13

4,40 Deutschland

KnowWare

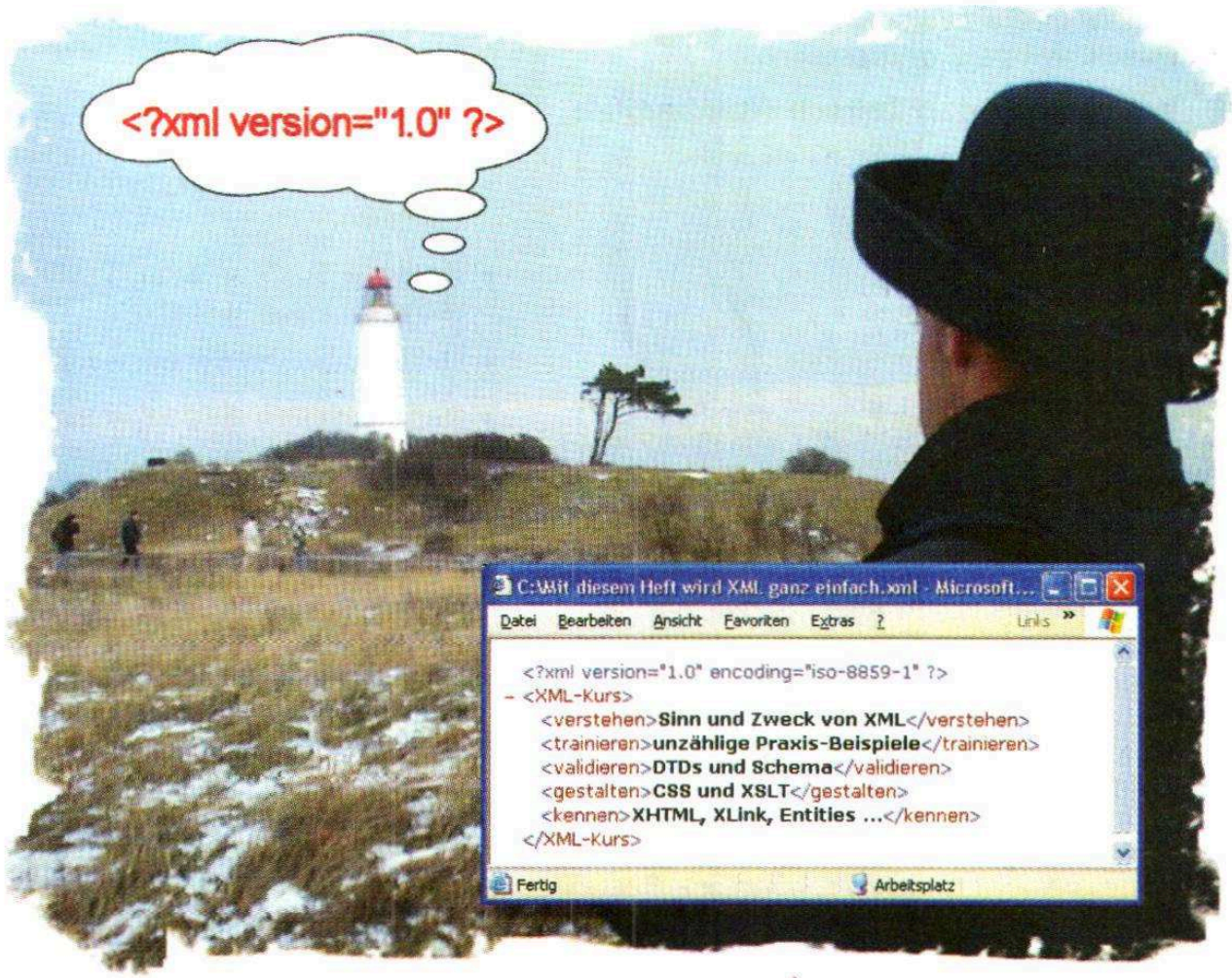
Extra!

XML

Viele Beispiele und Übungen

# XML

## leicht & verständlich




Johann-Christian Hanke

www.KnowWare.de

Deutschland: 4,40 EUR Österreich: 5,00 EUR
Schweiz: 8,60 SFR Luxemburg: 5,20 EUR Italien: 6,00 EUR

4 195087 904405 06013

KnowWare Extra!

# XML leicht & verständlich

## ■ durchgesehener Nachdruck von „XML für Einsteiger"

Autor: Johann-Christian Hanke
ISBN 87-90785-89-4, 2006-03

Michael Maardt, verlag@knowware.de – Karl Antz, lektorat@knowware.de
Printer: OTM Denmark, Published by KnowWare

**Bestellung für Endverbraucher und Vertrieb für den Buchhandel**

KnowWare-Vertrieb
Postfach 3920
D-49029 Osnabrück
Tel.: +49 (0)541 33145-20
Fax: +49 (0)541 33145-33
knowware@amedis.de
www.knowware.de/bestellen

**Vertrieb für den Zeitschriftenhandel:**

DPV Network GmbH, PF 570412,
D-22773 Hamburg
Tel: (+49) 040 37845-6229
Fax: (+49) 040 37845-6247
www.dpv-network.de

**Worum es geht**

Hinter KnowWare steht der Gedanke, Wissen leichtverständlich und preisgünstig zu vermitteln.

**Wo und wann sind die Hefte erhältlich?**

Neue Hefte sind im Allgemeinen zwei Monate im Handel, und zwar bei Kiosken, im Bahnhofsbuchhandel und im Buchhandel – bei vielen Verkaufsstellen sowie im Buchhandel auch länger. Alle beim Verlag vorrätigen Titel kannst du immer bestellen.

**Bestellung**

- bei deinem KnowWarehändler - Bestellformular am Ende des Heftes ausfüllen!
- beim KnowWare-Vertrieb, siehe oben

**www.knowware.de**

- Beschreibungen und Bilder aller Hefte.
- Mehr als 100 kostenlose PDF-Dateien – zu jedem Heft gibt es eine kostenlose PDF-Datei der ersten 15-20 Seiten
- Ausverkaufte Hefte: das ganze Heft als PDF ist kostenlos
- Geplante Hefte
- Online-Bestellung
- Kostenloser Newsletter. Viele Vorteile für dich.
- Interne Suchfunktion. Du findest schnell, was du suchst.
- Informationen für neue Autoren
- Serviceseiten zu den Heften, hauptsächlich von den Autoren selbst.
- KnowWare in anderen Sprachen
- 500 Webseiten

www.knowware.de

# Inhaltsverzeichnis

# Vorwort: Herzlich willkommen zum XML-Kurs!

## XML ist erst in der Entwicklung!

Ich denke nicht gerne an meinen ersten XML-Kurs zurück. Der Grund: Ich hatte sie enttäuscht! Bisher waren sie mir treu gefolgt, meine Kursteilnehmer. Ich hatte ihnen die ersten Schritte beim Gestalten von Webseiten beigebracht, hatte sie „gequält" mit HTML und JavaScript und ... sie kamen wieder! Sie kamen wieder, denn die Arbeit an Webseiten machte ihnen Spaß.

Sie kamen wieder, denn von XML erhofften sie sich mehr. Wenn die ganze Welt davon redet, musste doch etwas dran sein. Sie suchten den ultimativen Kick für ihre Webseiten, doch sie merkten bald, dass ...

- XML nicht vorrangig zum Gestalten von Webseiten gedacht war und damit keinesfalls HTML ablösen würde. (Zumindest nicht in den nächsten Jahren.)
- XML im Gegensatz zu HTML erst entwickelt wird. Viele Dinge, die mit HTML seit Jahren selbstverständlich sind, gingen mit XML einfach nicht. Selbst simple Querverweise (Hyperlinks) funktionierten nicht einmal im Internet Explorer 5.5.
- XML alleine überhaupt keine Daten darstellt sondern nur die Struktur sichert.
- XML nur von den allerneusten Browsern unterstützt wird und schon deshalb für das Web allein eigentlich ungeeignet war.

Doch Bange machen galt nicht. Nach dem ersten Schock entwickelten meine Teilnehmer eine Art Pioniergeist.

Es kann auch spannend sein, wenn man sich mit einer neuen Technologie beschäftigt. Mit einer Technologie, die sich wohl erst in den nächsten Jahren voll durchgesetzt haben wird. Einer Technologie, von der man derzeit noch nicht sagen kann: „So ist es", sondern sagen muss „So wird es sein", oder schlimmer noch „So könnte es werden". Alles ist im Fluss.

Einiges von dem, was ich hier beschreibe, gilt derzeit erst als Empfehlung, nicht als verabschiedeter Standard. Änderungen vorbehalten!

## Was lernst du hier?

Schritt für Schritt mache ich dich mit den Eigenschaften und Vorteilen der neuen Seitenbeschreibungssprache (von der ich inzwischen durchaus überzeugt bin!) vertraut. Das Heft ist als Kurs aufgebaut, wir gehen beispielorientiert vor und üben zwischendurch!

Hier der Inhaltsüberblick:

- Die Väter von XML: SGML und HTML
- XML versus HTML
- Vorteile von XML als Sprache der Zukunft
- XML gibt es schon: WML, WSF und Co.
- XHTML als Neuformulierung von HTML
- XML-Praxis: wohlgeformte Dokumente
- Einführung in das Konzept der DTDs
- XML-Dateien gültig machen
- Der neue Standard: XML-Schema
- Test durch validierende Parser
- XML-Dokumente mit CSS gestalten
- Mehr Möglichkeiten durch XSLT

Leserservice! Ich biete dir alle Beispieldateien zum direkten **Download** an. Lade sie dir von der Serviceseite zum Heft www.jchanke.de/xml herunter. Dort findest du weiterführende Tipps und Infos zum Heft und ein Feedback-Formular.

## Was ist KnowWare?

KnowWare ist ein Projekt, Wissen auf möglichst einfache, verständliche und preiswerte Art zu vermitteln. Diese geniale Idee stammt von Michael Maardt, er gab 1993 in Dänemark sein erstes Heft „Nutze Deinen PC optimal" heraus.

Inzwischen gibt es die handlichen KnowWare-Hefte in vielen Ländern der Erde, allein auf Deutsch sind weit mehr als 100 Titel erschienen.

Über www.knowware.de kannst du dir von jedem Heft ein genaues Bild machen und Inhalt und Probeseiten herunterladen. Mehr über *meine* anderen KnowWare-Titel und über mich findest du auch auf www.jchanke.de/knowware.

Dein „XML-Tutor" Johann-Christian Hanke,
Berlin 2002, 2003 (und 2006)

# So fing alles an: SGML als Dokumentbeschreibungssprache

Vielfalt macht krank! Vielleicht kennst du das Drama auch: Du bekommst einen Text von einem Bekannten und ... kannst ihn nicht öffnen. Der Grund: Der Bekannte benutzt eine andere Textverarbeitung als du und es gibt keinen Filter.

Doch selbst wenn du die gleiche Textverarbeitung verwenden würdest, wäre der Erfolg noch nicht garantiert. Angenommen dein Bekannter benutzt eine neuere Version: Auch dann sieht es schlecht aus, denn dein Programm kennt das Format der neueren Version nicht.

## Der Turmbau von Babel

Und tatsächlich geht es bei der Computerei oft zu wie beim „Turmbau von Babel". Jeder Hersteller nutzt für seine Programme sein eigenes Format und alle paar Jahre ändert er dieses. Und selbst die raffiniertesten Filter schaffen es nicht immer, alle Eigenschaften verlustfrei von einem Format ins andere zu transferieren.

Was machst du, wenn du in zehn Jahren die Dokumente ansehen möchtest, die du damals verfasst hattest? Ob das so einwandfrei klappt? Einverstanden, bei Privatkorrespondenz mag das weniger tragisch sein, doch was passiert mit wichtigen Dokumenten von Behörden und Institutionen?

Dieses Problem ist nicht neu, sondern zeichnete sich schon vor Jahrzehnten ab. Deshalb befasste sich schon in den siebziger Jahren des vorigen Jahrhunderts ein gewisser Professor Charles F. Goldfab im Auftrag von IBM mit der Dokumentbeschreibung. Heraus kam die so genannte GML, die Generalized Markup Language.

Ziel der GML war es, Dokumente so zu beschreiben, dass das Ergebnis weder von einer bestimmten Plattform noch von einer spezifischen Anwendung abhing. Es ging nicht um das Aussehen eines Dokumentes, sondern um die logische Struktur wie die Einteilung in Überschriften, Kapitel, Seiten, Abschnitte.

## SGML wird zum ISO-Standard

GML wurde im Laufe der Jahre weiterentwickelt. Heraus kam SGML. SGML steht als Abkürzung für Standardized Generalized Markup Language, zu deutsch „allgemeine, standardisierte Sprache zur Textauszeichnung".

Die Entwicklungen gingen so weit, dass SGML 1986 als internationaler Standard anerkannt werden konnte, als ISO 8879.

**Wichtig zu wissen: SGML speichert nicht das Layout, sondern die logische Struktur von Dokumenten.**

Ziel von SGML war es tatsächlich, wichtige Dokumente unabhängig von ständig wechselnden binären Dateiformaten (Word, WordPerfect) oder Betriebssystemen (Windows, Mac-OS, Unix usw.) dauerhaft elektronisch sichern zu können.

Behörden, Firmen, Bildungseinrichtungen und Institutionen speichern Dokumente in SGML. Ein Bekannter von mir schwärmte noch vor Jahren, dass nun alle wichtigen wissenschaftlichen Arbeiten in SGML gesichert würden. Er arbeitete als Assistent an einer Berliner Universität.

Er war wahrscheinlich einer der wenigen, die damals schwärmten. Die meisten kannten SGML überhaupt nicht, trotz seiner offenbar überragenden Vorteile. Oder war da noch was?

## SGML ist kompliziert!

SGML hatte einen entscheidenden Schönheitsfehler: Es war zu kompliziert. Man hatte bei der Entwicklung an Verwaltungen und Behörden und nicht an „Otto Normaltexter" gedacht. Demzufolge waren die Software-Werkzeuge zum Erstellen von SGML teuer und nicht sehr weit verbreitet.

Und so wundert es kaum, dass SGML außerhalb der schon erwähnten Einrichtungen kaum Verbreitung fand. Doch dann kam das World Wide Web und die Welt veränderte sich ...

# HTML als Seitenbeschreibungssprache für das Web

Was war 1989/90? Richtig, der Mauerfall, die Wende. Doch darauf wollte ich gar nicht hinaus. Das World Wide Web wurde erfunden!

Und zwar nicht von Honecker, Gorbatschow oder Helmut Kohl, sondern von Tim Berners-Lee in der neutralen Schweiz. Dieser Mensch arbeitete als Informatiker an einem Kernforschungszentrum bei Zürich.

## WWW: Das Internet wird grafisch

Was die Meisten vergessen: Das Internet gab es zu der Zeit schon längst. Schließlich gehen die Ursprünge dieses Netzwerks bis in die 60er Jahre zurück! Die ersten E-Mails schickte man sich schon 1971 in den USA, beliebte Dienste im später weltweit operierenden so genannten „Internet" wurden neben E-Mail auch das Usenet (Diskussionsforen), der Dateidownload mit FTP, die Dateisuche mit Archie oder die WAIS-Datenbankrecherche.

Doch die Bedienung des Internets war ein Graus: Es handelte sich um einen durch und durch elitären Verbund für elitäre Menschen wie Militärs, Wissenschaftler und Universitäts-Mitarbeiter, und selbst die schafften es nicht immer.

Um Informationen abzurufen, musste man erst kryptische Befehle lernen und sich die Finger auf der Tastatur wund hämmern. Wer DOS kennt oder schon einmal mit reinen Unix-Rechnern zu tun hatte, weiß, was ich meine. Eine grafische Benutzeroberfläche gab es damals nicht.

