   - (c) ☐ Formatierung von Dokumenten
   - (d) ☐ Vorbereitung von Dokumenten zum Austausch zwischen verschiedensten Anwendungen
3. Welche der folgenden Aussagen ist richtig:
   - (a) ☐ XML ist eine Erweiterung von HTML
   - (b) ☐ XML ist eine Anwendung von XHTML
   - (c) ☐ XML ist flexibler als SGML
   - (d) ☐ XML ist die pragmatische Variante von XHTML
   - (e) ☐ Nichts davon ist richtig

Eingaben löschen | Aufgaben lösen

3 Was alles in den Shaker kommt

Angel's Face (Cocktail)

1cl: Apricot Brandy
1cl: Calvados
3cl Gordon's Dry Gin
Eiswürfel

Mixanleitung
In den Shaker einige Eiswürfel geben und den Gin, Calvados und Brandy dazu. Kurz schütteln und in ein Cocktailglas abseihen. Schmeckt ganz sanft – verleiht aber nach ein paar Gläsern Flügel.

Farben und ihre Bedeutung

Bis auf wenige Ausnahmen ist ein Cocktail eigentlich immer eine sehr farbenfrohe Angelegenheit. Dies liegt nicht nur daran, dass sich das Wort „Cocktail" mit „Hahnenschwanz" – der sich ebenfalls häufig stark kolo-riert darstellt – übersetzen lässt: spätestens nach zwei bis drei Gläsern wird auch die eigene Umgebung merklich bunter.

Aus diesem Grund, und nicht zuletzt wegen der besseren Übersichtlich-keit, haben wir versucht, das „Prinzip Farbe" in diese Broschüre zu in-tegrieren. Wenn Sie daher im Folgenden auf farblich unterlegte Begriffe stoßen, so bedeutet dies im Einzelnen:

Blau : kennzeichnet Marken, Attribute, Anweisungen, etc. die sich im *Wurzelelementbereich* eines XML-Dokuments befinden.

Grün : hat prinzipiell die gleiche Bedeutung wie Blau für den *Prolog* ei-nes Dokuments.

Orange : ist zu behandeln wie Blau und Grün, die betroffenen Elemente be-finden sich jedoch im Bereich *Diverses*. Das Rätsel um die Be-griffe Wurzelelementbereich, Prolog und Diverses wird direkt im folgenden Kapitel aufgelöst.

Rot : hat dafür generell zwei Bedeutungen. Zu einen haben wir damit *fehlerhafte Anweisungen* oder *Befehle* markiert. In der PDF-Versi-on dieser Broschüre sind die restlichen rot markierten Textstellen, zum Beispiel das Inhaltsverzeichnis, anklickbar. Wir hoffen, dass aus dem jeweiligen Umfeld ersichtlich ist, ob es sich um einen Fehler oder eine anklickbare Stelle handelt.

Magenta : ist nicht nur für die Deutsche Telekom reserviert. In dieser Farbe hinterlegte Textstellen können ebenfalls angeklickt werden und führen direkt zu externen Web-Adressen oder Download-Bereichen.

Die Leidtragenden dieses Verfahrens sind wahrscheinlich diejenigen, die dieses Skript monochrome gedruckt über den Dispatch bestellt haben, anstatt es farbig und umsonst aus der Leseecke zu ziehen. Nichtsdestotrotz haben wir uns bemüht,[1] die aus einer Schwarzweißdarstellung bedingten Benachteiligungen so gering, wie möglich zu halten.

3.1 Aufbau eines XML-Dokuments

→Beispiel 3.1 auf der nächsten Seite

Der rein formale Aufbau eines XML-Dokuments lässt sich wohl am Einfachsten anhand eines Beispiels verdeutlichen – schauen wir daher den Engeln ins Gesicht und uns den Angel's Face in XML-Notation an.

Die generelle Struktur eines XML-Dokuments unterteilt sich in die drei Sektionen Prolog, Wurzelelementbereich und Diverses.

Innerhalb jeder Sektion können *Markups*, im Allgemeinen Start- und Ende-Tags sowie Entitäten, als auch *Character Data* enthalten sein. Bevor es deshalb weitergeht, sollten wir sicherheitshalber noch detaillierter auf die oben vorgestellten und für Sie wahrscheinlich neuen Begriffe eingehen.

Markups : Wie Ihnen vielleicht schon aufgefallen ist, werden Auszeichnungen in XML durch das sogenannte Markup-Prinzip realisiert. Dies bedeutet, dass der auszuzeichnende Text jeweils durch eine öffnende und schließende Marke umklammert ist.

Anstelle von „Marke" wird auch häufig der Begriff „Tag" verwendet. Der Dokumentbereich von einer öffnenden bis hin zur schliessenden Marke wird als *Element* bezeichnet.

Entität : Vorsichtig ausgedrückt übernimmt eine Entität (oder Entity) die Funktion eines Textbausteins, d.h. die im Dokument vorhandene Zeichenkette wird durch einen Text ersetzt, der ansonsten nicht oder nur umständlich dargestellt werden könnte.

Da der obere Satz immer noch sehr kompliziert klingt, hier ein Beispiel: Das bekannte „Kaufmanns-Und" (&), wird sowohl in HTML als auch in XML durch die Entität `&` dargestellt – Kapitel 3.5 wird sich noch ausführlicher mit dem Thema Entitäten beschäftigen.

Character Data : Als Character Data wird nun folgerichtig alles bezeichnet, was nicht in die Bereiche Markup oder Entität gehört – also im Normalfall Text.

[1] Ein nachträglicher Dank geht hiermit an unsere Deutsch-Lehrerinnen und -Lehrer.

```
<?xml version='1.0'?>
<!DOCTYPE bar SYSTEM "bar.dtd">
<!-- Willkommen in der XML-Bar -->
<bar>
 <cocktail>
  <name>Angel's Face</name>
  <zutat>
   <substanz>Apricot Brandy</substanz>
   <menge>1cl</menge>
  </zutat>
  <zutat>
   <substanz>Calvados</substanz>
   <menge>1cl</menge>
  </zutat>
  <zutat>
   <substanz>Gordon's Dry Gin</substanz>
   <menge>3cl</menge>
  </zutat>
  <zutat>
   <substanz>Eiswürfel</substanz>
   <menge />
  </zutat>
  <mixanleitung>In den Shaker einige Eiswürfel geben...
  </mixanleitung>
 </cocktail>
</bar>
<!-- Innerhalb von (der?)  <bar> können
weitere Cocktails folgen -->
```

Beispiel 3.1: Angel's Face – diesmal ganz in XML

3.2 Elementinhalte

Im Gegensatz zu seiner Bedeutung beginnen wir bei der detaillierten Analyse des XML-Cocktails nicht mit dem Prolog, sondern wenden uns zuerst dem Bereich des Wurzelelements zu, da dieser den eigentlichen Datenbestand enthält. Denjenigen unter Ihnen, die HTML nicht nur in Zusammenhang mit Spezialeditoren kennen, werden durchaus ähnliche Züge entdecken; der Ordnung halber gibt es aber auch hier nochmal den strukturierten Überblick.

Wie oben bereits gezeigt, besteht ein Element normalerweise aus Character Data, welches durch Marken geklammert ist.

```
<name>Angel's Face</name>
```

Natürlich können Elemente auch verschachtelt werden, wobei Sie jedoch

auf die genaue Verschachtelungsreihenfolge zu achten haben.

```
<zutat>
 <substanz>Apricot Brandy</substanz>
 <menge>1cl</menge>
</zutat>
```

Eine besondere Bedeutung hat das *leere* Element, für das in XML zwei Darstellungen existieren. Alternativ zu einem Element ohne Character Data

```
<substanz>Eiswürfel</substanz>
<menge></menge>
```

wird häufig, wie in Beispiel 3.1, die Kurzschreibweise „`<menge />`“ gewählt.[2]

! → Ebenfalls erlaubt, aber äußerst unschön, ist die Mischung von Character Data und weiteren Elementen, beispielsweise

```
<zutat>
Beim Abmessen der Zutaten ist auf
äußerste Genauigkeit zu achten!
 <substanz>Apricot Brandy</substanz>
 <menge>1cl</menge>
</zutat>
```

Da eine solche Vermixung zwar den Geschmack des Cocktails steigern kann, auf keinen Fall jedoch die Lesbarkeit Ihres XML-Dokuments, sollten Sie auf Konstrukte dieser Art besser verzichten.

3.3 Marken in XML und HTML...

Wie im letzten Kapitel gezeigt, orientiert sich das Handling der Marken oder Tags stark an HTML. Höre ich da irgendwo einen leisen Widerspruch? Richtig – sofern Sie diese Broschüre aufmerksam verfolgt haben wissen Sie bereits, dass es eigentlich umgekehrt ist.

→Abbildung 2.1 auf Seite 5 Da HTML eine *Anwendung* von SGML ist, welche wiederum als der ältere Bruder von XML angesehen werden kann, geben XML, bzw. SGML die Regeln vor, woran sich die Anwendungen zu orientieren haben.

Schauen wir nochmal genauer hin:

```
<cocktail>
 <name>Angel's Face</name>
</cocktail>
```

[2] In HTML werden leere Elemente häufig durch eine einzelne, öffnende Marke (Beispiel: `<HR>`) dargestellt. Wir wir noch sehen werden, ist dies nach den Konventionen von XML nicht gestattet.

...Gemeinsamkeiten...

Wie bereits mehrfach erwähnt, existiert zu jeder öffnenden Marke (`<cocktail>`) ein schließendes Äquivalent (`</cocktail>`) mit führendem „/“. Natürlich ist auch die Schachtelung von Marken erlaubt, wobei das übliche „Last-in-First-Out“-Prinzip gilt. Das heisst, dass die zuletzt geöffnete Marke als erste wieder geschlossen werden muss; Markenüberschneidungen sind verboten.

...und Unterschiede

Da XML keine eigene Anwendung ist, sondern Sie ja selbst mit Ihrem Dokument eine Anwendung *auf* XML definieren, sind Ihrer Kreativität bei der Auswahl Ihrer Markennamen kaum Grenzen gesetzt.[3]

Unbedingt beachten, da von HTML wahrscheinlich eher ungewohnt, sollten Sie jedoch die folgenden Punkte:

! →

- Jede XML-Marke *muss* ein schließendes Äquivalent besitzen. Ein Ausnahme bilden lediglich leere Elemente, die durch die Sonderform `<marke />` dargestellt werden können.
- Groß-/Kleinschreibung wird strikt unterschieden. Ein Element

  ```
  <nAmE>Angel's Face</NaMe>
  ```

  ist fehlerhaft, weil hier, wie bereits oben erwähnt, zu einem öffnenden Tag kein schliessender existiert.
- Innerhalb eines XML-Dokuments existiert genau ein (!) Wurzelelement, welches sämtliche weiteren Elemente enthält und dessen Name nicht mehrfach verwendet werden darf. In unserem Beispiel ist dies das Wurzelelement **`<bar>`**.

→Beispiel 3.1 auf Seite 11

- Leerraum[4] zwischen der öffnenden Klammer und dem Markennamen ist verboten. Ein Leerraum vor dem abschließenden „>“ ist dagegen erlaubt.

```
<!-- Der öffnende Tag ist fehlerhaft...-->
< name>Angel's Face</name >
<!-- ...der schließende dagegen o.k. -->
```

[3] Die wenigen festgelegten Namensbeschränkungen werden ein paar Seiten später aufgeführt, da sie auch Attribute und Entitäten betreffen.

[4] Unter Leerraum werden die Zeichen verstanden, die bei der Auswertung des XML-Dokuments gewissermassen überlesen, bzw. auf ein *Leerzeichen* verkürzt werden. Dazu gehören das Leerzeichen (Blank) selbst, der Tabulator, Zeilenwechsel (LF), sowie der Wagenrücklauf (CR).

3.4 Attribute

Wie nicht anders zu erwarten, können natürlich auch XML-Marken durch Attribute erweitert bzw. variiert werden.

Als Beispiel ergänzen wir unser Element `<name>` durch das Attribut `klassifizierung`, welches Werte wie Longdrink, Cocktail, Fizz, u.v.a.m. annehmen soll, damit jeder Cocktail einer bestimmten Sparte zugeordnet werden kann.

```
<bar>
 <cocktail>
  <name klassifizierung='Cocktail'>Angel's Face</name>
  <zutat>
  ...
</bar>
```

Auch die Verwendung von Attributen ist bestimmten Regeln unterworfen, die wir Ihnen an dieser Stelle daher auch nicht vorenthalten wollen.

- Die Angabe von Attributen ist nur in einer *öffnenden Marke* zulässig.
- Werden mehrere Attribute gesetzt, so ist die Reihenfolge beliebig.
- Attribute *müssen* zwingend mit Werten versehen werden; Attributwerte werden durch Quotes (") oder Apostrophe (') geklammert.
- Vor oder nach dem Gleichheitszeichen darf Leerraum stehen.
- Werden einem Attribut mehrere Werte zugeordnet, so sind diese mit Leerzeichen voneinander zu trennen.
- Quotes und Apostrophe dürfen alternativ geschachtelt werden.

```
<name klassifizierung='Cocktail "besser Shortdrink"'>
Angel's Face</name>
```

→Kapitel 3.5 auf der nächsten Seite

- Die Verwendung des „<"-Zeichens ist im Attributwert verboten. Das „&" ist ausschließlich in Verwendung mit Entitäten gestattet.

  Soll das „&" im Attributwert als „Und-Zeichen" benutzt werden, so existiert für diesen Zweck die Entität `&amp;`. Beispiel:

```
<!-- Falsch! -->
<name erfinder='Feuerstack & Vieler'>
Angel's Face</name>
<!-- Richtig! -->
<name erfinder='Feuerstack &amp; Vieler'>
Angel's Face</name>
```

- Die mehrfache Angabe des gleichen Attributs ist nicht erlaubt.

```
<name preis='DM 15,00' preis='€ 9,00'>
Angel's Face</name>
```

3.5 Warum Character Data nicht einfach Text ist

Wir haben bereits gelernt, dass ein Elementinhalt nicht einfach als Text sondern als Character Data bezeichnet wird. Hintergrund des Ganzen ist nicht (nur), dass wir eine weitere Fachvokabel einführen wollten; in der Tat können einzelne Zeichen ein auf den ersten Blick etwas befremdliches Aussehen haben.

Beginnen wollen wir dabei mit den Zeichen, die logischerweise maskiert werden müssen weil sie generell innerhalb von XML für andere Aufgaben vorgesehen sind. Dies sind im Einzelnen:

Zeichen	eigentliche Bedeutung	Ersatzdarstellung (Entität)
<	Markenbeginn	<
>	Markenende	>
&	Entitätsbeginn	&
"	Attributwert	"
'	Attributwert	'

! → Die notorischen Problembereiter *Umlaute* müssen aufgrund der Tatsache, dass in einem XML-Dokument stets eine Kodierung vorhanden ist, nicht gesondert maskiert werden – warum das so ist, werden wir noch genauer in Kapitel 4.2 auf Seite 22 sehen.

Die oben gezeigte Ersatzdarstellung wird auch als *Entität* bezeichnet und ist nicht nur auf die bereits gezeigten Zeichen beschränkt, sprich: Sie können selbst nach Lust und Laune eigene Entitäten definieren.[5]

→Kapitel 4.2 auf Seite 22

Daneben *kann* jedes andere Zeichen als Entität mit Dezimal-, Hexadezimal- oder UCS-4-Codierung dargestellt werden. Aus diesem Grund sind die beiden folgenden Elemente identisch:

[5] Überraschenderweise übersetzt uns Langenscheidts Taschenwörterbuch Englisch den Begriff *Entity* mit „Wesen(heit), Dasein“, bzw. „juristische Person“ ;-)

```
<name>Angel's Face</name>

<name>&#65;&#110;&#103;&#101;&#108;&apos;&#115;&#32;
&#x46;&#x61;&#x63;&#x65;</name>
```

Längere oder komplexereSonderzeichenpassagen können mit Hilfe des CDATA-Abschnitts eleganter konstruiert werden, auch wenn dessen Benutzung anfänglich etwas kryptisch erscheint.

```
...
<mixanleitung>In den Shaker
<![CDATA[(> 200ml, gibt's bei Schluckspecht & Sohn)]]>
einige Eiswürfel geben und den Gin, Calvados und Brandy
dazu.</mixanleitung>
...
```

Die Funktionsweise von CDATA ist dabei eigentlich recht simpel: Alle Zeichen zwischen <![CDATA[und]]> werden ohne Interpretationsversuch überlesen – Sie sollten sich ganz nebenbei gesprochen bereits die Frage beantworten können, warum wohl die Verwendung der Zeichenfolge]]> innerhalb eines CDATA-Abschnitts verboten ist.

Vielleicht sind Sie im letzten Absatz über den Begriff „Interpretationsversuch“ gestolpert und fragen sich nun, *wer* denn eigentlich Ihr XML-Dokument irgendwann mal interpretiert.

Belassen wir es hier noch mit der Antwort: „Der Parser“. Die genaue Funktion eines XML-Parsers erfolgt dann in Kapitel 4.

3.6 Namen: nur Schall und Rauch?

→Kapitel 3.3 auf Seite 12

Aus dem Kapitel *Marken in XML und HTML* wissen Sie bereits, dass Sie die Namen für Marken relativ frei vergeben können, dies gilt unter anderem auch für die Benennung von Attributen und Entitäten.

Die Verwendung des Wörtchens „relativ“ weist an dieser Stelle schon darauf hin, dass dieser Freiheit irgendwo auch Grenzen gesetzt sind. Für Marken, Attribute und Entitäten sind diese Grenzen an den folgenden Stellen erreicht:

! →

- Das *erste Zeichen* eines Namens muss ein *Buchstabe* aus dem Unicode-Zeichensatz (UCS), ein Unterstrich (_) oder ein Doppelpunkt (:) sein.

  Ab dem *zweiten Zeichen* können zusätzlich Ziffern, der Bindestrich (-) oder ein Punkt (.) verwendet werden. Kaum zu glauben, aber wahr: das Konstrukt

  ```
  <_-.>Cocktailkirsche</_-.>
  ```

ist in XML ein korrekt benanntes Element.

→Kapitel 6.1 auf Seite 53

- Vorsicht bei der Verwendung des Doppelpunktes! Dieser wird auch in Zusammenhang mit *Namensräumen* benutzt, was gelegentlich zu Konflikten führt.
- Die Zeichenfolge „XML“ ist in jeder Form der Groß-/Kleinschreibung für das W3Consortium reserviert und darf daher nicht zu Beginn eines Namens verwendet werden. Nicht erlaubt sind daher Namen wie `XMLBar`, `xMLCocktail`, `XmlShaker`, ...

3.7 Aufgaben zur Selbstkontrolle

Bewerten Sie, möglichst ohne Zuhilfenahme der Unterlagen, die Richtigkeit der folgenden XML-Konstrukte:

1. `<XmLCocktail>Swimmingpool</XmLCocktail>`

   ☐ Korrekt ☐ Falsch

2. `<!-- erstellt am <#1-1-01#> -->`

   ☐ Korrekt ☐ Falsch

3. `<zutat menge=3cl>Cream of Coconut</zutat>`

   ☐ Korrekt ☐ Falsch

4. `<_-.>Sahne</_-.>`

   ☐ Korrekt ☐ Falsch

5. `<mixanleitung>Alle Zutaten im Shaker mit <zutat menge='3'>Eiswürfeln vermixen</mixanleitung>`

   ☐ Korrekt ☐ Falsch

6. `<dekoration>Ein Cocktai&#108;schirmchen</dekoration>`

   ☐ Korrekt ☐ Falsch

7. `<EndedesRezepts>`

   ☐ Korrekt ☐ Falsch

Eingaben löschen | Aufgaben lösen

4 Gerührt und geschüttelt

B52 (Shooter)

3cl: Tia Maria
3cl: Bailey's Irish Cream
1/2cl Captain Morgan Jamaika Rum

Mixanleitung
Zuerst den Tia Maria, dann den Bailey's in ein kleines, maximal 8cl fassendes (hitzebeständiges!) Glas geben. Dabei darauf achten, dass die Liköre nicht allzusehr ineinander fließen.
Obenauf eine Schicht von ca. 1/2cl Captain Morgan Rum geben, dessen 70% Alkohol gleich mit Hilfe eines Feuerzeugs entzündet werden.
Einen Strohhalm hineinstecken, kurz durchatmen und ganz schnell austrinken. Vorsicht! Der Strohhalm mag keine Hitze!

4.1 Aus Zutaten wird ein Drink, oder: Der Parser

Bislang haben wir uns ausschließlich mit dem Problem beschäftigt, Daten XML-konform anzulegen. Dabei ist ein wenig die Frage vernachlässigt worden, wer die strukturierten Daten letztendlich aufbereiten, weiterverarbeiten, darstellen soll.

Sofern Sie sich immer noch nicht – und trotz dieser Broschüre – von dem Gedanken gelöst haben, dass dies die Aufgabe eines WWW-Browsers ist, liegen Sie leider total falsch. Die „WWW-Sprache" ist nach wie vor HTML; denkbar wäre höchstens ein Konverter, der Ihre XML-Daten nach HTML überführt, welches dann wiederum...

Denkbar ist natürlich auch ein Programm, welches Ihre XML-Daten in ein Word-Format, eine Oracle-Datenbank oder gleich in PDF umsetzt. Damit befinden wir uns in Bereichen, die gänzlich außerhalb des WWW liegen – und ein entscheidender Vorteil von XML ist letztendlich die mediale Unabhängigkeit.

Natürlich gibt es bislang keine Applikation, welche alle oben aufgeführten Wünsche direkt umsetzt. Es existieren jedoch Programme, welche die Fähigkeit besitzen, Ihr XML-Dokument zu lesen, die darin enthaltenen Markups zu erkennen und nicht zuletzt die wiederum darin befindlichen Daten zu extrahieren. Diese Programme heißen *Parser*.

! → Um es noch einmal deutlich auszusprechen: Ein Parser ist *nicht in der Lage*, und es gehört auch *nicht zu seinen Aufgaben*, Ihre XML-Dokumente in irgendein anderes Format umzuwandeln oder für den Bildschirm

aufzubereiten! Dies ist im Normalfall die Aufgabe eines weiteren und eventuell selbst zu erstellenden Programms, welches in aller Regel einen Parser als Komponente benutzt.

4.1.1 Was tut dann ein Parser überhaupt?

Da ein Parser offenbar nichts direkt Verwertbares erzeugt, drängt sich diese Frage förmlich auf. Trotzdem nimmt Ihnen der Parser jedoch in Form von Überprüfungen bereits eine Menge Arbeit ab, die sich im Wesentlichen durch die zwei folgenden Begriffe beschreiben lässt:

Wohlgeformtheit : (Original: Wellformated) Um das Prädikat *wohlgeformt* zu erhalten muss das XML-Dokument alle Regeln erfüllen, die wir bislang bei der Verwendung von Marken, Attributen, Entitäten, etc. kennengelernt haben.

→Kapitel 3.3, 3.4 und 3.6

Gültigkeit : (Original: Validated) Ist das Dokument darüberhinaus mit internen oder externen Regeln verknüpft, so können einige Parser auch die festgelegten Vorschriften prüfen. Sofern dabei keine Fehler auftreten, heisst das Dokument *gültig*.

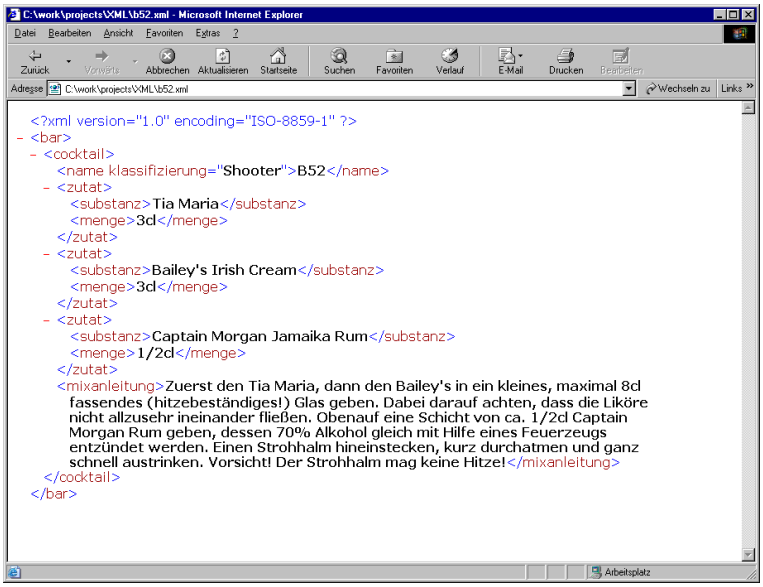


Abbildung 4.1: Microsofts Internet Explorer besitzt einen eingebauten Parser...

Es versteht sich von selbst, dass XML-Dokumente, die für eine externe Weiterverarbeitung vorgesehen sind, die beiden oben aufgeführten Kriterien nach Möglichkeit erfüllen. Umgekehrt gilt das natürlich auch für Dokumente, die Sie zur Weiterverarbeitung übernehmen.

4.1.2 Und wo bekomme ich einen Parser her?

→Abbildungen 4.1 und 4.2 Seit der Version 5 besitzt Microsofts Internet-Explorer einen eingebauten

Abbildung 4.2: . . . der auch auf Wohlgeformtheit prüft.

Parser (MSXML), der in der Lage ist, die Struktur eines XML-Dokuments im Browserfenster anzuzeigen und zusätzlich die *Wohlgeformtheit* zu kontrollieren.

→Abbildung 4.4 auf der nächsten Seite

Darüberhinaus kann von Microsofts Homepage der kostenlose zeilenorientierte Parser `xmlint` runtergeladen werden, der zusätzlich auf *Gültigkeit* prüfen kann.[1]

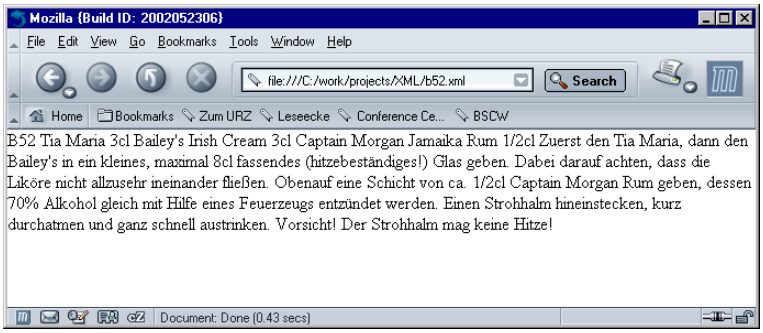


Abbildung 4.3: Andere Browser (hier: Mozilla) sind bei weitem nicht so komfortabel.

Viele Programmiersprachen besitzen mittlerweile eingebaute Routinen oder Klassen, über die ein interner Parser angesteuert wird. JAVA-Programmierer sollten sich das Package `javax.xml.parsers` näher anschauen, während für PHP-Interessierte die Klasse `phpxml` interessant sein kann.

→Kapitel 7 auf Seite 57

[1] Ganz nebenbei ist dieser Parser ist nur 32 KB groß.

Abbildung 4.4: Der Parser mit dem wahrscheinlich besten Preis-/Leistungsverhältnis: `xmlint`

4.2 Ä oder 0000000 00000000 00000000 11000100?

→Abbildungen 4.2 und 4.4

Im letzten Kapitel sind uns ganz nebenbei die ersten vom Parser bemängelten Fehler untergekommen. Für all diejenigen, die dieses Skript für lau aus der Leseecke gezogen haben und sich daher mit der 72dpi-PDF-Version rumschlagen müssen, sei die entsprechende Fehlermeldung noch einmal im Klartext dargestellt.

```
Im Textinhalt wurde ein ungültiges Zeichen gefunden
maximal 8cl fassendes (hitzebest
--------------------------------^
```

Genauer gesagt wurde der Fehler in Zeile 18 und Spalte 37 gefunden. Forschen wir in unserer Quelldatei an der angegebenen Position nach, so stoßen wir auf ein „ä“.

Der technische Hintergrund ist, man ahnt es bereits, ein komplexer und soll daher an dieser Stelle nur verkürzt und wahrscheinlich eher „populärwissenschaftlich“ wiedergegeben werden.