Nichts für Otto Normalverbraucher also, ganz abgesehen davon, dass du oder ich zu dieser Zeit sowieso noch keinen Zugang bekommen hätten. Ich glaube, dass die meisten noch nicht einmal etwas von der Existenz des Internets wussten.

Das änderte sich spätestens mit der Erfindung von Tim-Berners Lee. Sie hieß: World Wide Web. Grundlage war eine Dokumentbeschreibungssprache namens HTML.

Mit dem World Wide Web kamen zum ersten Mal die bunten, anklickbaren Seiten. Es kam das, was man landläufig auch als Homepage bezeichnet.

Eine „Homepage" oder besser „Webseite" ist nichts weiter als ein Dokument mit Text, Querverweisen und Grafiken. Dieses Dokument oder vielmehr diese Dokumente liegen auf den Webservern im World Wide Web. Anschauen kann man sich das Ganze mit Hilfe eines Browsers. Das ist ein Anzeigeprogramm, welches man auf seinem heimischen Rechner installiert.

Die heute bekanntesten Browser heißen Internet Explorer, Netscape Navigator, Mozilla, Firefox und Opera.

Zum Transport der Daten vom Webserver zum Browser dient ein Protokoll namens HTTP, Hypertext Transfer Protocol. (Deshalb also das `http://` vor jeder Web-Adresse!)

Betrachte den Internet-Dienst *World Wide Web* ruhig als eine Art „grafische Oberfläche" für das Internet. Alles ist intuitiv bedienbar mit der Maus.

## HTML als Sprache für Webseiten

Nun haben wir die ganze Zeit über diese Dokumente geredet, interessant ist vor allem die Sprache, in der sie verfasst sind. Unterhalten wir uns deshalb nun über HTML, die Hypertext Markup Language.

Lee war nicht dumm und hat das Rad natürlich nicht neu erfunden. Er besann sich auf vorhandene Standards. Webseiten im HTML-Format sind eine Kombination von:

- ASCII
- und SGML.

ASCII ist sicher bekannt, die Abkürzung steht für American Standard Code for Information Interchange. Es handelt sich um ein standardisiertes, reines Textformat. ASCII ist praktisch der „kleinste gemeinsame Nenner" beim Datenaustausch zwischen Rechnern. Jedes einfache Schreibprogramm kann ASCII speichern bzw. lesen.

Ältere Computerhasen kennen ASCII vielleicht noch unter der Bezeichnung MS-DOS-Text.

## Die berühmten Tags

Nun kann man allein mit ASCII jedoch weder Überschriften kennzeichnen noch Eigenschaften wie fett, kursiv oder unterstrichen zuweisen.

Und hier griff Lee auf SGML zurück. In SGML werden Eigenschaften wie „Das ist eine Überschrift, dieser Text beinhaltet eine Aufzählung usw." mit so genannten Tags dargestellt. Diese „Markierungen" notiert man in spitzen Klammern.

**Es gibt stets ein Tag zum Einschalten und eines zum Ausschalten**

Wenn du eine Überschrift erster Ordnung darstellen möchtest, schreibst du:

```
<h1>Headline erster Ordnung</h1>
```

H1 ist dabei die Abkürzung für Heading 1, Überschrift erster Ordnung. Eine Überschrift zweiter Ordnung wäre eine H2 usw. usf. Es gibt auch Tags für Listen, Definitionen, fett, kursiv usw. usf.

Das große Verdienst von Lee war es, nur einen begrenzten Tag-Vorrat von SGML zu übernehmen. Er nutzte nur die Anweisungen, die man unbedingt brauchte. Damit wurde HTML zu einer vergleichsweise sehr einfach zu verstehenden Untermenge von SGML. Bald gab es auch die ersten allgemein erhältlichen Anzeigeprogramme für HTML, ich erinnere an den Browser Mosaic oder den frühen Netscape Navigator.

Nun heißt es zwar, dass sich weder SGML noch HTML vorrangig um das Layout kümmern. Im Gegenteil, es geht im Prinzip um die Struktur!

Allerdings kommen die gängigen Browser bei der Interpretation von HTML zu ähnlichen Ergebnissen, egal ob bei der Darstellung von Text, Überschriften oder Listen. Schon deshalb muss man den HTML-Tags gewisse Layouteigenschaften zugestehen.

Außerdem gibt es natürlich auch spezielle Anweisungen zum Festlegen von Schriftart, -größe, Form und Farbe. Dahinter verbirgt sich eine zusätzliche Layoutsprache namens CSS, Cascading Style Sheets, die HTML optimal ergänzt.

## Hyperlinks gehören auch dazu

Hätten wir doch beinahe die Querverweise vergessen, die „Sprungschanzen" zu anderen Seiten. Die gehören zum Web wie die sauren Gurken zum Spreewald.

Auch dafür hat sich Lee eine Notation ausgedacht, die mit dem Anchor-Tag (a wie Anker) beginnt. Ein Querverweis zur Seite von KnowWare sieht so aus:

```
<a href="http://knowware.de">Dieser
Link führt zu KnowWare</a>
```

Im Browser angezeigt wird dabei nur der Text Dieser Link führt zu KnowWare, in der Regel unterstrichen.

Jetzt wird dir sicher auch die Bedeutung der Abkürzung HTML klar. Hinter Hypertext Markup Language verbirgt sich die Auszeichnungssprache für „Hypertext-Dokumente", also für die Webseiten mit Hyperlinks.

## HTML wurde populär

HTML ist so einfach und verständlich, dass sich diese Sprache schnell durchgesetzt hat. Das World Wide Web und seine HTML-Seiten verhalfen dem Internet erst zu der Popularität, die es heute besitzt.

Für die Entwicklung der Sprachen und Standards für das Web ist übrigens ein Verband namens World Wide Web Consortium verantwortlich, kurz W3C. Was das W3C vorschlägt, wird wenige Jahre später in der Regel zum Industriestandard. Im W3C sind verschiedene Interessengruppen organisiert; auch Firmen wie Microsoft, IBM, Adobe oder Sun. Leiter des W3C ist Tim Berners-Lee, die Homepage des Verbands findest du unter www.w3.org.

HTML liegt derzeit in der Version 4.x vor. Die Sprache ist ausgereift. Eine ausführliche Einführung in HTML bekommst du in meinem Heft „Homepages für Einsteiger", in „Homepages mit HTML und CSS" oder natürlich bei Stefan Münz unter www.selfhtml.org.

HTML-Vorkenntnisse helfen beim Verstehen von XML, sind aber nicht Bedingung.

# Eigenschaften von XML als Sprache der Zukunft

Was SGML verwehrt blieb, hat der Ableger HTML spielend geschafft: HTML wurde ein Bestseller. Firmen, Institutionen und immer mehr Privatleute nutzen HTML, um ihre Inhalte ins Web zu stellen.

Die Anzeigeprogramme für HTML-Seiten werden immer besser, die Software-Werkzeuge auch. Zwar bevorzugt der Profi weiterhin die Handarbeit, doch längst können große Textverarbeitungen ihre Dokumente seit Jahr und Tag auch als HTML-Datei abspeichern.

**HTML hat sich als allgemeiner Web-Standard durchgesetzt.**

Und wenn HTML nun so großartig ist, warum bricht man dann nicht in lang anhaltenden Jubel aus und lässt es dabei bewenden?

## Was ist XML überhaupt?

Wieso musste man schon wieder etwas Neues erfinden? Warum nach SGML und HTML plötzlich XML? Was ist das überhaupt?

Die Abkürzung XML steht für eXtensible Markup Language. Das kann mit erweiterbare Auszeichnungssprache übersetzt werden. XML wurde 1998 vom World Wide Web Consortium (W3C) vorgestellt, die ersten Entwicklungen begannen aber schon 1996.

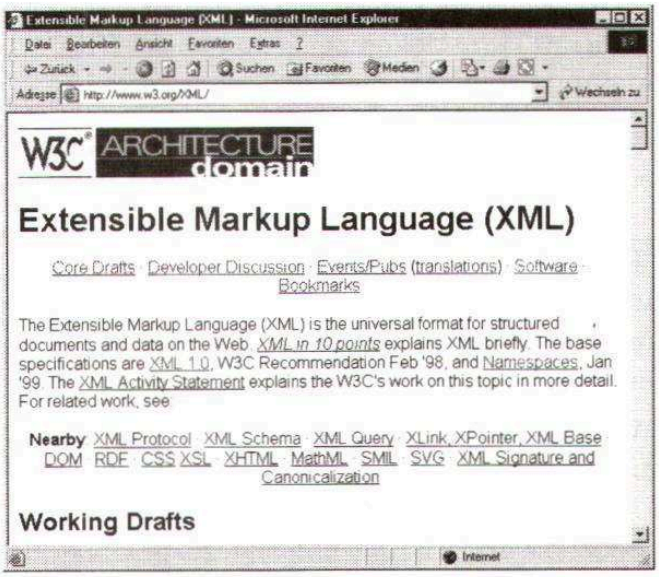
## Mythen und Fakten zu XML

Zu XML grassieren derzeit einige Mythen und falsche Annahmen in der Internet-Gemeinde.

Richtig ist, dass es sich bei XML um eine Dokumentbeschreibungssprache handelt. Das hat XML mit HTML oder SGML gemeinsam. Richtig ist auch, dass man seine eigenen Tags definieren kann! Jawohl, in XML bist du der Meister über die Dokumentstruktur!

Falsch ist allerdings die Annahme, XML sei in erster Linie der neue „Kick" zum Erstellen toller interaktiver Webseiten. XML ist weder der direkte Nachfolger von HTML noch eine reine Weblayout-Sprache im eigentlichen Sinne.

Ich bin sogar davon überzeugt, dass HTML bzw. die neue „HTML-Variante" XHTML in Sachen Weblayout noch auf Jahre hinaus der gängige Standard bleiben werden. XML setzt sich im Web erst langsam durch, da XML erst in Ansätzen unterstützt wird.




**Working Drafts beim W3C: XML ist noch eine Baustelle**

Doch ehe wir uns die weiteren Eigenschaften von XML anschauen, klären wir die Frage, warum man überhaupt eine neue Dokumentbeschreibungssprache brauchte!

## Nachteile von HTML

Um das zu verstehen, erörtern wir zuerst die Nachteile von HTML. Denn mit der Zeit wurde klar, dass auch HTML nicht das Non-plus-ultra war.

- HTML wurde für die Anzeige im Web-Browser gedacht, eignet sich daher weniger für den Druck oder für Layoutaufgaben. Seitenränder? Papierformat? Spaltensatz? Kopf- und Fußzeile? Inhaltsverzeichnis, Index und Querverweise? Selbst unter Mithilfe der Layoutsprache CSS kannst du HTML diese Eigenschaften kaum entlocken.
- Der Tag-Vorrat ist fest vorgegeben, dadurch ist diese Sprache nicht sehr flexibel. Was machst du, wenn du beispielsweise Vektorgrafiken oder Kalkulations-Tabellen in deinen Projekten brauchst? Nur mit HTML geht das nicht!

- HTML allein kann keine sich dynamisch verändernden Inhalte darstellen. Gerade das ist allerdings wichtig: Du brauchst aus einer riesigen Bestelldatenbank nur bestimmte Produkte? Fehlanzeige! Die Datendarstellung in HTML-Tabellen ist statisch und erst mit Hilfsmitteln wie Skriptsprachen (Perl, PHP usw.) kannst du Inhalte aus Datenbanken abfragen und in immer wieder neu zu erstellenden HTML-Dokumenten ausgeben.
- Struktur und Layout sind vermischt.

Gerade der letztgenannte Nachteil ist wohl der entscheidende! Was ist daran so nachteilig?

## Vermischung: Struktur und Layout

Ich hatte behauptet, dass sich HTML vorrangig um die Struktur kümmert, weniger um das Layout. Das stimmt auch, aber nur teilweise.

Wie schon erwähnt besitzt in HTML jedes Tag eine bestimmte eingebaute Eigenschaft. Eine Überschrift erster Ordnung `<h1></h1>` wird größer als eine `<h2></h2>` angezeigt. Überschriften und Absätze (`<p></p>`) schaffen sich außerdem einen bestimmten Abstand rundherum.

Eine mit den Tags `<b></b>` (b wie bold) formatierte Passage wird gefettet und das Tag `<hr>` erzeugt je nach Browser eine mehr oder weniger aufwändig gestaltete Linie. So könnte man alle Tags durchgehen und stellt schließlich fest, dass HTML in Grenzen auch Layoutaufgaben übernimmt.

**In HTML sind Struktur und Layout deshalb vermischt.**

Allerdings ist HTML nichts Halbes und nichts Ganzes: In Sachen Layout ist HTML wegen seiner begrenzten Gestaltungsmöglichkeiten nicht optimal. Deshalb arbeiten Webdesigner mit Trick 17 oder verwenden mit CSS-Style Sheets eine zusätzliche Layoutsprache.

Die strukturellen Nachteile hatten wir schon besprochen: Der Tag-Vorrat reicht nicht aus und die statische Datenverwaltung mit HTML-Tabellen ist alles andere als flexibel. Für zukünftige Aufgaben ist HTML ungeeignet.

## Beschränkung auf das Web

Aber auch heute ist HTML mit seiner Beschränkung auf das Web recht unflexibel.

Falls man die auf Web-Seiten dargestellten Informationen für andere Zwecke (Printkatalog, Audio-Ausgabe usw.) braucht, muss man diese erst umständlich in das jeweils andere Format konvertieren.

Umgedreht gilt das natürlich auch: Daten aus Textverarbeitungen, Katalogen oder Datenbanken müssen erst in HTML umgewandelt werden, damit sie im Web dargestellt werden können. Wäre es nicht toll, wenn man ein Basis-Format für alles hätte?

Dieses Basisformat gibt es nun!

## XML als reine Strukturspache

XML ist im Gegensatz zu HTML eine reine Strukturspache, die keinerlei Layouteigenschaften transportiert. Daraus folgt: XML eignet sich in Reinform zwar nicht zur Darstellung (Anzeige) von Daten. Dafür sichert es aber in ganz hervorragender Art und Weise die Struktur eines Dokumentes.

XML speichert nur die Daten, nur die Struktur. Die Gestaltung kann man mit Hilfe der entsprechenden Layoutsprache dem jeweiligen Zweck anpassen.

XML ist besonders ideal für Daten, die von vornherein strukturiert werden müssen. Ich denke an Adresslisten oder andere in Tabellenform zu verwaltende Daten wie Datenbanken, wissenschaftliche Belegarbeiten usw. usf.

Aber auch für normale Dokumente schaffst du eine Struktur, damit du diese Informationen in XML notieren kannst.

**Bevor du ein XML-Projekt beginnst, musst du stets zuerst die Struktur planen.**

Das Erdenken einer Struktur ist zugegebenermaßen recht aufwändig, viel aufwändiger als das simple Erstellen von HTML-Dokumenten mit einem grafischen HTML-Editor.

## Haupteigenschaften von XML

Dafür sind einige Eigenschaften von XML bestechend und gehen weit über das hinaus, was HTML kann:

- Du kannst, wie schon erwähnt, deine eigenen Tags definieren.
- Lege zu diesen Tags, falls gewünscht, sogar Attribute fest.
- Per Regelsatz beschreibst du diese Tags und Attribute exakt. Dieser Regelsatz nennt sich Dokumenttyp-Definition, kurz *DTD* oder *Schema*. Schema ist der neuste Standard.
- Struktur und Layout sind streng getrennt, ich erwähne das deshalb noch einmal, weil es so wichtig ist!