Generell erwartet der Parser seinen zu parsenden Text, also Ihr XML-Dokument, im Unicode Transformation Format, oder kurz: UTF. UTF ist wiederum eine verkürzte Darstellung des Universal Character Set (UCS). Über UCS heißt es wiederum auf der Unicode Home Page:

> „Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language.“

Es geht also um die einheitliche und übergreifende Darstellung der verwendeten Zeichen. Was prinzipiell erstmal hervorragend klingt, ist in der Praxis häufig mit Nachteilen behaftet. Im Falle von UCS äußert sich der Pferdefuß darin, dass zur Darstellung jedes Zeichens anstelle des klassischen Bytes gleich derer vier benötigt werden, was wiederum bedeutet, dass auch Ihre Dokumente um den Faktor 4 an Größe zunehmen.

Abhilfe schaffen hier die komprimierenden UTF-Formate, wobei uns jetzt weniger interessieren soll, wie da genau komprimiert wird,[2] sondern eher, wie wir unser XML-Dokument in ein UTF-Format bekommen.

→Abbildung 4.5

Der einfachste Weg ist häufig der, an den am wenigsten gedacht wird. Tatsächlich besitzen mittlerweile viele Editoren, unter anderem, seit Windows 2000, das „klassische“ Microsoft Notepad die Möglichkeit, direkt in UTF zu speichern.

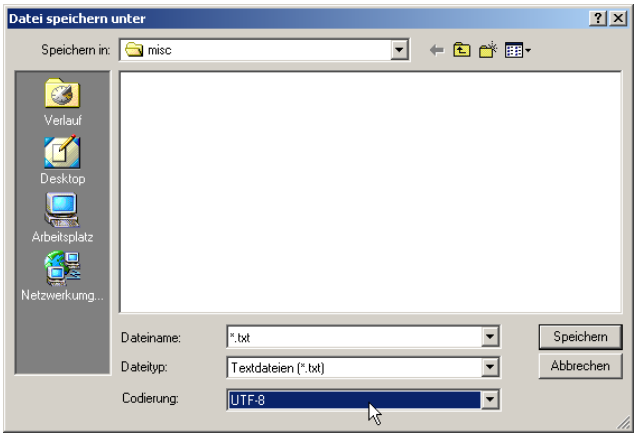


Abbildung 4.5: Grundsätzlich gilt: Sofern nicht anders angegeben, verlangen Parser ein UTF-Format.

Sofern Ihnen Ihr Lieblingseditor keinen UTF-Filter anbietet, können Sie die gewünschte Kodierung alternativ über das `encoding`-Attribut im Prolog Ihres Dokuments festlegen. Das folgende Beispiel demonstriert diese Technik.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<bar>
 <cocktail>
  <name>B52</name>
  ...
```

Die in diesem Beispiel verwendete Kodierung ISO-8859-1 enthält den klassischen ASCII-Zeichensatz, sowie alle westeuropäischen Sonderzeichen – also all die Zeichen, die in „normalen“ Editoren verwendet werden.

4.3 XML arbeitet

Langsam aber sicher verlassen wir den Bereich der Erfassung und wenden uns dem Problem zu, wie denn unsere Daten aufbereitet werden können.

Wir erinnern uns: XML selbst beinhaltet lediglich die Möglichkeit, erfasste Daten über ein Markup zu strukturieren. Auch der Parser schafft

[2] Wer es wirklich ganz genau wissen will kann unter `http://www.unicode.org` nachschauen.

hier keine Abhilfe, da seine Aufgabe im Wesentlichen darin besteht, die aufgebaute Struktur zu überprüfen.

Die letztendliche Aufbereitung des XML-Dokuments muss daher externen Anwendungen vorbehalten bleiben, die logischerweise mit dem Dokument verknüpft werden müssen. Dies geschieht im Regelfall über Verarbeitungsanweisungen.

Verarbeitungsanweisungen werden normalerweise im Prolog des XML-Dokuments eingefügt, gelegentlich tauchen sie aber auch schon mal hinter dem Wurzelelement auf. Die Struktur einer Verarbeitungsanweisung ist dabei ausgesprochen schlicht, sie hat die Form

```
<?Name Attributliste?>
```

oder, um ein konkreteres Beispiel mit den Verarbeitungsanweisungen `xml` und `xml-stylesheet` zu zeigen:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<?xml-stylesheet type='text/xsl' href='bar.xsl'?>
<bar>
 <cocktail>
  <name>B52</name>
  ...
```

Für uns ist momentan noch nicht von Interesse, *was* alles in Verarbeitungsanweisungen (im oberen Beispiel die bildschirmgerechte Aufbereitung über XSL) untergebracht werden kann, wichtig ist jetzt erstmal das *Wie*.

Rein syntaktisch kommt es dabei lediglich auf die folgenden Punkte an:

- Zwischen dem ersten ? und dem Namen darf sich kein Leerzeichen befinden, eine Verarbeitungsanweisung `<? Name>` ist also fehlerhaft.
- Und, wie nicht anders zu erwarten, darf die Zeichenkombination `?>` nicht in der Attributliste auftauchen.

Rein syntaktisch betrachtet ist daher

```
<?Prost alle='zusammen'?>
```

eine korrekte Verarbeitungsanweisung – wer immer sie auch letztendlich umsetzen mag.

4.3.1 Formatierung

Verarbeitungsanweisungen werden häufig dazu benutzt, um aus XML-Dokumenten eine formatierte Ausgabe zu zaubern – ganz nebenbei wird

diese Broschüre dadurch um ein Kapitel ergänzt, mit dem sich „wirklich“ etwas anfangen lässt.

Durch die Verwendung von Verarbeitungsanweisungen existieren netterweise gleich mehrere Möglichkeiten, aus XML-Anwendungen formatierte Ausgaben zu erzeugen.

XSL : (= e*X*tensible *S*tylesheet *L*anguage) XSL ist gewissermaßen eine eigene „Programmiersprache“, die nicht ganz einfach zu erlernen ist. XSL ist *kein* Bestandteil von XML und damit auch nicht Inhalt dieser Broschüre.

CSS : (= *C*ascading *S*tyle*S*heets) CSS waren ursprünglich als Formatvorlagen für HTML gedacht, sie können jedoch auch direkt mit XML-Dokumenten verknüpft werden. Cascading Stylesheets sind allgemein die simpelste Möglichkeit, XML-Anwendungen Format zu verleihen.

Halten Sie sich bitte nochmal deutlich vor Augen, dass die Formatierung von Daten, weder durch CSS noch durch XSL, nicht einmal ansatzweise Bestandteil unseres Kernthemas XML ist. Nichtsdestotrotz können auch wir ein gewisse Neugier nicht verhehlen. Lassen Sie uns daher die graue Theorie ein wenig beiseite legen und dafür einen Kurzausflug in die Welt der Darstellung von XML-Dokumenten unternehmen.

Exkurs: Aufbereitung durch CSS

Auf den folgenden 1 1/2 Seiten wird demonstriert, wie Sie ein XML-Dokument mit Cascading Style Sheets verknüpfen und in einem WWW-Browser darstellen können.

Dagobert Duck (Spardrink)

1Spritzer: Soda
Leitungswasser

Mixanleitung
Das Soda in ein Longdrinkglas geben, das Ganze mit Leitungswasser auffüllen.

Der oben aufgeführte Cocktail wird in XML-Notation in der Datei `dd.xml` gespeichert.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<?xml-stylesheet type='text/css' href='bar.css'?>
<bar>
 <cocktail>
  <name klassifizierung='Spardrink'>Dagobert Duck</name>
  <zutat>
   <substanz>Soda</substanz>
   <menge>1Spritzer</menge>
  </zutat>
  <zutat>
   <substanz>Leitungswasser</substanz>
   <menge />
  </zutat>
  <mixanleitung>
  Das Soda in ein Longdrinkglas geben, das Ganze mit
  Leitungswasser auffüllen.
  </mixanleitung>
 </cocktail>
</bar>
```

Das zugehörige Cascading Stylesheet (`bar.css`) kann wie folgt realisiert werden.

```
cocktail {  display: block;
            font-family: sans-serif;
            background-color: lightgreen;
            border: solid;
}
name {      display: block;
            font-size: 18pt;
            color: blue;
}
zutat {     display: list-item;
            font-size: 12pt;
}
menge {
            padding-left: 0.5cm;
}
mixanleitung {
            font-size: 11pt;
            font-style: oblique;
            background-color: yellow;
            display: block;
}
```

Auch ohne tiefgreifende CSS-Kenntnisse dürfte schnell klar werden, welche Formate wie angewandt werden.

Die Datei `dd.xml` kann nun wie eine HTML-Datei in einem WWW-Browser (Internet Explorer ab Version 5, Netscape ab Version 6) geöffnet werden. Das erzielte Ergebnis wird je nach den vorhandenen CSS-Fähigkeiten des verwendeten Browsers variieren, es dürfte aber Abbildung 4.6 sehr nahe kommen.

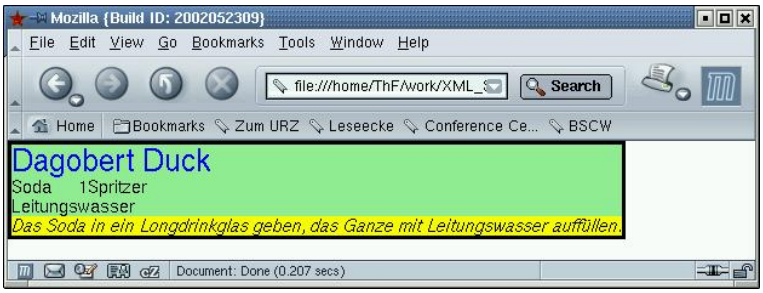


Abbildung 4.6: Die meisten moderneren Browser (hier: Mozilla) sind in der Lage, XML-Dokumente mit CSS-Anweisungen darzustellen.

Ungleich eleganter wirkt das Ganze, wenn die Aufbereitung, anstatt durch starre Cascading Style Sheets, via XSL vorgenommen wird. Allein durch den Austausch der entsprechenden Verarbeitungsanweisung

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!--?xml-stylesheet type='text/css' href='bar.css'?-->
<?xml-stylesheet type='text/xsl' href='bar.xsl'?>
<bar>
...
```

wird ein Aussehen wie in Abbildung 4.7 erzielt.

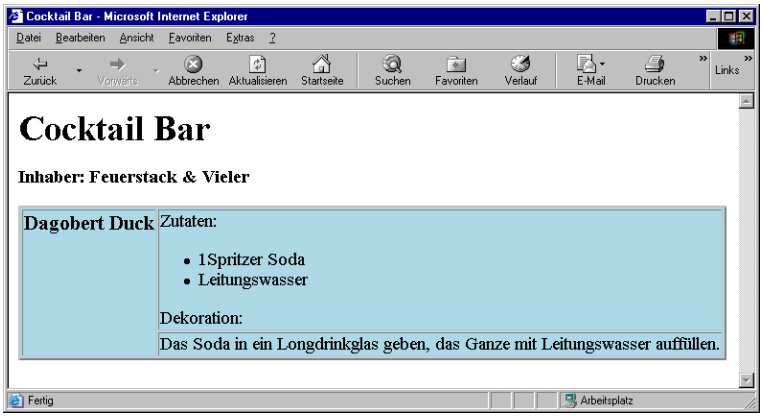


Abbildung 4.7: Der gleiche Cocktail, aber anders zubereitet.

Die zugehörigen XSL-Verarbeitungsanweisungen sind im Vergleich zu CSS natürlich komplexer, Sie finden Sie trotzdem im Anhang B.1. Darüberhinaus wollen wir nicht verheimlichen, dass XSL bislang nur durch den Internet Explorer umgesetzt werden kann!

! →

Einen Ausweg aus diesen „Browserabhängigkeiten“ bieten sogenannte Serverside-Anwendungen – wir werden am Ende dieser Broschüre darauf zurückkommen.

→Kapitel 7 auf Seite 57

Damit wollen wir unseren Ausflug beenden und zum ursprünglichen Thema zurückkehren.

4.3.2 Ganz speziell: Die `xml`-Verarbeitungsanweisung

Eine Sonderrolle spielt die im Normalfall zuerst auftretende Verarbeitungsanweisung mit dem Namen `xml`. Diese muss, falls vorhanden, die erste aller Verarbeitungsanweisungen im Dokument sein – weswegen sie sich immer direkt in der ersten Zeile befindet. Zu beachten: Auch Kommentare dürfen der `xml`-Verarbeitungsanweisung nicht vorangestellt werden.

Die `xml`-Verarbeitungsanweisung gilt auch als Pseudoanweisung, da ihr Inhalt, im Gegensatz zu allen anderen Verarbeitungsanweisungen, direkt vom Parser berücksichtigt wird.

Prinzipiell dürfen der `xml`-Verarbeitungsanweisung die folgenden drei Attribute übergeben werden:

`version` : definiert die Versionsnummer der verwendeten XML-Version. Voreinstellung: `1.0`

`encoding` : bekommt als Attributwert die verwendete Zeichenkodierung. Voreinstellung: `UTF-8`

`standalone` : beschreibt, ob das Regelwerk für Marken, Attribute und Entitäten im Dokument enthalten ist, oder über eine externe Document Type Definition (DTD) geladen wird. Voreinstellung: `no`

! → Die Attribute müssen in der oben angegebenen Reihenfolge aufgeführt werden! Eine `xml`-Verarbeitungsanweisung der Form

```
<?xml encoding='UTF-8' version='1.0'?>
```

würde vom Parser aufgrund der falschen Reihenfolge als fehlerhaft reklamiert.

4.4 Kommentare

Kommentare dienen der Übersichtlichkeit Ihres XML-Dokuments und helfen dadurch Ihnen, als auch denjenigen die Ihre XML-Daten weiterverarbeiten müssen.[3]

Ganz nebenbei können temporär nicht benötigte XML-Anweisungen durch Auskommentieren für den Parser „unsichtbar gemacht“ werden.

Kommentare beginnen mit der Zeichensequenz `<!--` und enden mit `-->`.

```
<?xml version='1.0'?>
<?xml-stylesheet type='text/css' href='bar.css'?>
<!-- Bildschirmausgabe über Cascading Stylesheets -->
<!-- ?xml-stylesheet type='text/xsl' href='bar.xsl'? -->
<!-- XSL ist auskommentiert, da nicht Bestandteil
dieser Broschüre -->
<bar>
 <cocktail>
 ...
```

! → Der Kommentartext darf die Zeichenfolge „`--`“ nicht enthalten.

[3] Sie werden aus diesem Grund eher selten an der FernUni benutzt. :)

4.5 Aufgaben zur Selbstkontrolle

Versuchen Sie, möglichst ohne Zuhilfenahme der Unterlagen, die folgenden Fragen zu beantworten, bzw. die Sätze zu vervollständigen:

1. Welche Aussage tifft auf das folgende XML-Dokument zu?

```
<?xml version='1.0'>
<bar>Blauer Kakadu</bar>
```

   (a) ☐ Das Dokument ist wohlgeformt.

   (b) ☐ Das Dokument ist gültig.

   (c) ☐ Das Dokument ist wohlgeformt und gültig.

   (d) ☐ Das Dokument ist nichts von all dem.

2. Wie müssen innerhalb eines XML-Dokuments Umlaute kodiert werden?

   (a) ☐ Umlaute können gar nicht verwendet werden, da nur Zeichen erlaubt sind, die sich auf den ersten 127 Stellen der ASCII-Tabelle befinden (Beispiel: ä→ae).

   (b) ☐ Umlaute werden generell als Hexadezimal-Entität abgelegt (Beispiel: ä→`&#00E4;`).

   (c) ☐ Umlaute müssen nicht besonders behandelt werden, sofern beim Speichern des Dokuments das darin verwendete Encoding (Beispiel: `ISO-8859-1`) beachtet wird.

   (d) ☐ Als Bürger der USA interessieren mich Umlaute überhaupt nicht.

3. Welches der folgenden Attribute gehört *nicht* in die `xml`-Verarbeitungsanweisung?

   (a) ☐ `Version`

   (b) ☐ `standalone`

4. Was passiert, wenn ein XML-Dokument in einem WWW-Browser geöffnet wird?

   (a) ☐ Das Dokument wird vollständig formatiert angezeigt.

   (b) ☐ Das Dokument wird unformatiert angezeigt, der Browser berücksichtigt jedoch die vorhandenen Strukturen und rückt entsprechend ein.

   (c) ☐ Das Dokument erscheint als ASCII-Fließtext.

   (d) ☐ Die Frage kann nicht generell beantwortet werden, das Ergebnis hängt vom verwendeten Browser ab.

Eingaben löschen | Aufgaben lösen

5 Was alles in den Shaker darf

Alexander (Digestif)

1 Teil: Cognac
1 Teil: Créme de Cacao braun
1 Teil: Sahne

Dekoration
Zimt

Mixanleitung
Alle Zutaten – mit Ausnahme des Zimts – werden im Shaker inklusive einigen Eiswürfeln geschüttelt und in ein Cocktailglas abgeseiht. Je nach Geschmack den fertigen Drink mit etwas Zimt bestreuen.

Im nun folgenden Teil der Broschüre beleuchten wir nach der Wohlgeformtheit nun die Gültigkeit der XML-Daten. Wir werden ein Regelwerk definieren, welches die einzelnen Elemente unserer Cocktailbar beschreibt: Was gehört zu einem Cocktail? Sind die Zutaten in ausreichender Anzahl und vollständig aufgeführt? Sind Bilder erwünscht? Wird die Rezeptstruktur (Beschreibung, Zutaten, Mixanleitung) eingehalten?

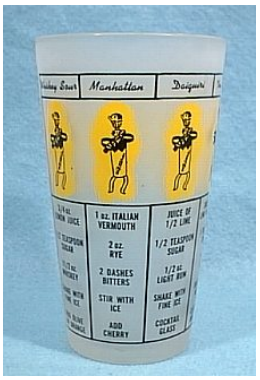


Abbildung 5.1: Messbecher mit *gültigen* Cocktailrezepten

Entsprechen unsere Cocktailrezepte solchen Regeln, so sind sie *gültig* (*valid*) und können den verschiedensten Anwendungen zur Verarbeitung gereicht werden, beispielsweise dem Mixer an der Bar, dem Verlag zur Erstellung eines Rezeptbuches, oder – mal was ganz Besonderes – einem Mixautomaten.[1]

[1] Der flächendeckende Einsatz solcher Automaten, die es übrigens wirklich gibt, ist bislang nur deshalb ausgeblieben, weil sich Maschinen morgens um halb 4 nicht auf Diskussionen mit betrunkenen und verzweifelten Ehemännern einlassen wollen.

5.1 DTD

Das XML-Regelwerk wird in einer DTD – *Document Type Definition* festgelegt. Diese Gültigkeitregeln beschreiben

- Elemente, Attribute, Entitäten – *Welche Marken gibt es?*
- Inhalte und Häufigkeitsangaben – *Was darf drin sein und wie oft?*
- Beziehungen einzelner Elemente zueinander. Reihenfolge, Abhängigkeiten, Verschachtelungen, Struktur – *Erst wird gemixt, dann dekoriert.*

Die DTD-Anweisungen können wahlweise innerhalb des XML-Dokuments, oder außerhalb, in einer expliziten DTD-Datei, deklariert werden. In den folgenden, stark komprimierten Beispiel-Quellcodes wird zur besseren Lesbarkeit auf die interne Deklaration – die *Quick and Dirty* Variante – zurückgegriffen. Im Produktionseinsatz ist die externe Deklaration wegen ihrer klaren Trennung von Daten und Regelwerk vorzuziehen.

5.1.1 Interne Deklaration

Gehen wir zunächst von einer sehr einfachen Bar aus, die einen Cocktail mit einem Namen und einer Zutat enthält.

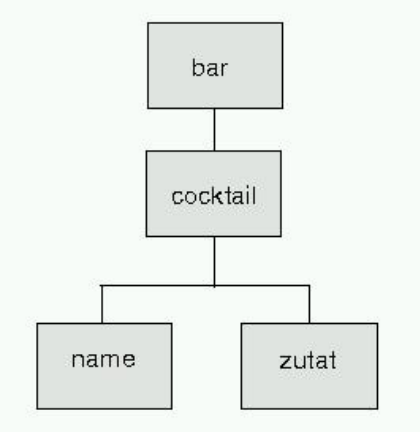


Abbildung 5.2: Grundstruktur unserer Bar

Die entsprechende `bar.xml` wird dieser Struktur wie in Beispiel 5.1 auf der nächsten Seite gerecht.

Die aufmerksamen Leser haben schon die beiden Erweiterungen entdeckt, die über die reine Wohlgeformtheit hinausgehen.

Zum einen wird der XML-Tag um `standalone='yes'` ergänzt, womit signalisiert wird, dass alle notwendigen Informationen – eben auch die Gültigkeitsregeln – in dieser XML-Datei vollständig vorhanden sind.

Daneben werden die DTD-Anweisungen direkt im `DOCTYPE`-Tag eingebettet – daher die Bezeichnung *Interne Deklaration*. Die allgemeine Syntax lautet:

```
<?xml version='1.0' standalone='yes'?>
<!DOCTYPE bar [
          <!ELEMENT bar (cocktail)>
          <!ELEMENT cocktail (name,zutat)>
          <!ELEMENT name (#PCDATA)>
          <!ELEMENT zutat (#PCDATA)>
]>
<bar>
 <cocktail>
  <name>
   Alexander
  </name>
  <zutat>
   3cl Creme de Cacao braun
  </zutat>
 </cocktail>
</bar>
```

Beispiel 5.1: Ohne Regeln bleibt auf Dauer keine Bar unbeschädigt.

```
<!DOCTYPE name [ <DTD-Anweisung> ... ]>
```

Was den Aufbau des DTD-Statements `ELEMENT` (siehe Beispiel) angeht, so werden Sie schon jetzt erkennen, dass die `bar` einen `cocktail` im Angebot hat, der wiederum einen `namen` hat und aus einer `zutat` besteht. Im Kapitel 5.2 auf Seite 36 wird dessen Bedeutung und Aufbau vertieft.

5.1.2 Externe Deklaration

Die gleiche Bar kann aber auch allgemeingültiger realisiert werden.[2] Durch die Trennung von Datenbestand (XML) und Regelwerk (DTD) in jeweils eigenen Dateien gelingt es, mehrere unterschiedliche Datenbestände ein und demselben Regelwerk zu unterwerfen.

Eine Verknüpfung von XML und DTD wird wieder im `DOCTYPE`-Tag des XML-Dokuments durchgeführt, diesmal jedoch lediglich mit dem Verweis auf eine DTD-Datei – die sogenannte *Externe Deklaration*. Der `xml`-Tag-Parameter `standalone` entfällt deshalb.

Für die im Beispiel aufgebaute Mini-Bar benötigen wir daher eine `bar.xml`:

[2] Stellen Sie sich vor, Sie wollten unterschiedliche Datenbestände wie Cocktails, Sektcocktails, Longdrinks und Shortdrinks verwalten, die sicherlich alle Namen, Zutaten und Mixanleitungen haben - sich von der Datenstruktur her also ähneln.

```
<?xml version='1.0'?>
<!DOCTYPE bar SYSTEM 'bar.dtd'>
<bar>
 <cocktail>
  <name>
   Alexander
  </name>
  <zutat>
   3cl Creme de Cacao braun
  </zutat>
 </cocktail>
</bar>
```

sowie die DTD-Anweisungen in `bar.dtd`:

```
<!ELEMENT bar (cocktail)>
<!ELEMENT cocktail (name,zutat)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT zutat (#PCDATA)>
```

Andere XML-Datenbestände können nun mit dem gleichen `DOCTYPE`-Verweis auf diese `bar.dtd` Bezug nehmen.

5.1.3 Öffentliche Deklaration

Wird die zu verknüpfende Regeldatei von einem größeren Anwenderkreis benutzt (dessen Umfang so ungefähr ab weltweit aufwärts vermutet wird), bzw. planen Sie eine derartige Regeldatei zu erstellen, so empfiehlt sich die Einbindung als *öffentliche Deklaration*.

Im Vergleich zur lokalen Verknüpfung wird diesmal das Schlüsselwort `SYSTEM` durch `PUBLIC` ersetzt. Hinter `PUBLIC` folgt wieder ein URI, der sich nun jedoch aus einer Eigentümer-, Dokumentenklassen- und Sprachcodebeschreibung zusammensetzt (jeweils durch „//“ voneinander getrennt), bevor als Verknüpfungsadresse der bereits bekannte URL folgt.

Eine der bekanntesten öffentlichen Deklarationen bezieht sich auf die DTD mit der Regelbeschreibung von XHTML.

```
<!DOCTYPE bar PUBLIC
'-//W3C//DTD XHTML 1.0 Transitional//EN'
'http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd'>
```

Der Zeilenumbruch des letzten Beispiels hat freundlicherweise automatisch dafür gesorgt, dass wir die öffentliche Deklaration schrittweise analysieren können.

Zeile 1 : birgt soweit nichts Neues. Das Schlüsselwort `PUBLIC` zeigt lediglich an, dass es sich um eine öffentliche Verknüpfung handelt.

Zeile 2 : `-//W3C` beschreibt den Eigentümer oder Ersteller der DTD, als das W3Consortium. `//DTD XHTML...` zeigt an, dass es sich um ein Regelwerk für XHTML handelt in der Version `1.0`. Das Ganze ist dazu in Englisch (`//EN`) verfasst worden.

Um nochmal kurz auf die Eigentümerschaft zurückzukommen: Das führende „`-`“ (in `-//W3C`) weist darauf hin, dass das W3Consortium ein *nicht registrierter* Eigentümer der DTD ist. In Analogie dazu existieren, erkennbar am führenden „`+`“, zumindest in der Theorie auch registrierte Besitzer – bis zur Stunde der Drucklegung konnte dem Autorenteam jedoch niemand verraten, wo sich Interessenten registrieren lassen können.

Zeile 3 : gibt abschließend nur noch die Adresse an, von der die gewünschte DTD geladen werden soll.

Tipp: Sofern Sie daran interessiert sind mal einen Blick hinter die Kulissen zu werfen, sollten Sie – *aber bitte erst nachdem Sie den Rest dieser Broschüre gelesen haben* – die oben aufgeführte DTD `xhtml1-transitional` auf dem eigenen Rechner speichern. Auch wenn sich die DTD auf XHTML bezieht, so ist sie aufgrund der hohen Kompatibilität ein gutes Beispiel um zu zeigen, wie beispielsweise HTML definiert ist.

Na, konnten Sie Ihre Neugier nicht zügeln und haben bereits in der `xhtml1-transitional.dtd` rumgeschnuppert? Lassen Sie mich abschließend feststellen, dass auch die Syntax für DTDs erstmal chaotischer ausschaut, als sie letztendlich wirklich ist.

5.2 Elemente

Julia (Cocktail)

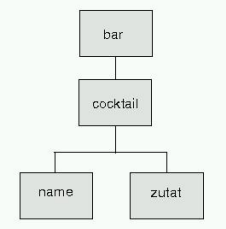
2 cl: Amaretto
2 cl: Rum weiß
6 cl: Sahne

Dekoration
Erdbeeren

Mixanleitung
Zutaten im Elektromixer mit einigen Eiswürfeln gut durchmixen und das fertige Mix in einen großen Sektkelch füllen. Erdbeere an den Glasrand stecken.

Wie sehen nun solche DTD-Regeln aus?

Beginnen wir mit der Überlegung, welche gültigen Elemente ein XML-Datensatz haben darf, in welcher Anzahl Elemente auftauchen dürfen, welche Beziehungen oder Abhängigkeiten mehrerer Elemente zueinander haben, ob eine Reihenfolge eingehalten werden muss und schließlich, was für eine Struktur das ganze Datengebilde aufweisen soll.



Im vorhergehenden Kapitel wurde eine Beispiel-Bar konstruiert, die einen Cocktail mit einem Namen und einer Zutat enthält (siehe Abbildung 5.2). Name und Zutat sollen lediglich Text enthalten. All das wird in einer entsprechenden *Document Type Definition* (DTD) mit dem `ELEMENT`-Tag festgelegt:

```
<!ELEMENT Elementname Elementinhalt>
```

Mit dem *Elementnamen* werden gültige XML-Elemente definiert. Die chronologische Reihenfolge der Elemente im XML-Dokument muss der Liste der `ELEMENT`-Tags entsprechen.