## Weitere Eigenschaften von XML

XML besitzt noch weitere Eigenschaften, die der Sprache Zukunftsfähigkeit verleihen:

### XML als freier, lizenzfreier Standard

Fangen wir mit dem größten Knüller an. XML ist ein offener, lizenzfreier Standard. Jeder kann dieses Dateiformat nutzen, jeder kann es verstehen, jeder kann es (mit etwas Mühe) lesen. Die Fakten liegen auf dem Tisch.

**Alle Standards sind auf der Seite des W3C veröffentlicht, unter www.w3c.org!**

Es ist also nicht mehr nötig für Programmentwickler, ihre eigenen, zum Konkurrenten unkompatiblen binären Dateiformate „zu entwickeln". Egal ob Textverarbeitung, Tabellenkalkulation oder Präsentationsprogramm: Man könnte sich auf einen Regelsatz einigen, der als allgemeines Format von allen Herstellern genutzt wird.

Ob die Hersteller das wirklich tun, wird sich zeigen. Zumindest Sun benutzt in seinem Office-Paket StarOffice 6/7 (Open Office 1/2) das XML-Format als „Hausformat". Dazu informiere ich dich später noch ganz ausführlich!

Zwar arbeitet auch die Microsoft-Office-Familie schon seit 1999 ansatzweise mit dem XML-Format. Allerdings hat das mit „standardgerecht" nicht viel zu tun: Microsoft benutzt beim Konvertieren nach HTML so genannte XML-Dateninseln, die in HTML eingebettet werden. Hier legt der Hersteller Word- oder Excel-spezifische Informationen ab, die den verlustlosen Re-Import in Office sichern sollen.

Das eigentliche binäre Standardformat wurde nicht durch XML abgelöst, auch in der aktuellen Office-Version XP ist alles so geblieben wie gehabt.

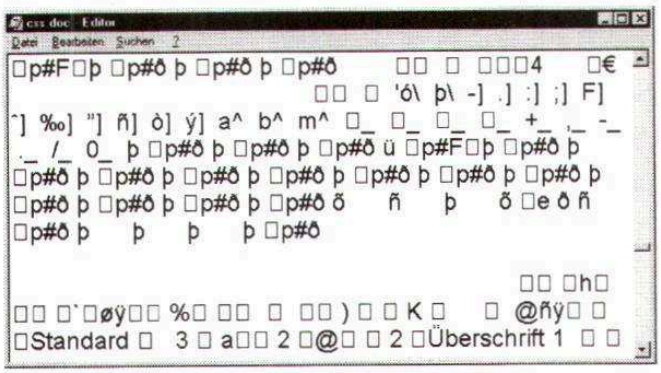
### XML ist internationalisierbar

XML lässt sich internationalisieren, es arbeitet mit beliebigen Zeichensätzen zusammen. In diesem Kurs bleiben wir jedoch bei ISO-8859-1, unserem westlichen Zeichensatz.

### XML ist rein textbasiert

Der XML-Standard ist rein textbasiert, benutzt kein Binärformat. Was daran so vorteilhaft ist?

Hast du dir mal den Spaß gemacht und eine Word- oder Excel-Datei in einem einfachen Texteditor geöffnet?




**Gruselig: Binärformat von Word im Texteditor**

Dann siehst du, was ich meine. Hier wird man mit einer verwirrenden Flut unverständlicher Zeichen konfrontiert, deren eigentliche Bedeutung wahrscheinlich nur Microsoft kennt.

Textdateien dagegen können ganz einfach ausgelesen und verstanden werden.

Zugegeben, einen Nachteil hat das Textformat: Der Speicherbedarf ist höher. Doch Speicherplatz wird heutzutage immer günstiger. Deshalb spielt das Argument wohl kaum mehr eine Rolle und die Vorteile des Textformats überwiegen den „Speicherplatz-Nachteil".

Außerdem gibt es inzwischen längst leistungsfähige Komprimierformate wie ZIP oder ARJ, mit denen man XML-Dateien auf „handliche Größe" zusammenschrumpfen kann.

StarOffice-Dokumente werden beim Speichern automatisch mit dem Zip-Algorithmus komprimiert.

## Ideal für langfristige Dateiablage

XML eignet sich genau wie SGML für die langfristige Datenablage. Ein weiterer schon angesprochener Vorteil: XML kann leicht in andere Formate umgewandelt, transformiert werden.

## XML ist modular, erweiterbar

Überall wird sie angewandt: Die Modulbauweise. Egal ob bei Ikea-Möbeln, Legosteinen, Personalcomputern oder XML.

Und das sind wir schon beim nächsten Punkt: Auch XML lässt sich erweitern, ausbauen. Schließlich heißt es Extensible Markup Language, erweiterbare Auszeichnungssprache.

XML ist eine so genannte Meta-Sprache, auf deren Grundlagen andere Sprachen erstellt werden können!

Du kannst sogar selber eigene Sprachen erstellen, die auf dem Regelwerk von XML beruhen! Doch soweit wollen wir es nicht treiben. Wir greifen auf vorhandene „Module" zurück.

## Sprachen auf XML-Basis

Viele dieser Module bzw. Sprachen auf XML-Basis gibt es schon bzw. sie befinden sich in der Entwicklung.

**MathML** (Mathematical Markup Language) ist eine Beschreibungssprache, gedacht speziell zum Erstellen mathematischer Formeln.

Du brauchst ein paar Hyperlinks in XML? Allein mit XML geht das leider nicht. Doch die Sprachen **XLink** und **XPointer** sorgen für die Verknüpfungen mit anderen Dokumenten. Die Verweismöglichkeiten gehen tlw. weit über die Konzepte von HTML-Hyperlinks hinaus, da beispielsweise auch auf ganze Seitenbereiche verwiesen werden kann.

Du möchtest etwas Pepp und Multimedia in deine tristen XML-Seiten bringen? Greife zum „SMIL-Modul". **SMIL** (Synchronized Multimedia Integration Language) wird für die Einbindung von Multimedia (TV, Video) eingesetzt werden.

Vektorgrafiken erzeugst du mit *SVG* (Scalable Vector Graphics). Das ist der künftige Standard zum Erzeugen von skalierbaren Vektorgrafiken.

Die für Wap-Handys gedachten Web-Seiten werden heute schon in **WML** erstellt. WML ist ebenfalls eine „XML-Praxisanwendung".

So viel Zukunftsmusik! Ist XML denn derzeit schon mehr als ein Wunschtraum und eine riesengroße Baustelle?

Leider: Baustelle ist immer noch der passende Begriff! Doch einige Abschnitte wurden schon fertig gestellt, wie ich dir auf der nächsten Seite zeigen kann.

# Einige Anwendungsbeispiele: XML in der Praxis

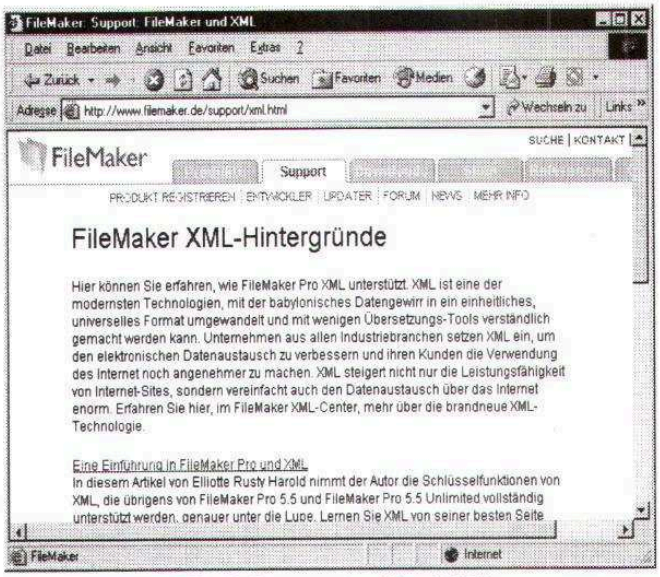
Das eben erwähnte modulare Konzept von XML führt hoffentlich dazu, dass sich XML als plattformunabhängiges, Programm-übergreifendes und universelles Dokumentformat durchsetzt.

Mutige und viel versprechende Praxis-Ansätze gibt es zuhauf. Egal ob Datenbank oder Office-Anwendung. XML ist längst nicht nur auf das World Wide Web beschränkt.

## FileMaker nutzt das XML-Format

So nutzt beispielsweise der Hersteller FileMaker bei seiner bekannten **Datenbank FileMaker Pro** schon heute den XML-Standard. FileMaker selbst kann sich dadurch auf die ureigenen Aufgaben einer Datenbank konzentrieren: Auf das Speichern, Suchen und Sortieren von Daten.

Die Darstellung der XML-Daten und deren Umwandlung in HTML wird dabei von jedem handelsüblichen Browser übernommen.




**Die Datenbank FileMaker setzt voll auf XML**

Auf den Webseiten von FileMaker ist dieser Vorteil sehr gut dokumentiert. Surfe zu www.filemaker.de (.com) und suche nach dem Stichwort `xml`.

## WAP ist eine XML-Anwendung

Egal ob Zauberwort oder Flop des ausgehenden 20. Jahrhunderts: Auch die auf Wap-Handys dargestellten Web-Seiten wurden schon damals in WML erstellt. Auch WML ist ein „waschechter XML-Dialekt".


**WAP ist eine Art „Mini-HTML" auf XML-Basis**

Dabei greift WAP auf einige aus HTML bekannte Tags zurück. Die Syntax wurde jedoch dem strengen XML-Standard unterworfen.

## HTML im neuen Gewand: XHTML

Apropos strenger Standard: Selbst unserem Klassiker HTML wurde vom World Wide Web Consortium als XHTML „neues Leben eingehaucht". XHTML ist nichts weiter als eine Neuformulierung des HTML-Standards unter Zuhilfenahme der strengen XML-Regeln. Dazu mehr Informationen auf Seite 53.

## Halbherzig: Microsoft Office

Wie schon erwähnt: Auch Microsoft-Office nutzt schon seit 1999 das XML-Format, wenn auch nicht in Reinkultur: MS Office arbeitet beim Konvertieren nach HTML mit so genannten XML-Dateninseln, die in HTML eingebettet werden. Hier legt Microsoft ganz einfach Word- oder Excel-spezifische Informationen ab, die den verlustlosen Re-Import in Office sichern sollen. Da das eigentliche binäre Standardformat von Office nicht durch XML abgelöst wurde, ist diese Lösung nur halbherzig. Warten wir auf die Version von Ende 2006!

## Windows Script-Host

Der Windows-Script-Host, die berühmt-berüchtigte Schnittstelle für Windows-Programmierer, arbeitet seit der Version 2.0 mit **WSF**, dem Windows Scripting Format. Auch dahinter verbirgt sich XML.

Schuld für den schlechten Ruf des Script-Hosts sind die vielen E-Mail-Würmer. Viele werden deshalb aktiv, weil der Script-Host installiert ist.

## Channel Definition Format

Überhaupt scheint Microsoft XML zu lieben, selbst die damals so „hochgepushten" unseligen Channels im alten Internet Explorer 4 werden durch das „XMLisierte" *CDF* beschrieben, das Channel Definition Format.

## XML bei StarOffice/OpenOffice

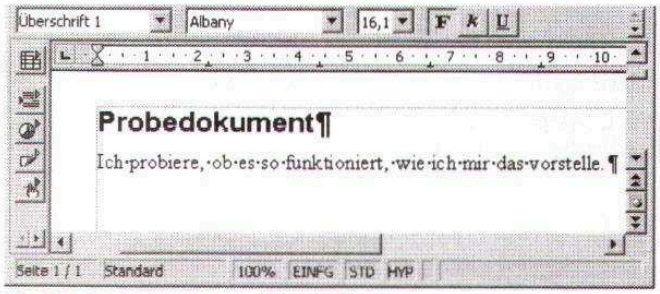
Zum Zeitpunkt des Schreibens gab es im Office-Sektor gerade eine kleine Revolution: Im Mai 2002 erschien die erste Office-Suite, die konsequent auf XML als Standardformat setzt, das schon erwähnte StarOffice von Sun.

StarOffice kostet in der kommerziellen Version ca. 79 Euro. Du kannst es über www.amazon.de bestellen. Die nur leicht abgespeckte kostenfreie Variante heißt dagegen *Open Office* und steht auf www.openoffice.org zu Download bereit.

Und tatsächlich speichert StarOffice alle Dokumente im XML-Format ab. Jeder kann die Informationen einsehen und verändern.

## Blick hinter die Kulissen

Du hast dieses Office auf deiner Festplatte installiert? Mach dir doch einmal den Spaß und untersuche eine Textdatei!




**Ein Probedokument in StarWriter**

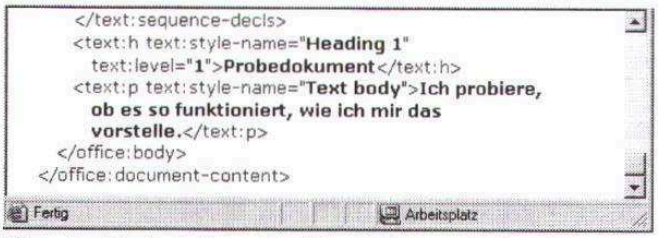
Das Probedokument ist schnell geschrieben. Es wird von StarWriter standardmäßig im Format *sxw* abgespeichert, zumindest scheinbar.

**In Wirklichkeit dient die jeweilige Endung nur zum Wiedererkennen.**

Alle Office-Dokumente werden nach dem ZIP-Verfahren komprimiert. Du kannst sie mit jedem Zip-fähigen Entpacker extrahieren, beispielsweise mit WinZip oder dem eingebauten Entzipper von Windows XP. Du musst lediglich die programmspezifische Endung durch *zip* ersetzen und gelangst an Inhalt der Datei heran! So einfach ist das!


**Aus *sxw* mache *zip*: Und schon entsteht ein Zip-Archiv**

Der Text-Inhalt des jeweiligen Dokuments findet sich in der Datei *content.xml*. Du kannst diese Datei mit einem einfachen Texteditor ansehen oder – nach Entfernen des Verweises auf die DTD – im Browser betrachten:




**Klar strukturiert: Ausschnitt aus der content.xml**

In *meta.xml* stehen u.a. die so genannten Meta-Informationen über das Dokument, die „Dateieigenschaften".

In *settings.xml* findest du Informationen z.B. zur Druckeinstellung u.v.a.m. Die Datei *styles.xml* beherbergt die für das Dokument definierten Formatvorlagen.

Die Datei *meta-inf/manifest.xml* beschreibt die Struktur der XML-Datei usw. usf.

Die so genannten Dokumenttyp-Definitionen (dazu mehr später) legt OpenOffice ebenfalls „öffentlich" auf deiner Festplatte ab.

Jeder kommt also an die Daten heran. Genau wie bei HTML gilt: Wer gute „Quelltext-Kenntnisse" hat, kann Office-Dokumente sogar komplett ohne „Office-Aufsatz" erstellen.

Genau wie zum Erstellen von HTML der reine Texteditor völlig genügt, kann nun jeder halbwegs fachkundige Computerfreund Office-Dateien „mit dem Editor" erstellen. Später sucht man sich sein Office-Produkt nicht mehr nach dem Dateiformat aus (Kann es Word 2000-Dateien lesen?) sondern nach dem Bedienkomfort. Im Hintergrund produzieren (hoffentlich) alle blütenreines XML.

XML-Schnittstellen sind offen und damit auch gut dokumentiert!

## Welche Nachteile hat XML?

Doch reden wir einmal über die Nachteile von XML. Ein offensichtliches Manko von XML ist der schon angesprochene größere Speicherplatzbedarf des Textformats im Vergleich zu binären Daten. Doch dieser Nachteil wird leicht ausgleichen: Da Speicherplatz heutzutage immer günstiger wird, spielt das Argument schon von daher kaum mehr eine Rolle.

Abgesehen davon existieren leistungsfähige Komprimierformate wie ZIP oder ARJ, die man zum Platz sparenden Sichern von XML-Textdateien verwenden kann.

Die Entwickler von StarOffice haben solch eine Zip-Routine in das Programm eingebaut.

## Eine Struktur, viele Layouts

Wenn Struktur und Layout getrennt sind, wie sollte man XML-Inhalte dann darstellen? Das hängt davon ab, für welchen Zweck du deine Daten benötigst und ob du sie überhaupt darstellen willst.

Bei speziellen XML-Anwendungen musst *du* gar nichts darstellen. Hier übernimmt die jeweilige Anwendung die Interpretation. Das WSF-Format funktioniert beispielsweise nur im Windows Scripting Host ab Version 2.0. Hier geht es um die Ausführung von Programmierbefehlen bei einem fest vorgegebenen Befehlsvorrat.

Auch WML, die Sprache für WAP-Seiten, greift paradoxerweise auf einen fest vorgegebenen Befehlsvorrat zurück. Hier werden die bekannten HTML-Tags zum Ausgeben der Daten verwendet. Und zwar im Handy.

Und die XML-Dateien von StarOffice bzw. OpenOffice machen eben (bisher) nur in den jeweiligen Programmen Sinn.

## Reines XML

Doch schauen wir mal auf „reines XML" in der Form, in der ich es dir beibringen will. Denken wir zuerst an die Darstellung im Web. Hier greifst du auf eine Layoutsprache wie CSS oder XSL zurück.

Während CSS noch ganz auf die Ausgabe im Web-Browser abgestimmt ist, wird das noch in Entwicklung befindliche XSL später sicher auch andere Aufgaben übernehmen können.

Forme die Daten dann so um, dass sie sich für den Druck in einem Hochglanz-Katalog eignen. Dafür eignet sich beispielsweise die Druckbeschreibungssprache Postscript oder der Adobe-Standard namens PDF (Portable Document Format).

Transformiere deine XML-Basis-Informationen gerne auch in ein Word-Dokument oder eine Kalkulationstabelle.

Bereite die Daten so auf, dass sie dem Kunden von einer Telefonsoftware vorgelesen werden.

Wie das alles gelöst werden wird, hängt vom jeweiligen Programm ab. Merke dir:

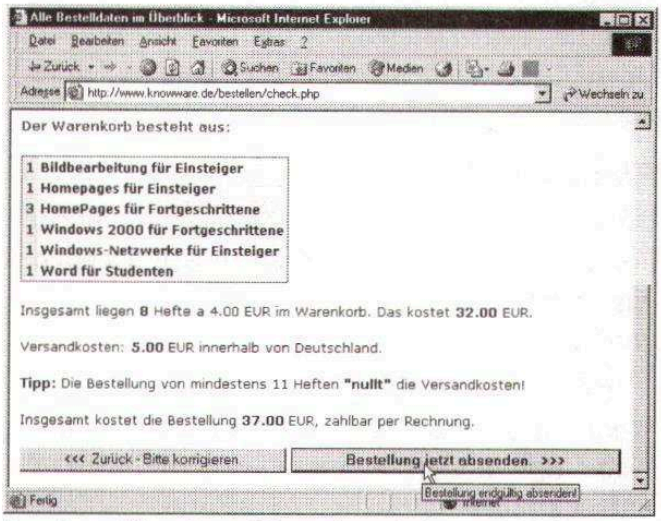
**Die einmal festgelegte Struktur bleibt, nur die Ausgabe lässt sich flexibel anpassen.**

Doch bleiben wir in der „Kurs"-Realität. Und die heißt derzeit immer noch Web-Browser und Internet.

## Das KnowWare-Bestellformular

Auch im Internet wird schon kräftig auf XML zurückgegriffen. Dabei arbeitet dieses Format jedoch in der Regel „hinter den Kulissen".

Kennst du noch das alte KnowWare-Bestellformular auf www.knowware.de? Es handelt sich um ein Warenkorb- und Bestellsystem mit interaktiver Auswertung. Du bekommst alle Daten vor dem Versenden noch einmal angezeigt. Selbst eine Bestätigungsmail für den Besteller wird vom Skript verschickt. Prima.




**Das KnowWare-Bestellformular arbeitete mit XML**

Doch wenn du zu www.knowware.de surfst, wirst du dich zu Recht fragen: Wo ist denn da XML? Richtig, die Seiten basieren auf HTML, welches tlw. statisch als feste Webseite, tlw. jedoch dynamisch mit PHP erzeugt wird.

Aber auch hinter diesem Formular steckte bis vor kurzem XML. Denn der alte Vertrieb von KnowWare arbeitete mit einem Datenbanksystem, welches auf XML-Daten angewiesen war. Der Vertrieb benötigte also ein XML-Dokument.

Deshalb musste ich das System so programmieren, dass die Bestelldaten intern in eine XML-Struktur umgewandelt und dann an den Vertrieb weitergeleitet wurden.

Inzwischen hat der Vertrieb jedoch auf ein anderes, XML-freies System umgestellt. Zumindest hier ist XML also im Rückzug.

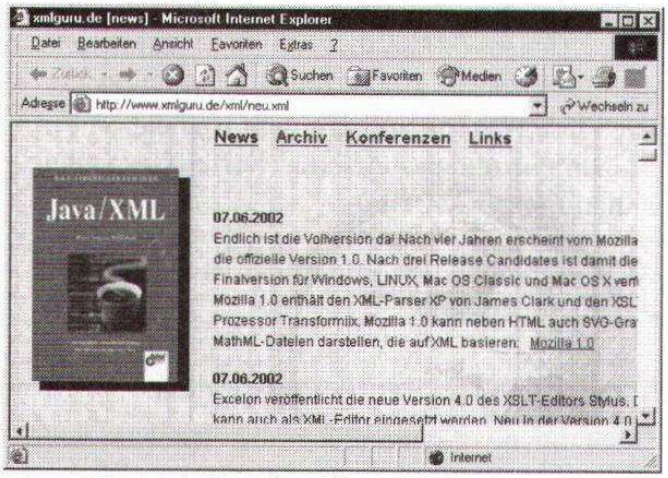
Solch ein Bestellformular ist durchaus kein Sonderfall mehr. Längst gibt es viele Webseiten, die sogar vollständig auf XML aufbauen. Doch warum merkst du davon in der Regel nichts?

## XML direkt im Browser ausgeben?

Macht es Sinn, überhaupt schon Webseiten vollständig in XML zu publizieren? So könnte mit Style Sheet-Sprachen wie CSS (und XSL) doch schon ein exaktes Layout erreicht werden?! Die Betonung liegt hierbei derzeit jedoch noch auf *könnte*.

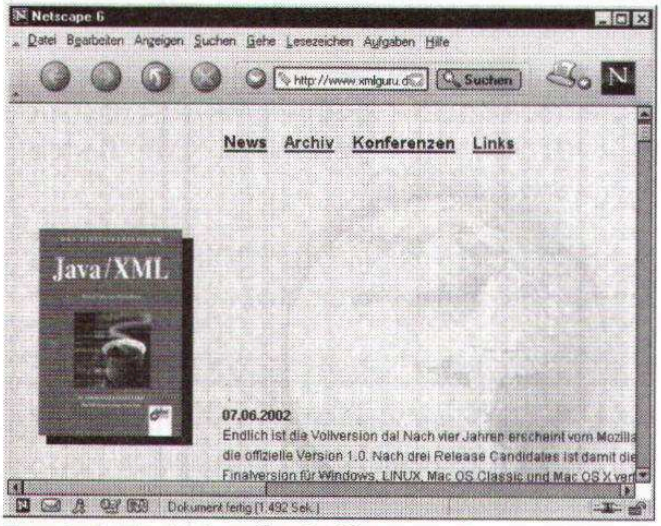
## Praxis-Beispiel XML-Guru

Schau dir als Praxis-Beispiel die Seiten meines Autorenkollegen Michael Seeboerger-Weichselbaum unter www.xmlguru.de an. Er bietet hier auch eine XML-Fassung seiner Seite an.




**Internet Explorer: Perfekter Sitz und Halt**

In der neusten Version 6 des Internet Explorers klappt die Anzeige perfekt. Die Seite ist praktisch nicht von einer normalen Webseite zu unterscheiden. Das Layout stimmt.




**Trauriges Ergebnis selbst im Netscape-Browser 6.2**

Doch im Netscape-Browser 6.2 rutschte das Layout völlig durcheinander, erst Version 7 und natürlich der Firefox zeigen alles korrekt an.

## Layoutabweichungen möglich

Zugegeben, mein experimentierfreudiger Kollege greift auf XSL als Formatiersprache zurück. Diese ist ein noch nicht so weit verbreiteter Standard wie CSS. Doch auch mit CSS habe ich in meinen eigenen Experimenten teils erhebliche Layoutabweichungen festgestellt.

## XML häufig „hinter den Kulissen"

Und trotzdem wird XML immer häufiger zur Datenspeicherung auch im Web-Publishing eingesetzt. Zum Beispiel bei den neuerdings immer beliebter werdenden **RSS-Newsfeeds** für „Content Syndication" (Übernahme von Inhalten).


**RSS-Feeds erkennt man oft am XML-Symbol**

Dahinter verbirgt sich eine Art moderner Newsticker: Abgelegt werden die Daten im XML-Format. Jeder kann darauf zugreifen. Damit hat man eine auch anderweitig verwendbare Struktur geschaffen, die in jedes beliebige Layout überführt werden kann. Und das ist der springende Punkt: jedes beliebige Layout.

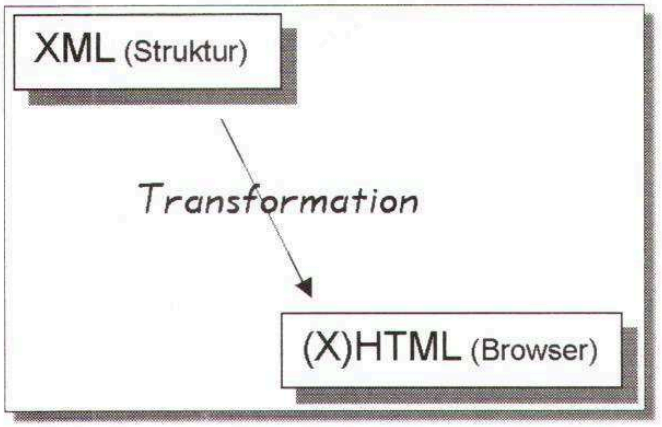
Die Ausgabe kann z.B. über ein Programm erfolgen, welches den Newsticker auf dem Windows-Desktop darstellt. Eins davon heißt Klipfolio. Wenn die Daten jedoch in eine Webseite integriert werden sollen, erfolgt das in der Regel weiterhin entweder als HTML bzw. XHTML.

**Die XML-Rohdaten werden im Format HTML bzw. XHTML dargestellt!**

Und hier wird es interessant: Wie bringt man dieses „Zauberwerk" zustande?

## Die Lösung: Transformieren!

Transformiere die Daten einfach von XML in HTML oder XHTML. Wie? Das gelingt mit Skriptsprachen. Diese können aus den XML-Basisdaten „klassisches" HTML erzeugen. Ich denke an PHP, Perl oder das mit ASP zusammenarbeitende VBS bzw. JavaScript.




**Auf Nummer sicher: XML nach HTML transformieren**

Die Grafik zeigt das Prinzip: Im Hintergrund liegen alle Daten im XML-Format vor. Der Anwender bekommt vom Webserver jedoch eine HTML-Seite zugeschickt. Der Browser muss dafür kein XML beherrschen.

Und tatsächlich gibt es auch bei Newsfeeds PHP-Lösungen, die aus den XML-Rohdaten ganz stinknormale HTML-Seiten generieren.

Das Konvertieren von XML nach HTML durch eine Skriptsprache ist jedoch für uns eine recht aufwändige Geschichte. Du brauchst Programmierkenntnisse! Dieses Thema werden wir in diesem Kurs daher nicht weiter behandeln.

Nur nebenbei: KnowWare hält Literatur zu einigen Programmiersprachen wie Perl, PHP oder JavaScript bereit, damit du wenigstens ein Grundwissen bekommst. Von mir gibt es den schon in der 2. Ausgabe erschienenen Titel „PHP für Einsteiger", der (hoffentlich) auch solides Skripting-Grundwissen vermittelt.

## Transformieren per XSL

Zurück zu XML. Auch hier gibt es eine „Layout- und Transformationssprache", die direkt in XML geschrieben wurde und für XML gedacht ist. Ich rede von XSL, eXtended Stylesheet Language. Es handelt sich um eine spezielle „Transformations-Sprache", die XML in andere Formate umwandeln kann.

Wenn wir uns ab Seite 63 mit dieser Sprache beschäftigen wirst du feststellen, dass man auch hier von einer Art „Programmierung" reden kann. Der Nachteil von XML für das Web? XSL wird bisher noch ungenügend unterstützt, siehe Abbildungen auf der Vorseite. Doch wir bleiben optimistisch!

# Zusammenfassung: Wichtige Eigenschaften von XML

Halten wir fest, dass *XML **allein*** für das Web-Publishing derzeit noch ungeeignet ist. Völlig ungeeignet sogar. Außerdem merken wir uns, dass XML hervorragend für Daten geeignet ist, die in eine Struktur gebracht werden müssen.

Klingt bisher nicht gerade aufregend, und trotzdem gibt es ein paar herausragende Eigenschaften, die die Fähigkeiten von HTML weit in den Schatten stellen.

## Definieren der eigenen Tags

Die erste Eigenschaft klingt wie eine kleine Sensation: Du kannst deine eigenen Tags definieren. Richtig gelesen, egal ob du

```
<Ueberschrift1>...</Ueberschrift1>,
<absatz>...</absatz> bzw.
<Hundekuchen>...</Hundekuchen>
```

schreibst, alles das ist zulässig. Was du mit einem jeweiligen Tag-Paar jedoch meinst, ist dir überlassen. Ganz im Gegensatz zu HTML (alle Tags sind vorgegeben, der Tag-Vorrat ist begrenzt) schreibt dir in XML keiner vor, welche Tags du verwenden sollst.

> Vor dem Schreiben eines XML-Dokuments solltest du dir also die Tags und deren Bedeutung gut überlegen.

Und da beginnt der Stress: Diese Vorarbeiten sind ähnlich aufwändig wie das Planen einer Datenbank!

Einmal geschaffen, musst du dich außerdem um das Aussehen dieser Tags kümmern, denn das liegt schließlich ebenfalls noch nicht fest.

## Eigene Attribute festlegen

Als ob das nicht reicht, kannst du zu diesen Tags sogar eigene Attribute definieren. Hier zwei Beispiele:

```
<absatz typ="kasten">...
<Hundekuchen lecker="nein">...
```

Auch hier ist dein Planungstalent gefragt. Welche Attribute benötigst du? Welche Werte sind erlaubt? Kannst du die gewünschten Eigenschaften vielleicht auch allein über Tags ausdrücken? Da fließt eine Menge Gehirnschmalz, bis man sich darüber im Klaren ist.

## Beschreibung per Regelsatz (DTD)

Zu allem Überfluss kannst du diese Tags und Attribute auch noch genau beschreiben. (Du kannst, musst aber nicht!)

Du kannst also vorher eine Art Anweisung definieren, mit der du alle gewünschten Tags, ihre Namen, Reihenfolge, Verschachtelung usw. definierst.

Du legst fest, welche Attribute zulässig sind und welche Werte eingesetzt werden dürfen. Du beschreibst, ob man ein Tag bzw. Attribut verwenden muss bzw. nur verwenden kann.