Als *Elementinhalt* dürfen die folgenden Werte verwendet werden:

`EMPTY`	leeres Element
`ANY`	beliebig
Elementmodell	Bei Kindelementen: Name, Abfolge und Häufigkeit Im Beispiel: `(name,zutat)`
`(#PCDATA)`	Parsed Character Data - Inhalt besteht aus Text

Zu unserer Beispiel-Bar benötigt man demnach die folgenden `ELEMENT`-Anweisungen:

```
<!ELEMENT bar (cocktail)>
<!ELEMENT cocktail (name,zutat)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT zutat (#PCDATA)>
```

5.2.1 Traumberuf Barmixer? (Teil I: Das Ziel)

Nun ist eine Bar mit nur einem Cocktail (der dazu auch noch aus nur einer einzigen Zutat besteht) zugegebenermaßen nicht marktfähig. Eine reichhaltige Getränkekarte mit anständigen Drinks sei daher genug Motivation dieses Elementmodell zu dem auszubauen, was in Abbildung 5.3 zu sehen ist: Mehr Cocktails, mehr Zutaten, Hinweise zur Dekoration, eine Mixanleitung und eventuell ein Foto.

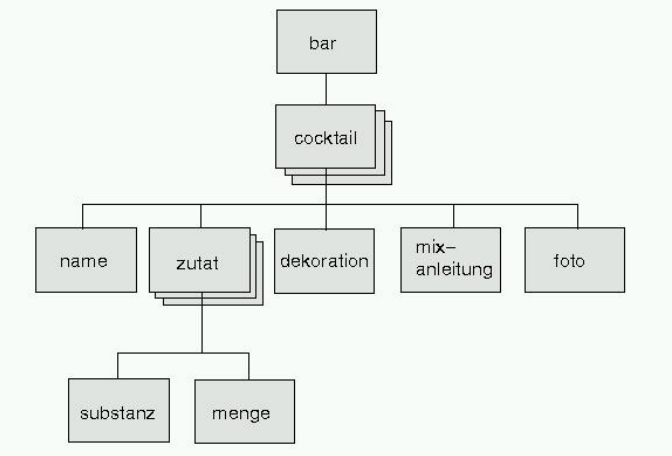


Abbildung 5.3: Elementmodell – Ziel ist eine vollständige Bar.

Teile dieses Barmodells werden im weiteren Verlauf dieser Broschüre immer wieder als Beispiel herangezogen werden.

5.2.2 Häufigkeitsangaben

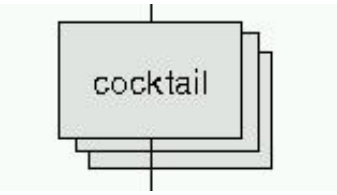


Abbildung 5.4: Mehrfachnennung von Elementen

Häufigkeitsangaben legen fest, wie oft ein Element, eine Elementfolge oder eine Elementauswahl auftreten soll bzw. darf. Dabei existieren vier mögliche DTD-Notationen:

`Element`	genau 1 mal
`Element?`	0 oder 1 mal (optional, höchstens einmal)
`Element*`	0 bis n mal (optional, beliebig oft wiederholbar)
`Element+`	1 bis n mal (erforderlich, mindestens einmal, beliebig oft wiederholbar)

Als angehende Barbesitzer wollen wir natürlich viele verschiedene Cocktails anbieten. Jeder Cocktail soll einen Namen haben und aus mindestens einer Zutat bestehen.[3] Dekoration und Mixanleitung sind unverzichtbar, ein Foto hingegen kann erübrigt werden.

Damit ergibt sich das folgende DTD-Konstrukt:

```
...
<!ELEMENT bar (cocktail*)>
<!ELEMENT cocktail
 (name,zutat+,dekoration,mixanleitung,foto?)>
...
```

! → Achten Sie darauf, dass Sie keine Leerräume (Tabulatoren, Leerzeichen, etc.) zwischen den Elementnamen und den Häufigkeitsangaben einsetzen.

5.2.3 Elementfolge

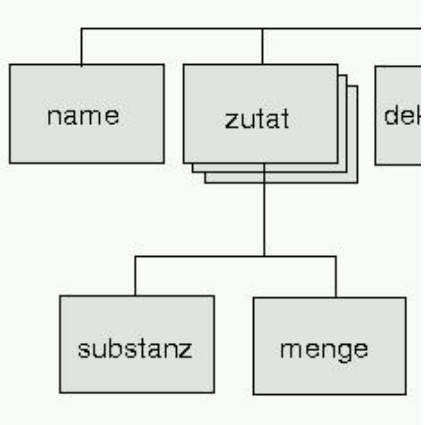


Abbildung 5.5: Die verbindliche Reihenfolge von Elementen

Die Elementfolge legt die verbindliche Reihenfolge von Elementen, Elementfolgen oder Elementauswahlen fest:

`(Element1,Element2,...)` erster, zweiter,...

[3] Den Autoren sind übrigens keine Cocktails bekannt, die nur aus einer Zutat bestehen. Cocktails, die nur aus Rum bestehen, bleiben *Rum*. Selbst der Dagobert-Duck-Spar-Cocktail (Rezept auf Seite 26) besteht aus Leitungswasser **und** einem Spritzer Soda.

Kleine Übung→ Lernen Sie die Anleitungen für die Cocktails *Julia* und *Alexander* auswendig. Cocktailname, Mengen und Substanzen der Zutaten. Der immer gleiche Schema-Aufbau aller Cocktails wird Ihnen die Paukerei erleichtern. Vorhandene Abhängigkeiten – Mengenangaben werden zum Beispiel nur im Zusammenhang mit den Zutaten, und nicht im Namen genannt – drängen sich dabei wie von selbst auf.

Reihenfolgen und Abhängigkeiten führen zu diesem DTD-Konstrukt:

```
...
 <!ELEMENT name (#PCDATA)>
 <!ELEMENT zutat (substanz,menge)>
 <!ELEMENT substanz (#PCDATA)>
 <!ELEMENT menge (#PCDATA)>
...
```

Demnach besteht jede Zutat aus genau einer Substanz und einer Mengenangabe. Eine Umkehrung – zuerst die Menge und dann die Substanz – ist nicht zulässig.

5.2.4 Elementauswahl

Die Elementauswahl beschreibt eine Entweder/Oder-Verknüpfung von Elementen, Elementfolgen oder Elementauswahlen:

`(Element1|Element2|...)` Element1 oder Element2 oder ...

Abweichend vom bisherigen Bar-Modell, beschäftigen wir uns in Form eines kleinen Einschubs mit Personalangelegenheiten. Unsere Bar steht ja noch ganz am Anfang. Daher können wir nur einen Angestellten beschäftigen. Die DTD hierfür sieht wie folgt aus:

```
...
 <!ELEMENT Angestellte (BarFachfrau|MixMeister) >
 <!ELEMENT BarFachfrau (#PCDATA) >
 <!ELEMENT MixMeister (#PCDATA) >
...
```

! → Sollte unsere Bar so richtig in der Szene einschlagen, so werden selbstverständlich weitere Mitarbeiter eingestellt. Die Kombination von Häufigkeitsangaben und Elementauswahl hat jedoch so ihre Tücken. Beachten Sie den folgenden Effekt:

```
...
 <!ELEMENT Angestellte (BarFachfrau|MixMeister)*>
 <!ELEMENT BarFachfrau (#PCDATA) >
 <!ELEMENT MixMeister (#PCDATA) >
...
```

beschreibt eine gemischte Angestelltengruppe, eine beliebige Reihenfolge von sowohl BarFachfrauen als auch MixMeistern, wohingegen

```
...
 <!ELEMENT Angestellte (BarFachfrau*|MixMeister*)>
 <!ELEMENT BarFachfrau (#PCDATA) >
 <!ELEMENT MixMeister (#PCDATA) >
...
```

entweder **nur** BarFachfrauen **oder nur** MixMeister – jeweils in beliebiger Anzahl – verpflichtet.[4]

5.2.5 Traumberuf Barmixer? (Teil II: Das Ergebnis)

Pangalaktischer Donnergurgler (Cocktail)

1 Flasche:	Alten Janx-Geist
1 Teil:	Wasser aus den Meeren von Santraginus V
3 Würfel:	arkturanischer Mega-Gin
4 l:	fallianisches Sumpfgas
1 Teil:	qualaktinischer Hyperminz-Extrakt
1 Zahn:	eines algolianischen Sonnentigers
1 Spritzer:	Zamphour

Dekoration
Olive

Mixanleitung
Zutaten mischen, Mega-Gin-Würfel darin zergehen lassen, Das Sumpfgas hindurchperlen lassen, über einen Silberlöffel den Minze-Extrakt tröpfeln und den Zahn auflösen.

[Kein Bild verfügbar - Der Film ist während der Zubereitung im Kameragehäuse verdampft!]

Unsere Bar ist nun vollständig. Die DTD-Anweisungen sind die Summe der bisherigen Überlegungen. Auf das Foto (eine Grafik – daher `ANY` und nicht `(#PCDATA)`) kann gegebenenfalls verzichtet werden, weil unsere Angestellten sowieso jeden Drink schon x-mal zubereitet haben

[4] Als Bar-Besitzer werden Sie natürlich die 2. Variante wählen, weil sie so eine der beiden, nach Geschlechtern getrennten, Angestelltentoiletten einsparen.

und wissen wie er aussieht. Die `bar.dtd`:

```
<!ELEMENT bar (cocktail*)>
<!ELEMENT cocktail
  (name,zutat+,dekoration,mixanleitung,foto?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT zutat (substanz,menge)>
<!ELEMENT substanz (#PCDATA)>
<!ELEMENT menge (#PCDATA)>
<!ELEMENT dekoration (#PCDATA)>
<!ELEMENT mixanleitung (#PCDATA)>
<!ELEMENT foto ANY>
```

Die Cocktails werden nun (→Beispiel 5.2 auf der nächsten Seite) in `bar.xml` abgelegt.

5.3 Attribute

Tequila Sunrise (Longdrink)

4 cl:	Tequila Silver
2 cl:	Grenadine
etwas:	Orangensaft

Dekoration
Schwarzer Trinkhalm

Mixanleitung
Einige große Eiswürfel in ein Longdrinkglas geben, den Tequila dazu und mit Orangensaft auffüllen. Zum Schluss Grenadine dazugeben und mit einem schwarzen Trinkhalm gerade so wenig umrühren, dass man den Sonnenaufgang im Glas noch sieht.

Mit Hilfe von *Attributen* können den Elementen zusätzliche Informationen beigefügt werden. In unserer Bar kann beispielsweise jedem Cocktail die Klassifizierung mitgegeben werden, ob es sich eher um einen Cocktail, einen Longdrink oder einen Collins handelt.

Attribute werden im Gegensatz zu Elementen, deren Inhalt als eigentliche Informationseinheit verstanden wird, in erster Linie als Steuerinformation benutzt. Ein Verlag könnte das Attribut Klassifizierung dazu nutzen, die verschiedenen Cocktails zu gruppieren und sie in Kapiteln zusammenzufassen.

In der DTD findet eine Auflistung möglicher Attribute im Anschluss an die Element-Definition statt:

```
<?xml version='1.0'?>
<!DOCTYPE bar SYSTEM 'bar.dtd'>
<bar>
 <cocktail>
  <name>Alexander</name>
  <zutat>
   <substanz>Cognac</substanz>
   <menge>3cl</menge>
  </zutat>
  <zutat>
   <substanz>Creme de Cacao braun</substanz>
   <menge>3cl</menge>
  </zutat>
  <zutat>
   <substanz>Sahne</substanz>
   <menge>3cl</menge>
  </zutat>
  <dekoration>Zimt</dekoration>
  <mixanleitung>
   Alle Zutaten - mit Ausnahme des Zimts - werden im Shaker
 inklusive einigen Eiswürfeln geschüttelt und in ein
 Cocktailglas abgeseiht. Je nach Geschmack den fertigen
 Drink mit etwas Zimt bestreuen.
  </mixanleitung>
  <foto>alexander.jpg</foto>
 </cocktail>
 <cocktail>
  <name>Julia</name>
  <zutat>
   <substanz>Amaretto</substanz>
   <menge>2cl</menge>
  </zutat>
  <zutat>
   <substanz>Rum weiß</substanz>
   <menge>2cl</menge>
  </zutat>
  <zutat>
   <substanz>Sahne</substanz>
   <menge>6cl</menge>
  </zutat>
  <dekoration>Erdbeeren</dekoration>
  <mixanleitung>
   Zutaten im Elektromixer mit einigen Eiswürfeln
 gut durchmixen und das fertige Mix in einen großen
 Sektkelch füllen. Erdbeere an den Glasrand stecken.
    </mixanleitung>
  <foto>julia.jpg</foto>
 </cocktail>
</bar>
```

Beispiel 5.2: Die komplett ausgestattete Bar.

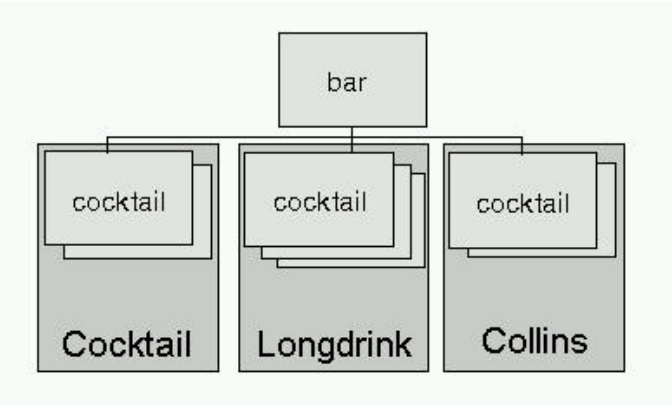


Abbildung 5.6: Attribute – hier als Sortierkriterium.

```
<!ELEMENT Elementname Elementinhalt>
<!ATTLIST Elementname Attributname1 Typ Default
                      Attributname2 Typ Default
                      ...>
```

Bevor die einzelnen Bestandteile des `ATTLIST` vorgestellt werden, zunächst unsere Beispiel-Bar, in der wir nach Abbildung 5.6 das Attribut `klassifizierung` als Zusatz zum Namen definieren:

```
<!ELEMENT bar (cocktail*)>
<!ELEMENT cocktail
          (name,zutat+,dekoration,mixanleitung,foto?)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST name klassifizierung (Cocktail|Longdrink|Collins)
          'Cocktail'>
```

Der *Elementname* beschreibt, auf welche Elemente sich die folgenden Attribute beziehen. Es dürfen nur vorangegangene, gültige Elemente verwendet werden – in diesem Fall `name`.

Der *Attributname* ist frei wählbar - im Beispiel `klassifizierung`.

Als *Attributtyp* können die folgenden Notationen verwendet werden:

Typ	Erläuterung, Beispiel DTD und Beispiel XML-Satz

`CDATA`	Zeichenkette aus beliebigen Zeichen (Character Data).
	
	`<!ATTLIST name klassifizierung CDATA 'Longdrink'>`
	
	`<name klassifizierung='Longdrink'>`
	`  Tequila Sunrise`
	`</name>`

`ID`	Es ist pro Element nur ein Attribut dieses Typs erlaubt. Namensregeln wie für XML-Namen. Der Wert muss eindeutig, das *erste Zeichen* darf *nicht nummerisch* sein! Voreinstellungen `#IMPLIED` oder `#REQUIRED`.
	
	`<!ATTLIST cocktail nummer ID #REQUIRED>`
	
	`<cocktail nummer='C0815'>`
	`  ...`
	`</cocktail>`

`ENTITY / ENTITIES`	Bezug auf den Namen einer generellen, externen, nicht parsbaren Entität – Zum Beispiel ein Bild.
	
	`<!ATTLIST cocktail foto ENTITY #IMPLIED>`
	
	`<cocktail foto='tequilasunrise.jpg'>`
	`  ...`
	`</cocktail>`

`NMTOKEN / NMTOKENS`	Zeichenkette, die nur aus Buchstaben, Ziffern, Punkt (.), Bindestrich (-), Unterstrich (_) und dem Doppelpunkt (:) bestehen darf (Name Token).
	
	`<!ATTLIST zutat menge NMTOKEN #REQUIRED>`
	
	`<zutat menge='1.5'>`
	`  Oliven`
	`</zutat>`

Typ	Erläuterung, Beispiel DTD und Beispiel XML-Satz

(Wert1|Wert2|...)	Aufzählung möglicher Attributwerte. Der Attributwert in einem XML-Satz muss einem der Werte aus dieser Liste entsprechen.
	`<!ATTLIST name klassifizierung`
	`(Cocktail|Longdrink|Collins) 'Cocktail'>`
	
	`<name klassifizierung='Longdrink'>`
	`  Tequila Sunrise`
	`</name>`

Für die Voreinstellung (*Default*) dürfen die folgenden Werte verwendet werden:

Voreinstellung	Erläuterung
`#REQUIRED`	In dem XML-Satz muss das Attribut immer angegeben werden.
`#IMPLIED`	Die Angabe des Attributs ist optional.
'wert'	Standardwert, falls das Attribut nicht angegeben wurde.
`#FIXED` 'wert'	Der einzig mögliche Wert für das Attribut.

5.4 Entitäten

Cuba Libre (Longdrink)

4 cl: Bacardi Rum weiß
1 l: Tropicola

Dekoration
Zitronenscheibe

Mixanleitung
Eiswürfel bis zum Rand in das Glas geben und mit Rum und Cola auffüllen.

Als Barbesitzer ärgern Sie sich natürlich über die EU-Verordnung zur Kennzeichnungspflicht in Lebensmitteln.[5] Sie tippen sich die Finger wund, weil bei jedem Ihrer Drinks eine Reihe von ellenlangen, fortwährend gleichen Zusatzstoff-Listen angehängt werden muss.

Den XML-Entwicklern ist dieser Umstand nicht verborgen geblieben, und so schufen sie *Entitäten*.

Entitäten sind Verweise auf Daten. Die Funktionsweise erinnert an *Textbausteine* – Entitätsnamen werden in einem XML-Satz oder innerhalb einer DTD durch den Inhalt der Entität ersetzt.

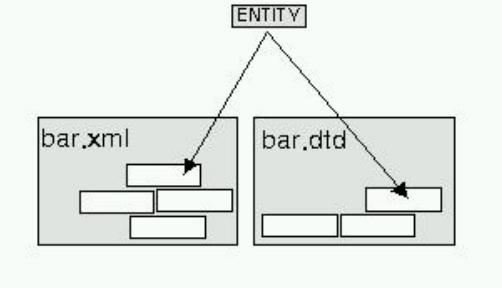


Abbildung 5.7: Entitäten – Textbausteine für XML und DTD

Generell unterscheidet man dabei zwei Arten: *Parameter Entitäten*, die nur innerhalb der DTD gültig sind, und *Generelle Entitäten*, gültig innerhalb der XML-Elemente.

[5] Bei verpackten Lebensmitteln werden Zusatzstoffe im Rahmen der Zutatenliste gekennzeichnet. Bei loser Ware muss ein Schild an oder neben dem Produkt auf die Verwendung von Zusatzstoffen hinweisen. In Gaststätten und Einrichtungen der Gemeinschaftsverpflegung müssen Zusatzstoffe in der Speisekarte, der Preisliste oder auf einem Aushang gekennzeichnet werden. Das kann auch in Form von Fußnoten geschehen. Siehe auch http://www.verbraucher.org/ bzw. http://www.zusatzstoffe-online.de/

5.5 Parameter-Entitäten

Parameter-Entitäten werden in der DTD definiert und gelten auch nur innerhalb der DTD.

`<!ENTITY % Entityname Entityinhalt>`

In unserer `bar.dtd` können wir nun immer wieder auftretende Geschmacksrichtungen als Textbaustein verwenden:

```
<!ENTITY % leckerwerte
  '(fruchtig|süß|trocken|spritzig|flach)'>
<!ELEMENT cocktail (#PCDATA)>
<!ATTLIST cocktail geschmack %leckerwerte;>
```

dies wird automatisch ersetzt zu:

```
<!ELEMENT cocktail (#PCDATA)>
<!ATTLIST cocktail geschmack
 '(fruchtig|süß|trocken|spritzig|flach)'>
```

! → Beachten Sie die etwas gewöhnungsbedürftige Notation mit dem Prozentzeichen (%) und dem Semikolion (;).

5.6 Generelle Entitäten

Generelle Entitäten sind für den Einsatz innerhalb der XML-Elemente gedacht, womit wir uns dem Problem mit der Kennzeichnungspflicht von Zusatzstoffen nähern.

`<!ENTITY Entityname Entityinhalt>`

In unserer `bar.dtd` können wir wieder häufig wiederkehrende Textteile deklarieren:

```
<!ENTITY enummern 'E412, E189, E503, E612, E613 und E614'>
```

die Anwendung findet innerhalb der XML-Datensätze statt

```
<zutat>
 3cl Creme de Cacao braun mit &enummern;
</zutat>
```

und dies wird ersetzt zu:

```
<zutat>
 3cl Creme de Cacao braun mit E412, E189, E503, E612, E613
und E614
</zutat>
```

! → Beachten Sie, dass im Gegensatz zu den Parameter-Entitäten hier in der DTD kein Prozentzeichen (%) stehen darf. Im XML-Datensatz wird mit dem kaufmännischen Und (&) und dem Semikolion (;) gearbeitet, so wie es von den Umlauten in HTML (ü = ü) her bekannt ist.

→ Kapitel 5.7 auf der nächsten Seite

Anstelle von Zeichenketten können auch Verweise auf externe oder öffentliche Daten verwendet werden. Mit parsbaren bzw. nicht-parsbaren Entitäten sind hier Text- bzw. Binärdaten (zum Beispiel ein Foto) gemeint:

Entityinhalt	Erläuterung
'Zeichenkette'	Interne Entität, 'E412, E189, E503, E612, E613 und E614'
`SYSTEM 'URI'`	Externe, parsbare Entität. Der URI verweist auf ein Dokument.
`PUBLIC 'FPI' 'URI'`	Öffentlicher Verweis auf eine externe, parsbare Datei und einen alternativen URI.
`SYSTEM 'URI' NDATA Notation`	Externe, nicht-parsbare Entität.
`PUBLIC 'FPI' 'URI' NDATA Notation`	Öffentlicher Verweis auf eine externe, nicht-parsbare Datei.

5.7 Notationen

Tom Collins (Collins)

5 cl: Gin
3 cl: Zitronensaft
2 cl: Zuckersirup
etwas: Sodawasser

Dekoration
Zitronenscheibe, Cocktailkirsche

Mixanleitung
Die Zutaten (ohne Sodawasser) mit Eiswürfeln im Shaker gut schütteln und in ein Longdrinkglas auf einige Eiswürfel abseihen. Mit etwas Sodawasser auffüllen. Mit einer Zitronenscheibe und einer Cocktailkirsche garnieren.

Mit Notationen werden externen, nicht-parsbaren Entitäten Hilfsanwendungen zugeordnet.

```
<!NOTATION Notationsname Hilfsanwendung>
```

In unserer `bar.dtd` definieren wir, dass das Programm `graphicstudio.exe` zu starten ist, sobald einem Cocktail ein Icon als Attribut übergeben wird:

```
<!ELEMENT cocktail (#PCDATA)>
<!ATTLIST cocktail icon ENTITY #IMPLIED>
<!ENTITY smiley SYSTEM 'smiley.jpg' NDATA showicon>
<!ENTITY sadly SYSTEM 'sadly.jpg' NDATA showicon>
<!NOTATION showicon SYSTEM 'graphicstudio.exe'>
```

In `bar.xml` erhält der Cocktail *Tom Collins* nun einen Smiley:

```
<cocktail icon='smiley'>
 Tom Collins
</cocktail>
```

Was passiert jedoch, wenn zur XML-Anwendung gar kein `graphicstudio.exe` installiert ist?[6]

[6] Zumindest hatten bis zum Zeitpunkt des Erscheinens dieser Broschüre weder Mix-Meister noch BarFachfrauen eine Windows-Version implantiert bekommen. Beide Angestelltengruppen können zwar lächeln, werden aber partout nicht gelb, womit der Smiley nicht darstellbar ist.

Die feste Zuordnung zwischen Hilfsanwendungen und Datenelementen bereitet den Autoren Zahnweh, zumal der offene Austausch von Daten sehr eingeschränkt wird. Ein solches Regelwerk lässt sich bestenfalls innerhalb eines homogenen Umfeldes – zum Beispiel innerhalb eines Firmen-Intranets mit fest vorgegebenen Rechner und Programmstrukturen – durchsetzen.

5.8 Aufgaben zur Selbstkontrolle

Versuchen Sie, möglichst ohne Zuhilfenahme der Unterlagen, die folgenden Fragen zu beantworten, bzw. die Sätze zu vervollständigen:

1. Ein XML-Regelwerk
   - (a) ☐ kann im Prolog einer XML-Datei stehen.
   - (b) ☐ kann beliebige Marken deklarieren.
   - (c) ☐ wird in einer DTD festgelegt.
   - (d) ☐ wird vom W3C festgelegt.
2. `<!ELEMENT Angestellte (BarFachfrau,MixMeister)*>`
   - (a) ☐ schafft jede Menge Arbeitsplätze.
   - (b) ☐ verlangt nur BarFachfrauen oder MixMeister.
   - (c) ☐ verlangt paarweise BarFachfrauen und MixMeister.
   - (d) ☐ verlangt beliebig viele BarFachfrauen oder MixMeister.
   - (e) ☐ verlangt eine Nennung in der Reihenfolge BarFachfrau und dann MixMeister.
   - (f) ☐ taucht nirgendwo auf den Web-Seiten des W3C auf.
3. Wie sieht die DTD-Anweisung für Attribute aus, mit der jedem Cocktail eine eindeutige Cocktailnummer zugeordnet wird?
   - (a) ☐ `<!ATTLIST cocktail cnr CDATA #REQ>`
   - (b) ☐ `<!ATTLIST cocktail cnr ID #IMPLIED>`
   - (c) ☐ `<!ATTLIST cnr ID #REQUIRED>`
   - (d) ☐ `<!ATTLIST cocktail cnr ID #REQUIRED>`
   - (e) ☐ `<!ATTLIST cocktail cnr ID '42'>`
   - (f) ☐ `<!ATTLIST cocktail cnr W3C>`
4. Entitäten
   - (a) ☐ eignen sich als Textbausteine.
   - (b) ☐ können auch auf nicht parsbare Daten (Bilder) verweisen.
   - (c) ☐ werden in einer DTD festgelegt.
   - (d) ☐ werden vom W3C untersagt.

Eingaben löschen | Aufgaben lösen

6 Unbekannte Rezepturen

6.1 Namensräume – oder: Der Ehren-Codex der Barmixer

Gin Fizz (Cocktail)

2/3:	Gin
1/3:	Zitronensaft
1 EL:	Zuckersirup
etwas:	Sodawasser

Dekoration
Zitronenscheibe, Cocktailkirsche

Mixanleitung
Die ersten drei Zutaten im Shaker inklusive Crushed Ice schütteln, dann in den Tumbler abseihen. Mit Soda den Longdrink auffüllen und mit Kirsche und Zitronenscheibe dekorieren.

Mittlerweile ist unsere `bar.xml` wohlgeformt und mit Hilfe der `bar.dtd` auch gültig. Alle XML-Elemente und Attribute sind damit namentlich eindeutig definiert – man spricht in diesem Fall vom *Namensraum*, den eine DTD bildet. Im Beispiel gehören alle Cocktails dem Namensraum unserer Bar an, der `bar.dtd`.

Was geschieht nun, wenn keine DTD vorhanden ist? Was, wenn XML-Elemente aus fremden Namensräumen, oder mehrere Namensräume gleichzeitig benutzt werden sollen? Besonders problematisch wird es, wenn sich dabei Elementnamen und Attribute überschneiden und ins Gehege kommen. Bei mehreren gleichnamigen Elementen, sind Konflikte innerhalb der XML-Anwendung vorprogrammiert.

→Beispiel 6.1 auf der nächsten Seite

Ein Beispiel: Zum Feierabend bestellt sich der letzte Gast bei unserem MixMeister einen *Gin Fizz*.