Dieses komplizierte Regelwerk gibt es in zwei „Geschmacksrichtungen". Die alt-eingeführte heißt DTD (Dokumenttyp-Definition). Die erste DTD erstellen wir gemeinsam ab Seite 26. Die neuere, weitaus kompliziertere Fassung nennt sich Schema. Die ersten „schematischen Schritte" beschreiten wir gemeinsam auf Seite 69.

> Nur XML-Dokumente mit DTD/Schema werden als gültig (valid) bezeichnet. Wenn du die DTD oder das Schema weglässt, ist das Dokument nur noch wohlgeformt (well-formed).

Kommen wir nun zur wichtigsten Eigenschaft.

## Trennung von Struktur und Layout

In XML sind Struktur und Layout streng getrennt. Dieser Punkt ist so wichtig, dass ich ihn nicht oft genug erwähnen kann.

Wenn du deine Daten in XML erfasst, weißt du damit noch lange nicht, wie diese dargestellt werden. Das Prinzip ist, wie schon erwähnt, gut mit Datenbanken vergleichbar.

Ob du deine Adressliste als Tabelle oder im Adresskartenformat für Aufkleber ausdrucken möchtest, bleibt dir überlassen. Die Roh-Daten sind immer da, die Ausgabe-Form kann stets variiert werden.

# Übungsteil A: Allgemeine Fragen zur Einführung

Du kennst inzwischen:

- SGML und HTML, die Vorläufersprachen zu XML
- Vor- und Nachteile von XML
- Haupteigenschaften von XML
- Anwendungen, die auf XML beruhen

## Fragen rund um XML

Bitte beantworte folgende Fragen „rund um XML". Eine oder auch mehrere Antworten sind richtig. Die Lösungen findest du auf der Serviceseite zum Heft, unter www.jchanke.de/xml. Hier gibt es einen Fragebogen mit interaktiver Auswertung.

| **1. Welches sind die Haupteigenschaften von SGML und XML?** | **2. Auf welche Sprachen greift HTML zurück?** |
|---|---|
| [a] sichert Formate wie fett, kursiv usw. | [a] ASCII |
| [b] sichert die logische Struktur von Dokumenten | [b] SGML |
| [c] Dokumentsicherung im universalen Word-Format | [c] XML |
| [d] plattformunabhängige Dokumentsicherung | [d] CDF |
| **3. Wann wurde SGML zum ISO-Standard?** | **4. Wer hat HTML entwickelt?** |
| [a] 1971 | [a] Microsoft |
| [b] 1986 | [b] Tim Berners-Lee |
| [c] 1989/90 | [c] Adobe und Sun |
| [d] 1996 | [d] FileMaker |
| **5. Wofür steht die Abkürzung HTML?** | **6. Welche Eigenschaften charakterisieren die Sprache XML?** |
| [a] Hyper Tool Mouseloader | [a] speichert nur die Daten, nur die Struktur |
| [b] Hypertext Multi Language | [b] nicht vorrangig für den Einsatz im Web |
| [c] Hypertext Markup Language | [c] rein textbasiertes Format |
| [d] Hyperlink Extended Markup Language | [c] lizenzfreier Standard |
| **7. Welche der hier genannten Formate haben nichts mit XML zu tun?** | **8. Wer ist für die Weiterentwicklung von Sprachen wie XML zuständig?** |
| [a] SMIL | [a] WAP (Wireless Application Protocol) |
| [b] WAP | [b] USB (Universal Serial Bus) |
| [c] XHTML | [c] W3C (World Wide Web Consortium) |
| [d] SQL | [d] Adobe |

# Lektion 1: Hallo XML! Erste praktische Versuche mit XML

Schluss mit der Theorie! Beginnen wir mit einigen ersten praktischen Übungen: In der ersten Praxis-Lektion lernst du folgendes:

- Erstellen eines einfachen XML-Dokuments mit dem Editor
- Erzeugen von Prolog, Tags und Wurzelelement
- Besonderheiten bei der Anzeige in unterschiedlichen Browsern
- Verwendung von Sonderzeichen und Entitäten

Tipp: Erstelle unter deiner Festplatte C:\ einen Ordner namens `xmlkurs`. Lege hier und in noch zu erstellenden Unterordnern alle Dokumente ab. Alles klar?

Zum Entwickeln von XML brauchst du nur einen Texteditor. Unter Windows bietet sich der Editor an. Wähle einfach START/AUSFÜHREN und tippe `notepad`. Drücke auf [ENTER]. Der Editor erscheint. Schreibe folgenden Code ab! Das ist gleichzeitig der Grundaufbau eines XML-Dokuments.

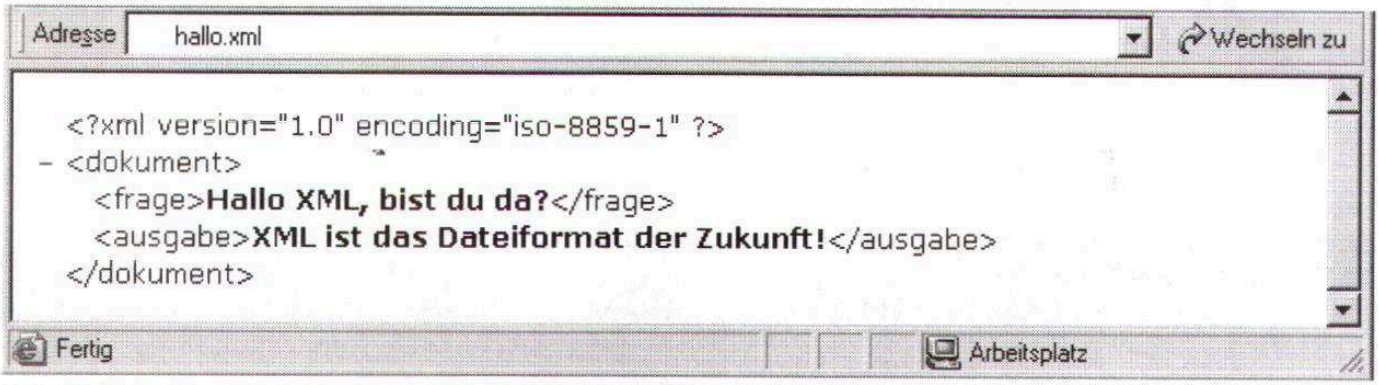
```
<?xml version="1.0" encoding="iso-8859-1"?>
<dokument>
    <frage>Hallo XML, bist du da?</frage>
    <ausgabe>XML ist das Dateiformat der Zukunft!</ausgabe>
</dokument>
```

Speichere das Dokument mit der Nachsilbe .xml, nenne es im Beispiel *hallo.xml*. Speichere es für unseren Kurs im Ordner *xmlkurs* ab.

Du möchtest eine bestimmte Nachsilbe im Editor „erzwingen"? Manchmal gibt es Probleme! Setze den Dateinamen deshalb gleich in Gänsefüßchen. Schreibe also "hallo.xml".

## Das Ergebnis im Internet Explorer ausgeben

Rufe das Dokument nun im Internet Explorer auf. So sieht es aus:




**Hallo XML! Der Internet Explorer gibt XML-Dokumente in einer Baumstruktur aus**

Wir nutzen den Internet Explorer deshalb, weil dieser Browser einige wichtige Eigenschaften besitzt, die manch andere Browser nicht bieten können. Du kannst aber auch den Firefox verwenden. Solange du keine Layoutanweisungen einbringst, wird bei diesen beiden Browsern jedes XML-Dokument in seiner Baumstruktur angezeigt.

Schauen wir uns nun genau an, was sich hinter den einzelnen Elementen verbirgt!

## Der Prolog

Die erste Zeile leitet ein XML-Dokument ein, das ist der so genannte Prolog. Der Prolog besteht in unserem Beispiel lediglich aus dem *<?xm...?>*-Tag:

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

Die Fragezeichen am Anfang und Ende sind typisch für dieses Tag. Mit dem Attribut version="1.0" definierst du die Version von XML, derzeit ist es die Version *1.0*.

Die Angabe `encoding="iso-8859-1"` wiederum bedeutet nichts weiter, als dass du unseren im europäischen Raum verbreiteten Zeichensatz *iso-8859-1* (Latin 1) verwendest. Dieser kann auch Umlaute und Sonderzeichen darstellen.

Ganz wichtig: Bei *diesem* Tag und seinen Attribut-Namen (nicht den Werten) ist **Kleinschreibung** Pflicht! Schreibe also nicht *<?XML ...* sondern *<?xml ...*

## Das Wurzelelement

Danach folgt das Wurzelelement. Dieses heißt im Beispiel *dokument*. Das Wurzelement ist Pflicht und leitet das Dokument ein.

**Das Wurzelelement ist eine Art Klammer. Es umschließt den eigentlichen Inhalt. Du kannst es mit *<body></body>* bei HTML-Dokumenten vergleichen.**

Die nächsten beiden Zeilen enthalten den eigentlichen Inhalt. Dafür sorgen die selbst definierten Tags *<frage></frage>* und *<ausgabe></ausgabe>*. Zum Schluss wird das Wurzelelement mit dem Tag *</dokument>* geschlossen. Bei „großen" Projekten denkst du dir natürlich wesentlich mehr und möglicherweise auch sinnvollere Tags aus.

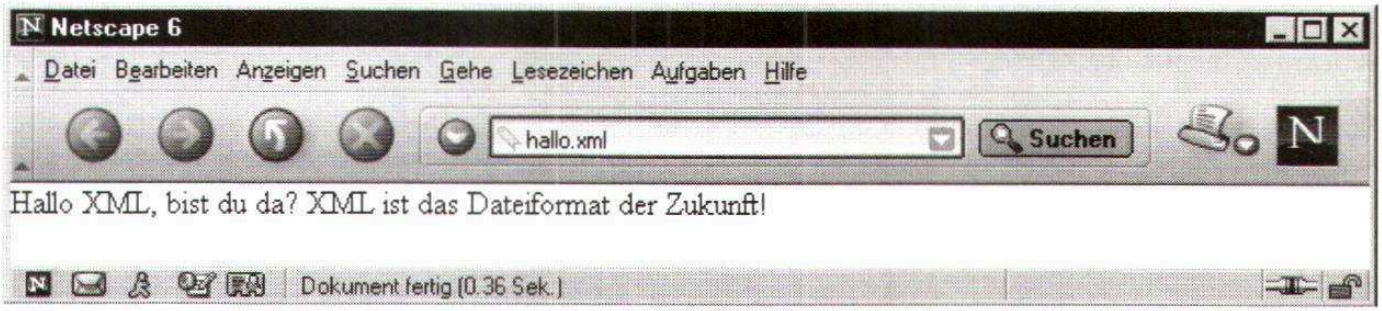
Das Schöne: Ganz im Gegensatz zu HTML sind keine weiteren Tags nötig. Das *<?xml ...>*-Tag vom Anfang muss nicht abgeschaltet werden! Lediglich das Wurzelelement *<dokument></dokument>* bildet die „Klammer".

Halten wir fest: Das Basis-XML-Dokument besitzt in der Grundausstattung einen Prolog, ein „Wurzelelement" und etliche selbst definierte Tags.

Verzichte bei den Tagnamen auf Umlaute, Sonderzeichen, Leerzeichen und Doppelpunkt. Beginne nicht mit einer Zahl. Groß- und Kleinschreibung werden unterschieden! So sind *<Ausgabe>* und *<ausgabe>* verschiedene Tags!

## Ansicht im Netscape-Browser

Wenn du möchtest, kannst du dir das Beispiel auch in einem anderen Browser ansehen. Rufe beispielsweise Netscape 6.x, Mozilla oder Opera auf. Gehe in den Ordner *XML-Kurs*. Ziehe die Datei *hallo.xml* bei gedrückter linker Maustaste in das Browserfenster hinein.




**Browser wie Netscape 6.x zeigen lediglich den reinen Text an**

Du stellst fest, dass etliche andere Browser lediglich den reinen Text anzeigen, hintereinander weg.

# Verwendung von Sonderzeichen und Entitäten

Besprechen wir an dieser Stelle schnell noch, welche Zeichen du verwenden darfst und welche nicht. Denn hier lauern ein paar Fallstricke.

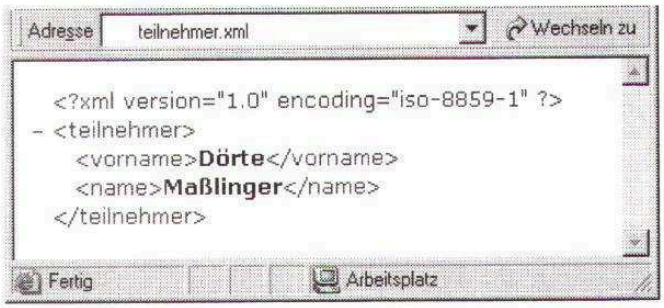
## Umlaute verwenden

Die gute Nachricht hatte ich schon verbreitet: Arbeite *in deinen XML*-Dateien (nicht bei den Tags!) ganz vergnügt auch mit Umlauten oder typischen Sonderzeichen wie dem ß. Ein XML-Dokument wie:

```
<?xml version="1.0" encoding=
"iso-8859-1"?>
<teilnehmer>
    <vorname>Dörte</vorname>
    <name>Maßlinger</name>
</teilnehmer>
```

ist also vollkommen korrekt. Schließlich haben wir hinter dem Attribut *encoding* den Wert *iso-8859-1* angegeben.

Das iso-8859-1 steht für Latin 1, für unseren westeuropäischen Zeichensatz, der auch die Umlaute oder das scharfe „s" zulässt. Damit decken wir alle gängigen Sonderzeichen z.B. des deutschen, französischen, spanischen und italienischen Sprachraums ab.




**Latin 1 stellt Umlaute und Sonderzeichen dar**

Wir bleiben bei diesem Zeichensatz, da wir uns so alle Probleme vom Hals schaffen. Die wichtigsten ISO-Standards habe ich dir in der nebenstehenden Tabelle dargestellt.

Schreibe das Dokument ruhig zur Übung ab. Speichere es unter dem Namen *teilnehmer.xml*. (Der Zeilenumbruch hinter *encoding* ist nur den engen Spalten geschuldet.)

## Die wichtigsten Ländercodes

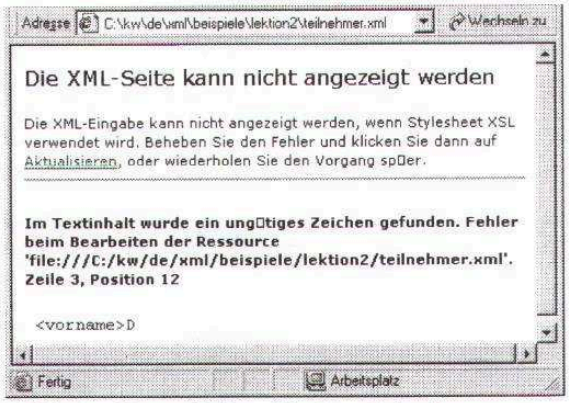
Hier zeige ich dir die wichtigsten Ländercodes. Ob du *ISO* groß oder als *iso* klein schreiben magst, bleibt dir überlassen.

| ISO-Standard | Ländercode |
|---|---|
| UTF-8 (Unicode) | universeller Zeichensatz, weltweit |
| **ISO-8859-1 (Latin-1)** | Westeuropa, Lateinamerika |
| ISO-8859-2 (Latin-2) | Zentral- und Osteuropa |
| ISO-8859-3 (Latin-3) | Südosteuropa |
| ISO-8859-4 (Latin-4) | Skandinavien, Baltikum |
| ISO-8859-5 | Kyrillisch |
| ISO-8859-6 | Arabisch |
| ISO-8859-7 | Griechisch |
| ISO-8859-8 | Hebräisch |
| ISO-8859-9 | Türkisch |
| ISO-8859-10 | Lappländisch, Nordisch, Eskimosprachen |
| EUC-JP oder Shift_JIS | Japanisch |

## Zeichensatz falsch? Fehlanzeige!

Mach dir doch einmal den Jux und verändere den Zeichensatz. Setze statt *iso-8859-1* einfach *iso-8859-3* (Südosteuropa) oder *iso-8859-5* (Kyrillisch) ein. Und schon werden die Zeichen durch andere ersetzt. Kunststück, die Umlaute fehlen in den entsprechenden Zeichensätzen!

Wenn du *utf-8* einsetzt (Unicode), bekommst du sogar eine Fehlermeldung. Hier sind die Umlaute zwar enthalten, doch dieser noch nicht sehr weit verbreitete „Universalzeichensatz" verlangt eine strenge Umlautmaskierung (Entitäten).




**Fehlermeldung: Beim „ö" stoppt der Browser**

Auch das Weglassen des *encoding*-Attributs führt zu einer Fehlermeldung. Denn dann geht der Browser generell von Unicode (utf-8) aus.

## Verbotene Zeichen umschreiben

Natürlich könntest du bei einem Fehler Dörte mit oe und Maßlinger mit „Doppel-s“ schreiben. Dann klappt alles. Doch dank unseres wunderbaren „Latin-1-Zeichensatzes“ ist das unnötig.

Daneben gibt es jedoch einige Zeichen, die du auf gar keinen Fall verwenden darfst:

- <
- >
- &

Die spitzen Klammern umkleiden beispielsweise die Tags und würden zu Fehlinterpretationen führen. Auch das kaufmännische *Und* hat eine Sonderbedeutung. Diese drei Zeichen darfst du auf gar keinen Fall „im Reinformat“ notieren.

Ebenfalls problematisch sind das doppelte " und das einfache ' Anführungszeichen.

Alle diese Zeichen solltest du umschreiben, man spricht auch von „escapen“ bzw. „maskieren“.

**Das Umschreiben gelingt mit den so genannten Entitäten (Entities).**

Entitäten sind eine Art „Platzhalter“. Eine Entität beginnt hier mit & und endet immer mit einem Semikolon ;. Die Umschreibungen greifen in diesem Fall auf die Zeichencodes der jeweiligen ISO-Tabellen zurück.

Man schreibt `&#NR;`, wobei für NR der entsprechende Code eingesetzt wird. So besitzt das Zeichen < den Code 60 und wird dementsprechend mit `&#60;` verschlüsselt. Du kannst aber auch den „freundlicheren Namen“ `<` verwenden; `lt` steht für **l**ower-**t**han, kleiner als.

## Freundlichere Namen für Entitäten

Folgende fünf „freundliche Namen“ – auch als *Namensentitäten* bezeichnet – sind erlaubt:

| Zeichen | Entität |
|---|---|
| < | &lt; |
| > | &gt; |
| & | &amp; |
| " | &quot; |
| ' (Apostroph) | &apos; |

Das war's dann auch schon mit der Freundlichkeit. Die aus HTML bekannten anderen Namensentitäten wie *ä* bzw. *ö* usw. sind leider nicht zulässig. Nimm die entsprechenden „ISO-Codes“ aus der Code-Tabelle, also für ö z.B. `&#246;` oder für ß `&#223;`. Das klappt selbst bei Unicode!

## Codes für wichtige Sonderzeichen

Die nächste Tabelle enthält einige wichtige Sonderzeichen und die entsprechende Entität.

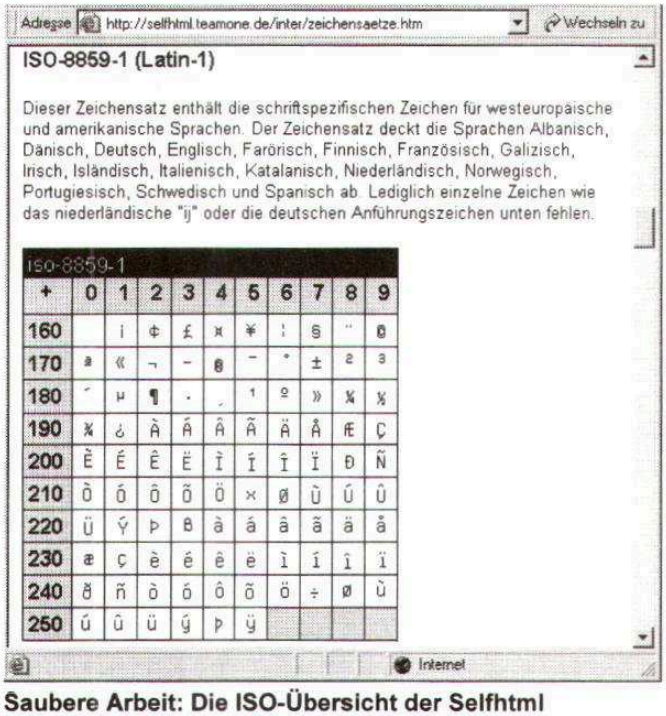
| Zeichen | Kodierung |
|---|---|
| » | &#187; |
| « | &#171; |
| © | &#169; |
| ® | &#174; |
| £ | &#163; |

**Der Euro läuft als Zeichen `&#8364;` außer Konkurrenz, meist klappt die Anzeige.**

## Gesamtübersicht zu Zeichensätzen

Eine gute Einführung in das Thema Zeichensätze bietet die Selfhtml unter:

selfhtml.teamone.de/inter/zeichensaetze.htm




**Saubere Arbeit: Die ISO-Übersicht der Selfhtml**

An dieser Stelle habe ich lediglich gezeigt, wie man mit Entitäten Sonderzeichen umschreibt. Du kannst Entities aber auch als Platzhalter bzw. Abkürzung (für Programmierer: als eine Art „Variable“) für ganze Zeichenfolgen einsetzen. Mehr dazu erfährst du ab Seite 43!

# Lektion 2: Baumstruktur von XML anhand einer Titelliste

Machen wir nun mit einem durchaus sinnvollen Projekt weiter. Wir wollen einige KnowWare-Titel in einer Titelliste ausgeben. In diesem Beispiel lernst du folgendes:

- Daten sinnvoll ordnen am Beispiel einer einfachen Titelliste
- Einbau eines zusätzlichen Knotens in XML
- Erkennen der Baumstruktur von XML
- Erstellen von wohlgeformten (well-formed) Strukturen

## Zuerst wird die Titelliste geplant, strukturiert

XML eignet sich besonders für strukturierte Daten. Und was ist eine Titelliste anderes als eine wunderbare Struktur? Für unsere KnowWare-Titelliste möchte ich pro Heft (`heft`) folgendes festhalten:

1. Titel (`titel`)
2. Autor (`autor`)
3. Verlag (`verlag`)
4. Beschreibung, *falls ich eine weiß* (`beschreibung`)
5. Preis (`preis`)

**Aufgabe:** Mache dich vorab mit den strengen Regeln von XML vertraut, siehe Seite 53f.!

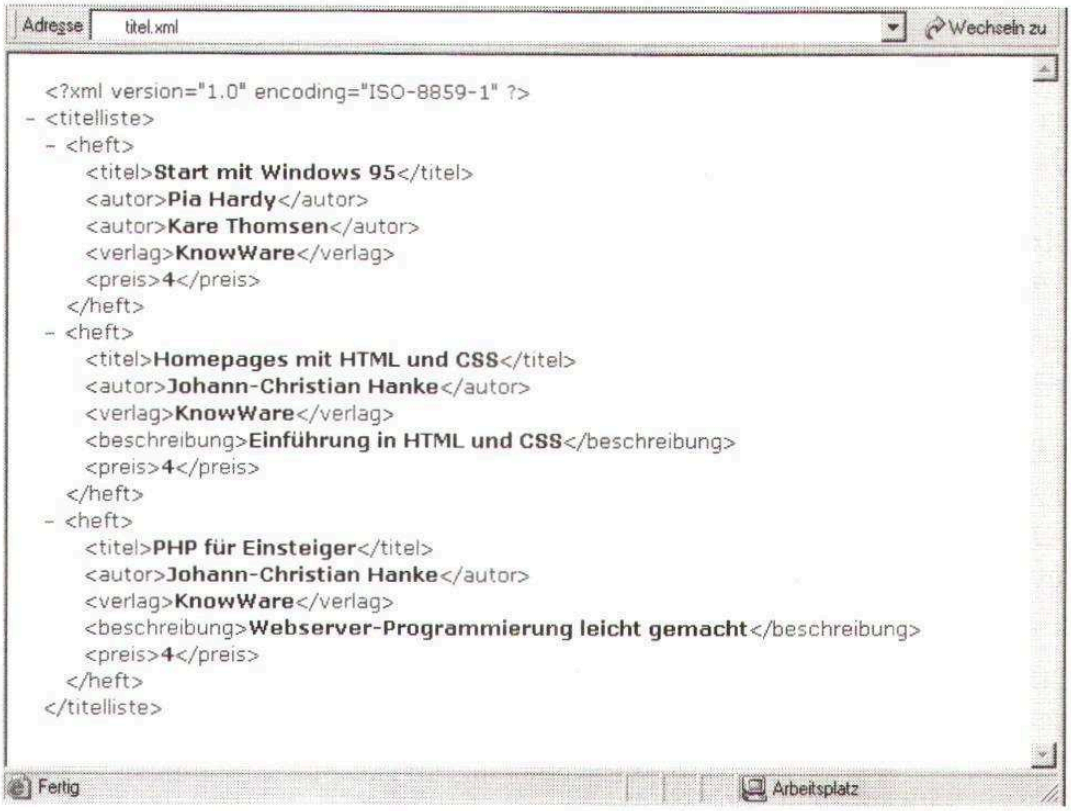
Schreibe einfach folgendes XML-Dokument ab. Ich habe es *titel.xml* genannt und lege es im Projektordner *xmlkurs* ab. Die Einrückungen sollen den Code lediglich etwas übersichtlicher gestalten:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<titelliste>
    <heft>
       <titel>Start mit Windows 95</titel>
       <autor>Pia Hardy</autor>
       <autor>Kare Thomsen</autor>
       <verlag>KnowWare</verlag>
       <preis>4</preis>
    </heft>
    <heft>
       <titel>Homepages mit HTML und CSS</titel>
       <autor>Johann-Christian Hanke</autor>
       <verlag>KnowWare</verlag>
       <beschreibung>Einführung in HTML und CSS</beschreibung>
       <preis>4</preis>
    </heft>
    <heft>
       <titel>PHP für Einsteiger</titel>
       <autor>Johann-Christian Hanke</autor>
       <verlag>KnowWare</verlag>
       <beschreibung>Webserver-Programmierung leicht gemacht</beschreibung>
       <preis>4</preis>
    </heft>
</titelliste>
```

Keine Lust zum Abschreiben? Die Beispieldokumente findest du – wie schon erwähnt – in nach Lektionen benannten Unterordnern. Download aller Beispiele via: www.jchanke.de/xml.

## Neben dem Wurzelelement gibt es weitere Knoten

Rufe das Dokument im Internet Explorer auf. Diese Titelliste enthält den obligatorischen Prolog und natürlich das Wurzelelement. Diese „große Klammer" heißt hierbei *<titelliste></titelliste>*.




**Gut zu erkennen: Durch mehrere Knoten entsteht eine Baumstruktur**

Interessant ist, dass es einen weiteren Ast gibt, also einen weiteren Knoten. Probiere es aus:

1. Klicke vor *<titelliste>*, also vor das Wurzelelement. Damit klappst du *alles* auf bzw. wieder zu.
2. Klicke vor das Tag *<heft>*. Damit klappst du den *jeweiligen Titel* auf bzw. wieder zu.

Es ist ähnlich wie im Windows-Explorer: Das Minuszeichen bedeutet: aufgeklappt. Ein Plus-Zeichen steht für einen noch nicht „entfalteten" Ast. Wie du merkst, bietet XML nichts weiter als die Struktur, um Daten anzuzeigen. Prolog, Wurzelelement und die einzelnen Tags sind das nötige „Beiwerk".

## Der Browser als Parser: Was ist denn Parsen?

Der Browser zeigt dir diese Struktur an. Er liest dein XML-Dokument aus. In der Fachsprache redet man davon, dass der Browser das Dokument „parst".

In der Praxis gibt es natürlich viele weitere Parser. Bei StarOffice parst die Textverarbeitung bzw. Tabellenkalkulation das XML-Dokument und zeigt es zum Bearbeiten am Bildschirm an. Beim Script Host (der Programmierschnittstelle von Windows) parst Windows dein Skript. Bei Newsfeeds wird die XML-Datei z.B. vom PHP-Skript geparst und als HTML dargestellt.

Zugegeben, in diesem Kurs nehmen wir weiterhin den (möglichst neuesten) Internet Explorer, weil er uns zur Verfügung steht und weil wir die Daten irgendwie ausgeben wollen. Wir sehen den Browser nicht als „Frontend für tolles Webdesign", denn darum geht es schließlich nicht. Er ist nicht mehr als ein „Anzeigewerkzeug". Vergiss nie, dass unsere Experimente vorrangig dazu dienen, die Struktur von XML zu lernen. (Auch wenn wir später mit CSS und XSL gestalten werden.)

### Titelliste als wohlgeformtes Dokument

Wichtig: Bis jetzt haben wir es mit einem „wohlgeformten Dokument" (well-formed) zu tun. Es besitzt keine Dokumenttyp-Definition, also keinen Regelsatz für die Tags.

# Lektion 3: Eine DTD für die Titelliste erstellen

Erstelle nun einen Regelsatz für die Titelliste, die so genannte Dokumenttyp-Definition. In dieser Lektion lernst du folgendes:

- Planen einer Dokumenttyp-Definition (DTD)
- Erstellen von gültigen (valid) Strukturen
- Schreiben einer DTD
- Begriff PCDATA
- Schlüsselwörter und Indikatoren der DTD
- Verweis auf DTD setzen

Widmen wir uns zuerst dem Thema Dokumenttyp-Definition, kurz DTD.

## Dokumenttyp-Definition

Wenn du meine Einleitung gelesen hast, weißt du Bescheid: Ein XML-Dokument kann mit einer Dokumenttyp-Definition versehen werden (muss aber nicht).

Zur Erinnerung: Die DTD ist ein Regelwerk, welches die gewünschten Tags, ihre Namen, Reihenfolge, Verschachtelung usw. definiert. Du legst fest, welche Attribute zulässig sind und welche Werte eingesetzt werden dürfen. Du beschreibst, ob man ein Tag bzw. Attribut verwenden muss bzw. nur verwenden kann.

Sinn und Zweck der Geschichte ist es natürlich, das Regelwerk *vorher* festzulegen. In dieser Übung wollen wir die DTD jedoch nachträglich erstellen. Denn das ist einfacher, da die XML-Struktur schon sichtbar ist und du dir etwas darunter vorstellen kannst.

## Gültig oder wohlgeformt?

Ein Dokument ohne DTD ist *well-formed* (wohlgeformt). Wenn es eine DTD besitzt und korrekt geschrieben wurde wird es *valid* (gültig).

Im folgenden Beispiel wandeln wir unsere wohlgeformte Datei *titel.xml* also in ein gültiges XML-Dokument um!

## Planen der DTD

Was müssen wir für unsere DTD wissen?

- Das Wurzelelement heißt *titelliste*.
- In diesem Wurzelelement stecken *n Elemente* (mehrere Elemente) namens *heft*.
- Jedes Heft besitzt genau einen Titel (Element *titel*).
- Jedes Heft besitzt einen oder mehrere Autoren (*autor*).
- Jedes Heft hat einen Verlag (*verlag*), dass es überall der gleiche ist, darüber sehen wir wohlwollend hinweg. ;-)
- Die Beschreibung gehört scheinbar nicht zu jedem Titel, das Tag *beschreibung* ist also optional.
- Nicht zu vergessen der Preis, der durch das Tag *preis* festgehalten wird.

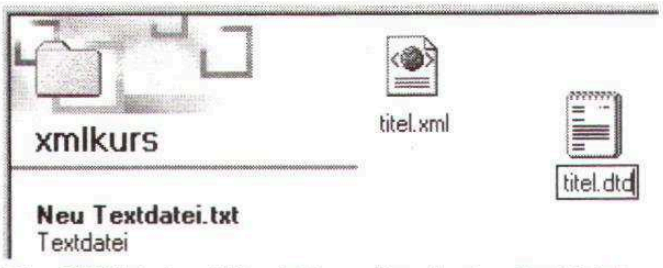
Das sind die Grundvoraussetzungen!

## DTD erstellen

Wir wollen uns im Beispiel gleich daran gewöhnen, mit externen DTDs zu arbeiten. Das erhöht die Übersicht, da XML-Dokument und DTD getrennt sind.

**Eine externe DTD ist nichts weiter als ein Textdokument mit der Endung *dtd*.**

Erstelle ein Textdokument namens *titel.dtd*.




**Eine DTD ist ebenfalls nichts weiter als eine Textdatei**

Tipp: Du kannst Textdateien auch auf die „objektorientierte" Art und Weise erstellen. Gehe in den gewünschten Ordner. Klicke mit rechts und wähle *Neu/Textdatei*. Überschreibe den Platzhalter-Namen mit *titel.dtd*.

## Quelltext der Datei titel.dtd

Und hier zeige ich dir den kompletten Quelltext der Datei *titel.dtd*. Schreibe mein Beispiel ab und vergiss nicht, die Daten zu speichern!