So richtig wach ist der Mann hinter dem Tresen nicht mehr. Wie war der Cocktailname noch gleich? *Gin Fizz* oder *Sodawasser*? Und musste `<zutat>-<name>` nicht ein Attribut `klassifizierung` haben? Welche Regelwerke sich hinter einem `<cocktail>-<name>` oder einem `<zutat>-<name>` verbergen, geht aus dieser XML-Datei nicht hervor. Um es vorweg zu nehmen: Der Mixmeister serviert dem Gast wahrscheinlich ein Glas Milch, weil ihm die passende DTD fehlt.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<cocktail>
 <name> Gin Fizz </name>
 <zutat>
  <menge> 2/3 </menge>
  <name> Gin </name>
 </zutat>
 <zutat>
  <menge> 1/3 </menge>
  <name> Zitronensaft </name>
 </zutat>
 <zutat>
  <menge> 1 EL</menge>
  <name> Zuckersirup </name>
 </zutat>
 <zutat>
  <menge> etwas </menge>
  <name> Sodawasser </name>
 </zutat>
</cocktail>
```

Beispiel 6.1: Gin Fizz gegen Feierabend

6.1.1 Qualifizierte Namen

Als Antwort auf fehlende DTD-Angaben hat das W3C im XML-Standard *qualifizierte Namen* (*qualified names*) eingeführt. Mit Ihnen werden in der XML-Datei die bisher bekannten Elemente um einen Namensraum erweitert.

`<`*`Elementname`* `xmlns:`*`URI`*`>`

Das Attribut `xmlns:` steht dabei für *XML name space*. Der URI bestimmt, auf welchen Namensraum sich das Element und alle darin enthaltenen Unter-Elemente beziehen. Eine hier genannte Adresse – sie muss nicht wirklich existieren – dient nur als Namenskonvention und ist quasi ein Ehren-Codex zwischen den Erzeugern der Daten und den Programmierern einer XML-Anwendung, die an beide appelliert: *Haltet euch an die gleichen Regeln*. Der URI steht für den Namen des Regelwerkes.

Für unser Beispiel formuliert der Gast seine Bestellung neu:

> *Mach mir einen Gin Fizz nach den Anleitungen von* `http://www.bar.de/cocktail` *und verwende Zutaten aus dem Fundus von* `http://www.zutaten.de`.

Die entsprechende `bar.xml` sieht dann so aus:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<cocktail xmlns:http://www.bar.de/cocktail>
 <name> Gin Fizz </name>
 <zutat xmlns:http://www.zutaten.de>
  <menge> 2/3 </menge>
  <name> Gin </name>
 </zutat>
 <zutat xmlns:http://www.zutaten.de>
  <menge> 1/3 </menge>
  <name> Zitronensaft </name>
 </zutat>
 ...
</cocktail>
```

Der Cocktail gelingt, die Milch wandert in den Ausguss.

! → An dieser Stelle sei nochmals auf die mögliche Nichtexistenz eines URI hingewiesen. Es handelt sich **nicht** um eine Auflistung von externen DTD-Adressen, weshalb das Namensraumkonzept bei der Validierung auch komplett ignoriert wird.

6.1.2 Mehrere Namensräume gleichzeitig verwenden

Waren im vorhergehenden Beispiel die Namensräume noch hierarchisch voneinander getrennt[1], so lassen sich mit leicht modifizierten Notation auch mehrere `xmlns` auf der gleichen Ebene miteinander vermischen:

```
<Elementname1 xmlns:Elementname2=URI
              xmlns:Elementname3=URI ...>
```

Entsprechend kann auch unsere `bar.xml` gestaltet werden:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<bar xmlns:cocktail=http://www.bar.de/cocktail
         xmlns:zutat=http://www.zutaten.de>
 <cocktail:name> Gin Fizz </cocktail:name>
  <zutat:menge> 2/3 </zutat:menge>
  <zutat:name> Gin </zutat:name>
 ...
</bar>
```

[1] Der `name`-Tag innerhalb einer `zutat` unterscheidet sich i.A. von dem in der Datenstruktur darüberliegenden Cocktail-`name`.

Außer Cocktail-Regelwerken kann man natürlich auch richtige Standards einsetzen. Falls Sie HTML4.0-Tags in Ihrem XML-Dokument verwenden wollen, so gilt der offizielle Namespace des W3C:

```
<html xmlns:'http://www.w3.org/TR/REC-html-40'>
```

! → Und auch hier noch einmal: Unter diesem URI finden Sie zwar keine DTD, Sie vereinbaren und akzeptieren jedoch stillschweigend, dass es sich bei den unterhalb vom <html>-Tag befindlichen Elementen um die bekannten <h1>, <p>, <hr> usw. handelt. Nach diesem Regelwerk schreiben Sie HTML-Seiten und nach diesem Regelwerk verhält sich auch jeder Web-Browser.

Wenn Sie neue HTML-Tags erfinden, so wird diese kein Browser anzeigen (können) sondern ignorieren, weil sie nicht im W3C-Regelwerk definiert sind.

6.1.3 Namensräume: Unser Fazit

Sicher haben Sie es bemerkt: Namensräume sind für den persönlichen Einsatz zu unpräzise und eignen sich eher zur Realisierung von überregionalen Standards. Die Autoren haben übrigens ein C2[2] gegründet und geben demnächst die *Cocktail Meta Language* (kurz CockML) heraus.

Für überschaubarere Einsatzbereiche, wie beispielsweise Firmen-Intra- und Extranets, eignen sich DTD-Anweisungen besser.

[2] *C2: Cocktail Consortium*

7 Auf Ex — Anwendungsbeispiele

Mike Collins (Collins)

6 cl: Irish Whiskey
3 cl: Zitronensaft
2 cl: Zuckersirup
etwas: Sodawasser

Dekoration
Zitronenscheibe, Cocktailkirsche

Mixanleitung
Die Zutaten – ohne Sodawasser – im Shaker gut schütteln und in ein Longdrinkglas auf einige Eiswürfel abseihen. Mit etwas Sodawasser auffüllen. Mit einer Zitronenscheibe und einer Cocktailkirsche garnieren.

„Another party is over…“ sang vor mehr als zwanzig Jahren Freddy Mercury in seinem Melancholy Blues, und er hatte wie so häufig Recht. Auch unsere Cocktailparty neigt sich langsam seinem Ende entgegen und wenn es noch eine offene Frage gibt,[1] dann sicherlich die, wofür der ganze bislang betriebene Aufwand denn gut gewesen sein soll.

Bislang haben wir eine praktische Umsetzung von XML-Daten kennengelernt, nämlich die auf Seite 26, wo wir mit Hilfe von Cascading Style Sheets XML-Daten für die Darstellung innerhalb eines WWW-Browsers aufbereitet haben. Im folgenden Abschnitt machen wir etwas sehr Ähnliches: Wiederum ist unser Zielmedium ein WWW-Browser, doch diesmal wird die Konvertierung durch ein PHP-Skript bereits auf dem Server vorgenommen (weshalb solche Lösungen auch *Serverside* genannt werden), und der Browser erhält bereits den fertigen HTML-Code.

Wofür das Ganze gut sein soll? Zum einen kann so die Darstellung unseres Dokuments sehr viel dynamischer werden. Denken wir an den Dagobert-Duck-Sparcocktail zurück, so stellen wir fest, dass die Mengenangabe immer *hinter* der Substanz erscheint – einfach aus dem Grund, weil diese Reihenfolge im XML-Dokument festgelegt ist, und wir durch Cascading Style Sheets keinen Einfluss darauf haben. Durch den Einsatz von Serverside-Konstrukten bieten sich da ganz andere Möglichkeiten, wie wir gleich noch feststellen werden.

Nicht vergessen sollten wir auch, dass es immer noch Anwender gibt, die sich nicht von ihrem Netscape 4.0 trennen wollen, für den wiederum CSS gänzlich unbekannt sind.

[1] Wahrscheinlich gibt es jede Menge, aber wir werden nur noch die eine beantworten.

7.1 Serverside xml2html mit PHP

Selbstverständlich gibt es auch Serverside xml2html-Lösungen für die beliebte Scriptingsprache PHP. Wie so oft wird man unter

`http://www.hotscripts.com/`

fündig. Dort stößt man unter anderem auf Michael P. Mehls XML-Klasse `phpxml-1.0`. Siehe auch

`http://www.phpxml.org/`

Eine einfache Cocktail-Anwendung inklusive Anzeige und Suchfunktion ist mit wenigen Zeilen PHP-Code realisiert. Der einleitende HTML-Code, das Suchformular sowie der detaillierte Tabellenaufbau entfallen hier aus Platzgründen; das vollständige Listing findet sich in Anhang B.2.

```
<!DOCTYPE HTML PUBLIC ¨-//W3C//DTD HTML 4.0 Transitional//EN¨>
<html>
...
<?php
  include(“../../source/xml.php“); // Include phpXML Klasse
  $xml = new XML(“bar.xml“); // XML object anlegen.

  if ( !empty($suchabfrage) )
  { // Nach Suchkriterium anzeigen
   $getraenke = $xml->evaluate(
   “//cocktail/*[contains(., $suchabfrage)]/..“);
  }
  else
  { // Alle anzeigen
   $getraenke = $xml→evaluate(“//cocktail“);
  }
  foreach ( $getraenke as $cocktail )
  { // durch alle XML-Datensaetze
   $name = $xml→get_content($cocktail.“/name[1]“);
?>
   <tr valign=top>
   <td><?php echo $name; ?></td>
   ...
```

→Abbildung 7.1 auf der nächsten Seite

Das PHP-Skript erzeugt damit serverseitig HTML-Code, der von jedem Browser angezeigt werden kann.

7.2 xml2tex mit Java

Als Alternativbeispiel, und nicht zuletzt um zu zeigen, dass XML-Anwendungen nicht doch immer wieder als konvertierte HTML-Ausgabe

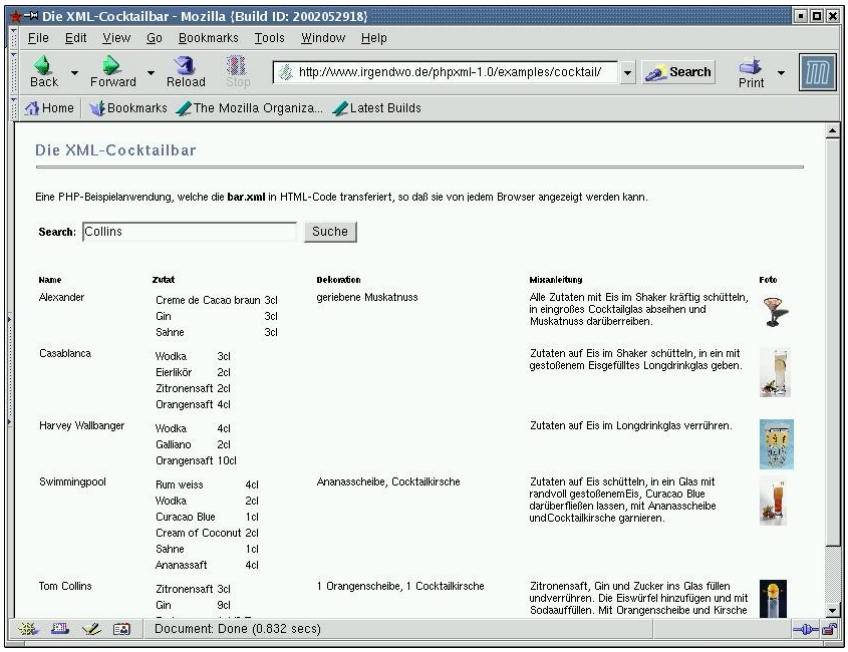


Abbildung 7.1: Cocktailrezepte mit phpxml-1.0

in einem Browserfenster landen, wollen wir nun zumindest prinzipiell demonstrieren, wie wir unsere Cocktailliste in ein schickes Rezeptbuch verwandeln können.

Wo liegt dabei überhaupt das Problem? Bekanntlich verfügt jeder WWW-Browser über eine eingebaute Print-Routine. Sofern Sie jedoch wie wir der Meinung sind, dass Schriftsatz etwas mehr als eine Aneinanderreihung von Buchstaben ist, sollten Sie weiterlesen – schließlich haben wir im oberen Absatz bereits das Attribut *schick* benutzt.

Angedacht ist also eine Konvertierung unserer XML-Cocktails zu einer druckfähigen Rezeptsammlung. Um eine optimale Druckaufbereitung zu erzielen, und um uns nicht mit mehr Arbeit als nötig zu belasten, bedienen wir uns des Schriftsatzsystems LaTeX als Backend, und zwar aus folgenden Gründen:

1. sind in LaTeX immer noch die besten Satzalgorithmen enthalten. Unsere Rezeptsammlung soll ja auch „nach was aussehen“.
2. besitzt LaTeX eine ähnliche Auszeichnungsstruktur wie XML. Eine Konvertierung kann sich daher (beinahe) auf den simplen Austausch von Marken beschränken.

7.2.1 Was wird benötigt?

Grundsätzlich benötigen wir die Java-Pakete `javax.xml.parsers`, sowie `org.xml.sax` die für uns den benötigten Parser bereitstellen. Beide Pakete sind seit der Version 1.4 in der kostenlosen Java 2 Standard Edition (j2SE) enthalten.

! → Um dieses Kapitel nicht ausufern zu lassen, haben wir darüberhinaus den Java-Quellcode auf die nötigsten Anweisungen komprimiert. Sie sollten daher zumindest *rudimentäre Java-Kenntnisse* besitzen um das folgende Beispiel zu verstehen. Das komplette nicht weiter kommentierte Programm-Listing finden Sie im Anhang B.3.

7.2.2 Wie sieht die Java-Klasse aus?

In einem ersten Schritt müssen die oben aufgeführten Pakete in eine eigene Klasse, in unserem Beispiel `BarListe.java`, eingebunden werden. `Barliste` selbst ist wiederum eine Ableitung der Standardklasse `DefaultHandler`.

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;

public class BarListe extends DefaultHandler{

 private String elementname;
 private String attrib_cocktail;
...
}
```

Innerhalb unserer Klasse können nun die geerbten Methoden `startDocument`, `startElement`, `characters`, `endElement` und `endDocument` überschrieben werden, die für die Umsetzung des XML-Codes verantwortlich sind. Das Verfahren gestaltet sich dabei wie folgt:

Durch die Instanzierung unserer Klasse wird automatisch ein Parser gestartet, der als Argument den Namen unseres XML-Dokuments übernimmt.

```
public class BarListe extends DefaultHandler{
...
 public static void main(...){
  SAXParserFactory saxpafac =
   SAXParserFactory.newInstance();
  saxpafac.setValidating(true);

  SAXParser saxpa = saxpafac.newSAXParser();
  saxpa.parse(args[0], new BarListe());
...
 }
}
```

Die Methode `parse` übernimmt in diesem Beispiel den Namen des XML-Dokuments aus der Kommandozeile und instanziert unsere Klasse `BarListe`. Über die Methode `setValidating` wird festgelegt, ob der Parser das Dokument zusätzlich auf Gültigkeit prüfen soll.

Die oben aufgeführten Methoden werden nun vom Parser in Abhängigkeit des eingelesenen Dokuments aktiviert, beispielsweise `startDocument` beim Beginn der Leseoperation oder `startElement` sobald eine öffnenden Marke (wie `<name>`) auftaucht.

```
public class BarListe extends DefaultHandler{
 public void startDocument() {
  System.out.println("\\documentclass{article}");
  ...
 }
}
```

Entsprechend komplizierter wird die Angelegenheit, sobald die ersten öffnenden Marken gefunden werden.

```
...
public void startElement(String nsu, String ln,
String qn, Attributes attr) {
 elementname = qn;
 if (elementname.equals("name")) {
  System.out.println("\\section{");
  attrib_cocktail = attr.getValue("klassifizierung");
 }
 ...
}
```

Der Methode `startElement` wird also jeweils im dritten Argument der Name des gefundenen Element übergeben, der dann im Methodenrumpf per `if`-Anweisung abgefragt werden kann. In unserem Beispiel wird jeweils bei jedem Fund der Marke `<name>` die LaTeX-Anweisung `\section{` geschrieben, also eine neue Überschrift erzeugt. Daneben kann über die Methode `getValue` der Wert des Attributs `klassifizierung` gesichert werden.

Der eigentliche Inhalt des Elements – also Character Data, oder um beim Beispiel zu bleiben, der Name des Cocktails, wird durch die Methode `characters` lokalisiert.

```
...
public void characters(char[] cont, int start,
int len) {
 data = new String(cont, start, len).trim();
 if (elementname.equals("name")) {
  System.out.println(data);
 }
 ...
}
```

Die Aufgabe die abschließende `}` zu setzen, um den `\section`-Befehl zu vervollständigen, bleibt der Methode `endElement` überlassen. Ähnliches gilt für die Methode `endDocument` welche am Ende des XML-Dokuments aufgerufen wird und die folgerichtig auch die letzte Zeile der LaTeX-Datei (`\end{document}`) schreibt.

Der auf diese Weise entstandene LaTeX-Quellcode schaut nicht unbedingt strukturiert aus, ist jedoch voll funktionsfähig wie Abbildung 7.2 zeigt.

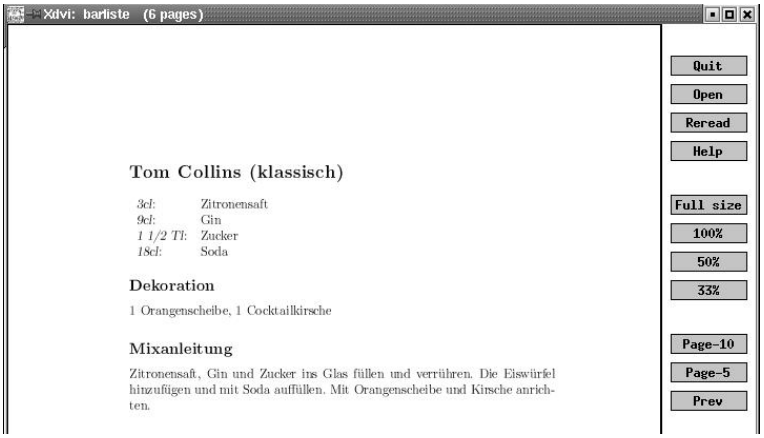


Abbildung 7.2: Aus XML wird mit Hilfe von Java ein „echtes“ Rezeptbuch.

Anhang A Auflösung aller Aufgaben

Aufgaben 2.4 auf Seite 7: 1 (b); 2 (c); 3 (e)

Aufgaben 3.7 auf Seite 18: 1 (Falsch); 2 (Korrekt); 3 (Falsch); 4 (Korrekt); 5 (Falsch); 6 (Korrekt); 7 (Falsch)

Aufgaben 4.5 auf Seite 30: 1 (a); 2 (c); 3 (a); 4 (d)

Aufgaben 5.8 auf Seite 51: 1 (a) oder (c); 2 (c); 3 (d); 4 (a), (b) oder (c)

Anhang B Programm-Listings

B.1 xml2html mit XSL

```
<?xml version='1.0' encoding=\"ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
    <html>
    <head><title>Cocktail Bar</title></head>
    <body>
    <H1>Cocktail Bar</H1>
    <H4>Inhaber: Feuerstack &amp;amp; Vieler </H4>
    <table border="2" bgcolor="lightblue">
    <xsl:for-each select="bar/cocktail" order-by="+ name">
       <tr>
          <td rowspan="2" valign="top">
          <H3><xsl:value-of select="name"/></H3>
          </td>
          <td colspan="2">
          Zutaten:
              <ul>
              <xsl:for-each select="zutat">
              <li>
                 <xsl:value-of select="menge"/>
                 <xsl:value-of select="substanz"/>
              </li>
              </xsl:for-each>
              </ul>
           Dekoration: <xsl:value-of select="dekoration"/>
          </td>
       </tr>
       <tr>
          <td>
             <P><xsl:value-of select="mixanleitung"/></P>
         </td>
       </tr>
    </xsl:for-each>
    </table>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

B.2 Serverside xml2html mit PHP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- cocktail-bar mit PHP -->

<html>

<head>
  <title>Die XML-Cocktailbar</title>

  <style type="text/css">
  <!--

    a, a:link, a:visited, a:active, a:focus
    {
        color : #6E749F;
        background : transparent;
        text-decoration : none;
        font-weight : bold;
    }

    a:hover
    {
        color : #6E749F;
        background : transparent;
        text-decoration : none;
        font-weight : bold;
    }

    body
    {
        margin : 0px 0px 0px 0px;
        background : #FFFFFF;
        font : 10pt Verdana, Geneva, Arial, Helvetica, sans-serif;
    }

    hr
    {
        color : #000000;
        background : transparent;
        width : 100%;
        height : 1px;
    }

    span.description
    {
        background : transparent;
        color : #000000;
        background-attachment : scroll;
        line-height : 150%;
    }

    span.header
    {
        background : transparent;
```

```
            color : #6E749F;
            font : bold 13pt Tahoma, Verdana, Geneva, Arial,
                   Helvetica, sans-serif;
            letter-spacing : 1px;
        }

        span.small
        {
            background : transparent;
            color : #000000;
            font : 8pt Tahoma, Verdana, Geneva, Arial, Helvetica, sans-serif;
        }

        td
        {
            background : transparent;
            font : 10pt Verdana, Geneva, Arial, Helvetica, sans-serif;
        }

        td.content
        {
            padding : 20px 20px 20px 20px;
            background : #FFFFFF;
            color : #000000;
            vertical-align : top;
        }

      -->
      </style>
    </head>

    <body>

    <form action="<?php echo $PHP_SELF; ?>" method="post">

    <table width="100%" border="0" cellspacing="0"
           cellpadding="0" align="center">
      <tr>
        <td class="content">
          <span class="header">
            Die XML-Cocktailbar
          </span>

          <hr><br>

          <span class="description">
            Eine PHP-Beispielanwendung, welche die <b>bar.xml</b> in
            HTML-Code transferiert, so dass sie von
            jedem Browser angezeigt werden kann.
          </span>

          <br><br>

          <table border="0" cellspacing="2" cellpadding="2">
            <tr>
```

```
            <td><b>Search:</b></td>
            <td><input type="text" name="term" value="<?php echo $term; ?>"
                       size="30" maxlength="30"></td>
            <td><input type="submit" value="Suche"></td>
          </tr>
        </table>

        <br><br>

        <table width="100%" border="0" cellspacing="2"
              cellpadding="2" align="center">
          <tr>
            <td><span class="small"><b>Name</b></span></td>
            <td><span class="small"><b>Zutat</b></span></td>
            <td><span class="small"><b>Dekoration</b></span></td>
            <td><span class="small"><b>Mixanleitung</b></span></td>
            <td><span class="small"><b>Foto</b></span></td>
          </tr>

          <?php

              // Include the <phpXML/> class.
              include("../../source/xml.php");

              // Create an XML object for the XML file.
              $xml = new XML("bar.xml");

              // Check whether a search term was given.
              if ( !empty($term) )
              {
                  // Only select those cocktail, in whose name or position
                  // the search string is present.
                  $government = $xml->evaluate(
                      "//cocktail/*[contains(., $term)]/..");
              }
              else
              {
                  // Select all members of the government.
                  $government = $xml->evaluate("//cocktail");
              }

              // Run through all members of the government.
              foreach ( $government as $cocktail )
              {
                  // Retrieve information about the cocktail.
                  // and Display the information.
                  $name  = $xml->get_content($cocktail."/name[1]");

                  ?>
                  <tr valign=top>
                    <td><?php echo $name; ?></td>
                    <td><table>
                  <?php

                  $zutat = $xml->evaluate($cocktail."/zutat");
```

```
                    foreach ( $zutat as $unddazu )
165                 {
                        $substanz  =
                           $xml->get_content($unddazu."/substanz[1]");
                        $menge  = $xml->get_content($unddazu."/menge[1]");
                    ?>
170                   <tr><td><?php echo $substanz; ?></td>
                          <td><?php echo $menge; ?></td>
                      </tr>
                    <?php
                    }
175                 ?>
                      </tr></table></td>
                    <?php
                    $dekoration =
                       $xml->get_content($cocktail."/dekoration[1]");
180                 $mixanleitung =
                       $xml->get_content($cocktail."/mixanleitung[1]");
                    $foto = $xml->get_content($cocktail."/foto[1]");

                    ?>
185                   <td><?php echo $dekoration; ?></td>
                      <td width=30%><?php echo $mixanleitung; ?></td>
                      <td><?php echo "<IMG height=50 SRC="$foto">"; ?>
                      </td>
                    </tr>
190
                    <?php
                }

            ?>
195
          </table>
        </td>
      </tr>
    </table>
200
    </form>
    </body>
    </html>
```

B.3 xml2tex mit Java

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;

public class BarListe extends DefaultHandler{

        private String elementname;
        private String attrib_cocktail;

        private String normaldata = "";
        private String substanz = "";
        private String menge = "";
        private boolean zutatflag = false;


        public void startDocument() throws SAXException{

            System.out.println("\\documentclass[12pt]{article}");
            System.out.println("\\usepackage[latin1]{inputenc}");
            System.out.println("\\usepackage{german}");
            System.out.println("\\begin{document}");

        }

        public void startElement(String nsu, String ln,
                                 String qn, Attributes attr)
        throws SAXException{

                elementname=qn;

                if (elementname.equals("name")) {
                    System.out.print("\n\\section*{");
                }

                if (elementname.equals("mixanleitung")) {
                    System.out.println("\n\\subsection*{Mixanleitung}");
                }

                if (elementname.equals("dekoration")) {
                    System.out.println("\\end{tabular}");
                    zutatflag = false;
                    System.out.println("\n\\subsection*{Dekoration}");
                }

                if(elementname.equals("zutat")) {
                        menge = "";
                        substanz = "";

                        if (!zutatflag) {
                            System.out.println("\n\\begin{tabular}{ll}");
                            zutatflag = true;
                        }
```

```
                }

                if(elementname.equals("name")){
                        attrib_cocktail =
                            attr.getValue("klassifizierung");
                }

                normaldata = "";

        }

        public void characters(char[] cont, int start, int len)
        throws SAXException{

                if(elementname.equals("name") ||
                   elementname.equals("dekoration") ||
                   elementname.equals("mixanleitung")) {
                        normaldata = normaldata +
                          new String(cont, start, len).trim() + "\n";
                }

                if (elementname.equals("substanz")) {
                    substanz = new String(cont, start, len);
                }
                if (elementname.equals("menge")) {
                    menge = new String(cont, start, len);
                    System.out.println("\\textsl{" + menge +"}: & " +
                       substanz + "\\\\");
                }

        }

        public void endElement(String nsu, String ln, String qn)
        throws SAXException{

            elementname=qn;

            if (elementname.equals("name")) {
                normaldata = normaldata.trim();
                System.out.println(normaldata +
                    " (" + attrib_cocktail + ")}");
            }

            if ((elementname.equals("dekoration")) ||
                (elementname.equals("mixanleitung"))) {
                normaldata = normaldata.trim();

                if (normaldata.equals("")) {
                    System.out.println("---");
                } else {
                    System.out.println(normaldata);
                }
            }

            if (elementname.equals("cocktail")) {
```

```
                System.out.println("\n\\newpage");
            }

        }


        public void endDocument() throws SAXException{

            System.out.println("\\end{document}");

        }

        public void warning(SAXException spex){
                System.err.println("Warnung!!!!!!");
                spex.printStackTrace(System.err);
        }

        public void error(SAXException spex){
                        System.err.println("Fehler!!!!!!");
                        spex.printStackTrace(System.err);
        }

        public void fatalError(SAXException spex){
                        System.err.println(\"Abbruch!!!!!!");
                        spex.printStackTrace(System.err);
        }


        public static void main(String[] args){
                SAXParserFactory saxpafac =
                       SAXParserFactory.newInstance();
                saxpafac.setValidating(true);


                try{
                        SAXParser saxpa = saxpafac.newSAXParser();

                        saxpa.parse(args[0], new BarListe());

                } catch (ParserConfigurationException pcex){
                        pcex.printStackTrace(System.err);
                } catch (SAXException sex){
                        sex.printStackTrace(System.err);
                } catch (IOException ioex){
                        ioex.printStackTrace(System.err);
                }
        }
    }
```

Anhang I Index