```
<!ELEMENT titelliste (heft+)>
<!ELEMENT heft (titel,autor+,verlag,beschreibung?,preis+)>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT verlag (#PCDATA)>
<!ELEMENT beschreibung (#PCDATA)>
<!ELEMENT preis (#PCDATA)>
```

## Die Bedeutung der einzelnen Zeilen

Huch, das sieht ja ziemlich kompliziert aus! Das liegt daran, dass DTDs in der Sprache SGML geschrieben werden. Und diese besitzt leider eine recht komplizierte Syntax. Schauen wir uns deshalb ganz in Ruhe an, was sich hinter den einzelnen Zeilen verbirgt, und zwar Schritt für Schritt!

1. In der ersten Zeile wird das Tag *titelliste* definiert. Das ist unser Wurzelelement. Jede Element-Definition beginnt mit <!ELEMENT ..., und zwar in Großschreibung. Da das Element *titelliste* nur einmal vorkommt, setzen wir keine weiteren besonderen „Auszeichnungsmerkmale". Doch was verbirgt sich hinter der Zeichenfolge *heft*+, die wir in runden Klammern notiert haben? Ganz einfach! Innerhalb von *<titelliste></titelliste>* befinden sich mehrere *<heft></heft>*-Paare. Das Plus-Zeichen hinter dem Tag bedeutet dabei lediglich, dass mehrere Elemente vorhanden sein dürfen. Würdest du das Plus-Zeichen weglassen, dürfte *<heft></heft>* nur ein einziges Mal auftreten.

```
<!ELEMENT titelliste (heft+)>
```

Du kannst das Plus-Zeichen hier auch hinter den runden Klammern notieren: `(heft)+`. Dann wirkt es auf die „Gruppe", denn die runden Klammern kennzeichnen eine Gruppe. Da *heft* in diesem Fall das einzige Element der Gruppe ist, ist die Position des Plus-Operators schnurzpiepegal.

2. Das Element *heft* wiederum enthält weitere Elemente. Diese musst du ebenfalls in runde Klammern setzen, also zu einer Gruppe zusammenfassen. Du reihst diese Elemente auf, und zwar einfach durch Kommas getrennt. Da das Komma als Trenner dient, musst du kein Leerzeichen zwischen den Elementen setzen, du kannst es aber machen.

```
<!ELEMENT heft (titel,autor+,verlag,beschreibung?,preis+)>
```

**Und bzw. Oder**: Mit diesem Komma als Trenner legst du eine strikte Reihenfolge fest! Das heißt, dass die Tags jetzt nur in dieser Reihenfolge verwendet werden dürfen. Man sagt dazu auch *Und*-Verknüpfung; titel *und* autor *und* verlag ... Wenn du dagegen eine beliebige Reihenfolge wünschst, eine *Oder*-Verknüpfung, setzt du einen senkrechten Strich | ( [AltGr] + „<>") und notierst hinter der schließenden runden Klammer ein Plus-Zeichen. Mehr dazu ab Seite 47.

3. Die Elemente *titel* und *verlag* kommen innerhalb von *<heft></heft>* jeweils nur ein einziges Mal vor. Deshalb müssen wir sie nicht weiter markieren. Nur bei den Elementen *autor*+ und *preis*+ findest du dagegen wieder den Plus-Operator. Aber das ist ja klar! Der Plus-Operator besagt schließlich, dass das Element mindestens einmal vorkommen muss und beliebig oft wiederholt werden kann. Man spricht in diesem Zusammenhang auch vom Wiederholungsoperator Plus.

```
<!ELEMENT heft (titel,autor+,verlag,beschreibung?,preis+)>
```

4. Doch was bedeutet das Fragezeichen hinter dem Element *beschreibung*? Dieser „Wiederholungsoperator" sorgt dafür, dass dieses Element optional (freiwillig) ist. Du kannst also eine Beschreibung setzen, musst es jedoch nicht tun.

```
<!ELEMENT heft (titel,autor+,verlag,beschreibung?,preis+)>
```

5. Jetzt haben wir die Reihenfolge und das Vorkommen der einzelnen Elemente festgelegt. In den nächsten fünf Zeilen werden diese Elemente nun der Reihe nach definiert. Dazu schreibst du hinter die Zeichenfolge <!ELEMENT einfach den Namen des Tags, beispielsweise *titel*. In runden Klammern folgt danach die komische Anweisung *(#PCDATA)*.

```
<!ELEMENT titel (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT verlag (#PCDATA)>
<!ELEMENT beschreibung (#PCDATA)>
<!ELEMENT preis (#PCDATA)>
```

## Wofür steht PCDATA?

PCDATA heißt, dass es sich bei diesem Element um *character data* handelt, um aus Zeichen bestehende Daten und nicht um Grafiken oder Multimedia-Elemente. Das P steht für parsed, also für „analysiert". Insgesamt ist PCDATA also die Abkürzung für ***parsed character data***.

Wie du merkst, ist die Syntax für eine DTD schon ziemlich kompliziert. Und dabei habe ich dir bisher noch nicht einmal alle Schlüsselwörter und Indikatoren gezeigt.

## Weitere Schlüsselwörter und Indikatoren

Die folgende Tabelle zeigt weitere für DTDs wichtige Symbole, Schlüsselwörter und Indikatoren:

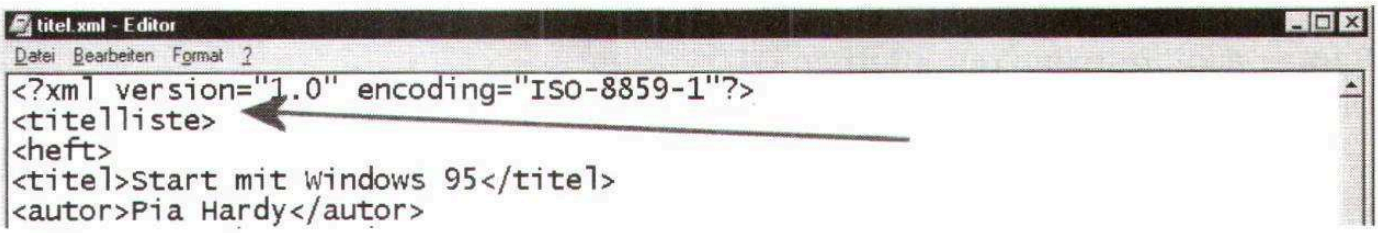
| Symbol | Erklärung | Beispiel |
|---|---|---|
| () | Die runden Klammern gruppieren Untertags, sie enthalten Attributwerte u. Schlüsselwörter. | <!ELEMENT titelliste (titel,autor+,verlag)> |
| , | Und-Verknüpfung: Das Komma steht für die genaue Anordnung der Elemente in der vorgegebenen Reihenfolge. | (titel,autor,verlag) |
| \| | Oder-Verknüpfung: Der senkrechte Strich heißt, dass nur eines der aufgereihten Elemente verwendet werden darf! | (titel\|autor\|verlag) |
| ohne | Wenn keine Markierung erfolgt, muss das Element einmal vorhanden sein. | (titel,autor,verlag) |
| + | Das Plus-Zeichen besagt, dass das Element ein oder mehrmals vorhanden ist. | autor+ |
| ? | Das Fragezeichen steht dafür, dass das Element optional ist. Es muss nicht verwendet werden. Wenn es erscheint, darf es aber nur einmal verwendet werden. | beschreibung? |
| * | Mit dem Sternchen markiert man, dass das Element beliebig oft verwendet werden kann, aber nicht verwendet werden muss! | stichworte* |
| #PCDATA | Diese Zeichenfolge steht für parsed character data, das Element kann also beliebige Zeichenfolgen enthalten, aber keine Multimediainhalte. | <!ELEMENT autor (#PCDATA)> |

## So setzt du einen Verweis auf die DTD

Die DTD wäre geschafft! Wunderbar. Fehlt nur noch eine Kleinigkeit, und zwar der Verweis auf die DTD.

**Dieser Verweis wird auch als Dokumenttyp-Deklaration bezeichnet!**

1. Rufe wieder das XML-Dokument auf, im Beispiel die Datei *titel.xml*. Gehe in den Quelltext. Tipp: Du hast die Datei im Internet Explorer aufgerufen? Dann hilft der Befehl *Ansicht/Quelltext*.
2. Der Verweis gehört unter den Prolog und über das Wurzelelement. Schaffe an dieser Stelle eine Leerzeile durch Druck auf [ENTER].




3. Tippe jetzt folgende Zeile:

```
<!DOCTYPE titelliste SYSTEM "titel.dtd">
```

## Aufbau der Dokumenttyp-Deklaration

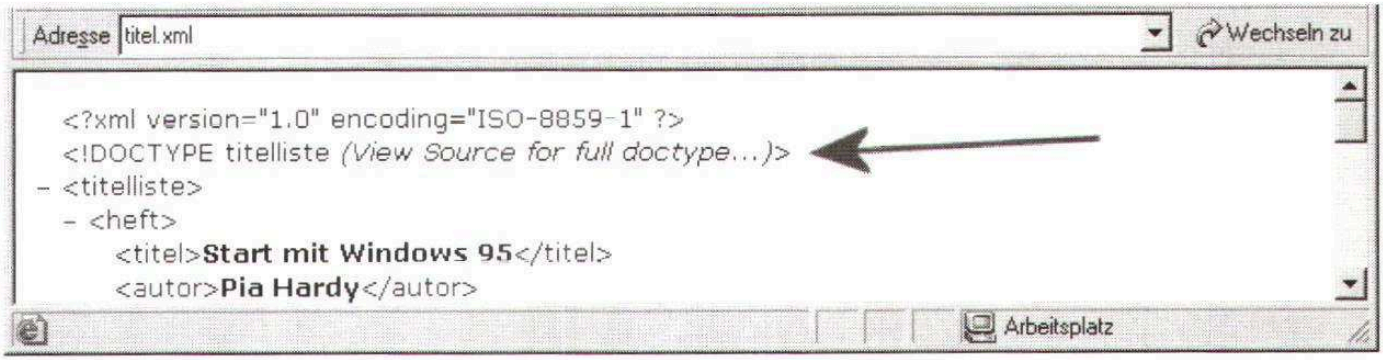
Was verbirgt sich hinter der Zeile `<!DOCTYPE titelliste SYSTEM "titel.dtd">`?

- Die DT-Deklaration wird durch <!DOCTYPE eingeleitet, und zwar in Großschreibung!
- Hinter das Schlüsselwort DOCTYPE setzt du den Namen, *den das Wurzelelemt deines XML-Dokuments trägt*. Das ist in unserem Fall *titelliste*.
- Jetzt folgt das Schlüsselwort SYSTEM. Dieses gibt an, dass die DTD lokal „auf unserem System" gültig ist. Sie gilt nur für unsere XML-Dateien und ist nicht öffentlich.

Nur nebenbei: Im Gegensatz dazu wird bei HTML-Dokumenten auf eine öffentliche DTD verwiesen. Diese liegt auf den Servern des W3C. Die Zeile beginnt häufig *<!DOCTYPE HTML PUBLIC ...>* Am Ende folgt der Pfad zur DTD auf dem W3C-Server. Wenn dich das Thema „HTML und DTD" interessiert (braucht man überhaupt DTDs bei HTML?), surfe einfach zu www.jchanke.de/homepage, klicke rechts oben auf Feature-Artikel und lies meinen Feature-Artikel zur DTD.

- Last but not least notierst du in Gänsefüßchen den Pfad zur eigentliche Datei, in welcher die DTD liegt. Da diese im gleichen Ordner liegt, schreiben wir nur *"titel.dtd"*.

Wenn alles glatt gegangen ist, müsste dir der Internet Explorer nach dem Aufruf folgendes zeigen:




**Das Vorhandensein einer DTD wird erkannt, mehr passiert jedoch nicht**

## Zusatzwissen: Interne versus externe DTD

Egal ob du bei mir CSS oder JavaScript lernst ... immer lernst du zwei Varianten kennen. In der Regel zeige ich dir zuerst, wie du den betreffenden CSS- oder JavaScript-Abschnitt intern (also im HTML-Dokument) speicherst. Oft machen wir das aus Bequemlichkeit oder einfach aus pädagogischen Gründen. Danach gehe ich natürlich auch auf die entsprechende externe Variante ein.

In diesem Heft habe ich dich ermuntert, die DTD gleich extern auszulagern. Die Vorteile liegen auf der Hand: Wenn du einmal eine DTD erstellt hast, kannst du sie für beliebig viele XML-Dateien anwenden. Nutze ein Strickmuster für viele XML-Dateien.

Der Vollständigkeit halber möchte ich dir jedoch auch verraten, wie eine interne DTD notiert wird. Die Syntax zeige ich dir gleich am Beispiel unserer Titelliste!

### Syntax zum Erstellen der internen DTD

Die Syntax zum Erstellen der internen DTD ist ganz einfach. Zuerst zeige ich dir noch einmal zum Vergleich den Verweis auf die externe DTD, und zwar gleich schematisch:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE wurzelelement SYSTEM "name.dtd">
<wurzelelement>
```

Wenn du die DTD (hier als Datei *name.dtd*) intern notieren möchtest, schreibst du:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE wurzelelement [
Anweisungen
]>
<wurzelelement>
...
</wurzelelement>
```

Der Anfang der Titelliste sieht folgendermaßen aus. Du findest das Beispiel auch unter dem Namen *titelintern.xml* im Ordner *lektion3*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE titelliste [
<!ELEMENT titelliste (heft+)>
<!ELEMENT heft (titel,autor+,verlag,beschreibung?,preis+)>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT verlag (#PCDATA)>
<!ELEMENT beschreibung (#PCDATA)>
<!ELEMENT preis (#PCDATA)>
]>
<titelliste>
    <heft>
      <titel>Start mit Windows 95</titel>
      <autor>Pia Hardy</autor>
      <autor>Kare Thomsen</autor>
      <verlag>KnowWare</verlag>
      <preis>4</preis>
    </heft>
    ...
</titelliste>
```

# Lektion 4: XML-Dokument auf Gültigkeit prüfen

In dieser Lektion lernst du folgendes:

- Warum eine DTD Sinn macht
- Unterschied zwischen validierenden und non-validierenden Parsern
- Praxis: Validieren eines Dokumentes

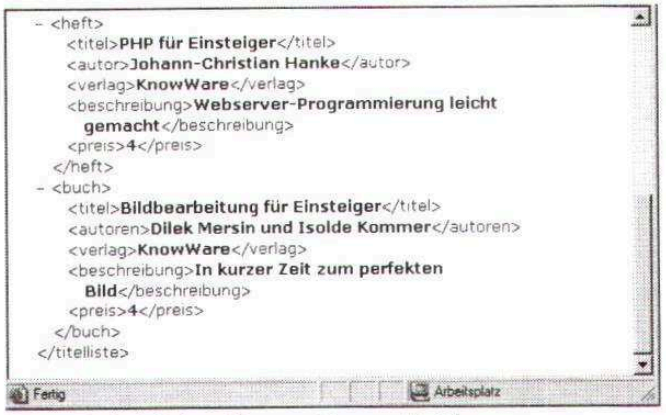
## Sinn und Unsinn der DTD

Bisher hast du das XML-Dokument und die korrespondierende DTD erstellt. Doch wozu überhaupt die DTD? Könnte man den Regelsatz nicht weglassen?

Na klar könnte man das. Dann erhielte man – wenn keine Verschreiber enthalten sind – immer noch ein wohlgeformtes Dokument.

Verstehe die DTD jedoch als Plan, den man vorher festlegt, als eine Art Strickmuster. Und es macht Sinn, sich an einen Plan zu halten.

Nehmen wir einfach unsere Titelliste. Angenommen, mehrere Mitarbeiter arbeiten an dieser Liste und führen die jeweiligen Datensätze später zusammen.




**Ohne einheitliches Konzept herrscht Chaos**

Doch der eine Mitarbeiter nennt die Felder statt <autor></autor> einfach *<autoren></autoren>*. Statt in <heft></heft> werden die Titel in *<buch></buch>* eingekleidet.

Natürlich klappt die Anzeige weiterhin. Auch so entstehen wohlgeformte „Datenbanken". Doch ohne „einheitliches Konzept" herrscht Chaos.

Wer soll nachher noch wissen, zwischen welchen der vielen unterschiedlichen Tags nun die jeweiligen Informationen zu finden sind?

## Wer überprüft den Code?

Wie du siehst, macht eine DTD (oder ein Schema) auf jeden Fall Sinn!

Nutzen wir also die DTD, das von uns entworfene Regelwerk. Alle weiteren Eingaben sollen sich diesen Regeln unterwerfen. Doch wie sieht diese „Unterwerfung" aus? Prüft da jemand, ob alles dem DTD-Muster entsprechend eingetragen wurde?

Testen wir es. Füge einen weiteren Datensatz in das Dokument *titel.xml* ein, in den wir vorsätzlich einige Fehler einbauen.

Speichere die Datei unter einem anderen Namen, beispielsweise unter *titelchaos.xml*.

Hier mein Vorschlag, der nur noch wohlgeformt, aber nicht mehr gültig ist:

```
<buch>
    <titel>Bildbearbeitung für
    Einsteiger</titel>
    <autoren>Dilek Mersin und Isolde
    Kommer</autoren>
    <verlag>KnowWare</verlag>
    <beschreibung>In kurzer Zeit zum
    perfekten Bild</beschreibung>
    <preis>4</preis>
</buch>
```

## Aufrufen im Internet Explorer

Rufe das Dokument bitte im Internet Explorer auf. Was stellst du erstaunt fest? Es passiert ... gar nichts! Der Browser prüft zwar, ob bei einem Verweis auch eine DTD vorhanden ist.

Ansonsten ist es dem Internet Explorer herzlich egal, was du dir bisher in der DTD ausgedacht hast! Egal ob Schreibung der Tags, Reihenfolge, Vorkommen – in der Regel wird es ignoriert.

In diesem Zusammenhang spricht man von **non-validierenden** Parsern.

Non-validierende Parser prüfen nicht!

Vielleicht denkst du jetzt: Da hätte ich mir die Mühe mit der DTD doch sparen können. Aber ...

## Bitte in die Zukunft denken!

Doch mit XML musst du immer auch in die Zukunft denken! Denn wie schon erwähnt, unsere „Internet-Explorer-Geschichte" ist lediglich die Notlösung für diesen Kurs.

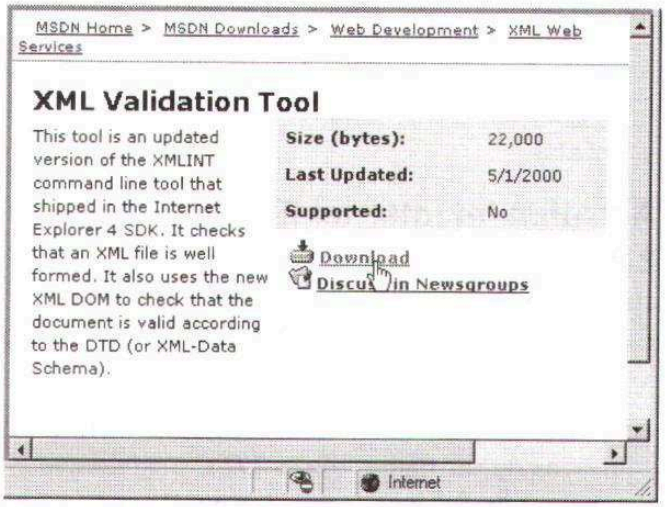
Künftige Programme werden sehr wohl testen, ob der Code korrekt ist. Sie müssen es sogar, denn die Eingabe von Daten (beispielsweise in einer Datenbank) macht ja nur dann Sinn, wenn du dich an das (selbst geschriebene) Regelwerk hältst.

Programme, die den Code anhand des Regelwerks prüfen (validieren), werden **validierende Parser** genannt.

## Prüfprogramm XMLINT

Doch wie findest du nun heraus, ob dein Code korrekt ist? Nutze Prüfprogramme, Tools, deren einzige Aufgabe im Fehlercheck besteht. Und da gibt es gleich mehrere ...

Ich habe für dich zum einen das Programm *xmlint.exe* aufgespürt. Dieses besticht durch seine Einfachheit und geringe Dateigröße.




**Nicht ganz taufrisch, aber kostenlos und funktionsfähig**

Das Programm XMLINT wurde dir bis vor kurzem von Microsoft zur Verfügung gestellt, und zwar auf http://msdn.microsoft.com/downloads. Suche nach `xmlint`! Findest du es noch? Lade dir das 23 kB große ZIP-Archiv herunter und entpacke es! (Ansonsten schaue doch mal in die Beispieldateien auf www.jchanke.de/xmlkurs, und zwar in den Ordner `lektion4`.)

## So validierst du mit XMLINT

Hast du die Dateien ausgepackt, beispielsweise mit Windows XP? Mit WinZip? Oder mit Free-Zip, welches du hier kostenlos bekommst?

members.ozemail.com.au/~nulifetv/freezip

Dann verfügst du über mehrere Dateien, von denen die *xmlint.exe* für uns am interessantesten ist. Es handelt sich um ein Programm, das auf DOS-Kommandozeilenebene bedient werden muss. Hier meine Schrittfolge, um mit dem Programm erfolgreich arbeiten zu können:

1. Kopiere die *xmlint.exe* nebst Begleitdateien in den gleichen Ordner, in dem auch die zu prüfende XML-Datei liegt.
2. Öffne eine DOS-Box. Dazu wählst du *Start/Ausführen*. Das *Ausführen*-Dialogfenster erscheint. Hier kannst du Befehle eintragen. Unter Windows 95/98/Me tippst du `command`. In Windows 2000/XP dagegen `cmd`. Drücke [ENTER]. Jetzt landest du in der DOS-Box!
3. Gehe mit den entsprechenden DOS-Befehlen an die gewünschte Stelle. Tippe zur Sicherheit zuerst

   ```
   cd \
   ```

   und [ENTER]. So gelangst du auf jeden Fall erst einmal ins Stammverzeichnis.
4. Mit *cd Ordnername* navigierst du nun zum gewünschten Ordner. Die Prüfdateien liegen unter *C:\xmlkurs*, wie von mir empfohlen? Dann tippe

   ```
   cd xmlkurs
   ```
5. Tippe jetzt einen Befehl nach dem Muster *xmlint Dateiname*. Dateiname ist natürlich der Name der zu testenden XML-Datei (*titelchaos.xml*). Schreibe also im Beispiel:

   ```
   xmlint titelchaos.xml
   ```

   und drücke [Enter].

## Ist das Dokument fehlerfrei?

Wenn das Dokument fehlerfrei ist, dürfte nichts weiter passieren. Unser Musterdokument ist aber nicht fehlerfrei. Spätestens die Verwendung des Eintrags *<buch></buch>* statt *<heft> </heft>* verstößt gegen die Regeln in der DTD!


**Der validierende Parser findet den Fehler!**

Jeder Regelverstoß wird mit einer Fehlermeldung quittiert. Diese liefert einen Hinweis auf die entsprechende Stelle.

Beachte, dass immer nur der erste Fehler angezeigt wird. Du merkst es dann, wenn du <buch></buch> in <heft></heft> umänderst. Jetzt kritisiert XMLINT noch die Stelle, an der <autoren></autoren> statt <autor></autor> steht.

## Validieren im Internet Explorer

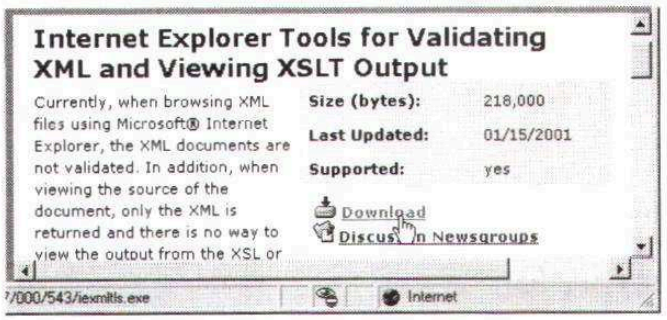
Wenn du möchtest, kannst du auch komfortabler validieren. Und zwar über den Internet Explorer. Lade dir dafür das zweite von mir entdeckte Programm herunter, *die Internet Explorer Tools for Validation XML and viewing XSLT Output*.

Surfe einfach zu

http://www.google.de

und suche nach der Zeichenfolge:

`iexmltls download`




**Lade dir die Datei iexmltls.exe auf den Rechner**

Gefunden? Dann klicke auf Download, bestätige die Lizenzvereinbarung und lade dir die Datei *iexmltls.exe* auf den Rechner.

## Validator installieren

In welchen Ordner hast du die Datei abgelegt? Bei mir landet alles in einem Ordner *download*. Doppelklicke auf die Datei *iexmltls.exe*. Folge den Schritten des „installation wizard".

Im Endeffekt wird der Ordner *iexmltls* auf deiner Festplatte eingerichtet, direkt unter C:\.


**Installiere die beiden Dateien mit der Endung *inf***

Hier interessieren uns jetzt die beiden Dateien mit der Endung *inf*. Es handelt sich um:

- msxmlval.inf
- mxcmlvw.inf

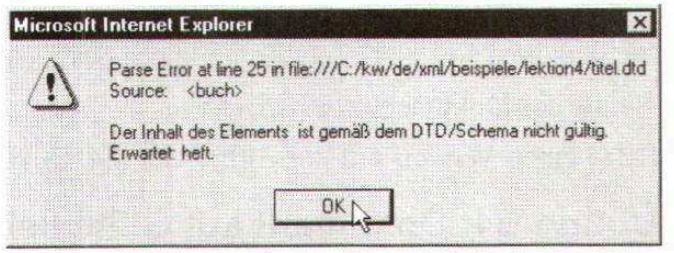
Klicke diese Dateien jeweils mit der rechten Maustaste an. Wähle im Kontextmenü den Befehl *Installieren*. Gedulde dich einen Moment.

## Validieren über Kontextmenü

Ab sofort findest du im Kontextmenü des Internet Explorers zwei neue Befehle vor, von denen uns nur:

- Validate XML

interessiert. Rufe die zu prüfende XML-Datei im Browser auf. Klicke mit der rechten Maustaste in das Dokument und wähle *Validate XML*.




**Die gleiche Fehlermeldung – diesmal als Dialogfenster**

Hinweis: Falls das Programm nicht mehr erhältlich sein sollte, musst du bei XMLINT bleiben. Abgesehen von der fehlenden „grafischen Oberfläche": Die Fähigkeiten sind die gleichen.

# Übungsteil B: Erste Übungen zu XML

Du weißt jetzt auch, wie man:

- wohlgeformte XML-Dokumente erstellt
- Prolog, Tags und Wurzelelement erzeugt
- DTDs schreibt und XML gültig macht
- XML-Dateien validiert

In diesem Übungsteil wiederholst und festigst du die Kenntnisse der ersten vier Lektionen. Alle Lösungen findest du selbstverständlich auf der Downloadseite www.jchanke.de/xml.

## Übung B1: Fragespiel: Einfaches XML-Dokument erstellen

Erstelle folgendes einfaches XML-Dokument: Das Wurzelelement soll *<fragespiel>* heißen. Innerhalb dieser Klammer sollen die Tags *<frage>* und *<antwort>* vorkommen. Den Inhalt kannst du dir frei ausdenken. Nenne das Dokument *nochfragen.xml*.

## Übung B2: DTD für Adressliste erstellen

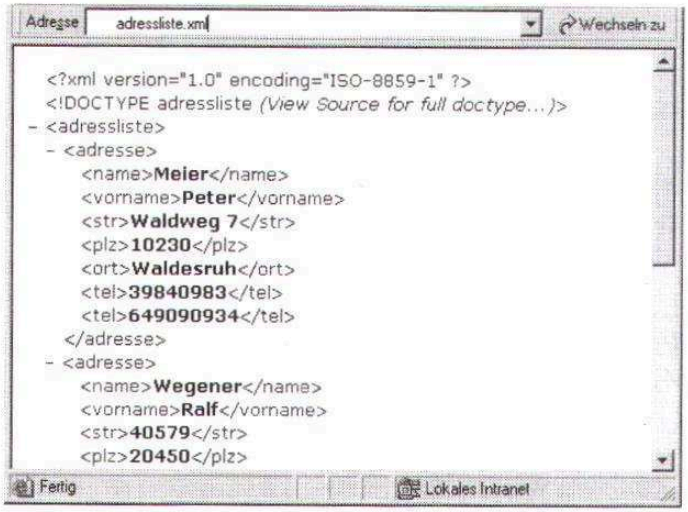
Plane ein umfangreiches Projekt, eine Adressliste. Beginne mit der DTD, der Regelanweisung.

Erstelle zuerst die Dokumenttypdefinition (DTD) für eine Adressliste in XML. Das Wurzelelement soll *adressliste* heißen.

Ein Adress-Datensatz wird innerhalb von *<adresse></adresse>* gespeichert

Die Struktur soll folgendermaßen aussehen und diese Reihenfolge besitzen: **name - vorname - str - plz - ort - tel - fax - email - www**

Nenne das Dokument *adressen.dtd*. Berücksichtige, dass die Telefonnummer auch mehrfach vorkommen kann. Berücksichtige außerdem, dass Faxnummer, E-Mail-Adresse und Webseite nicht bei jedermann selbstverständlich sind!




## Übung B3: XML-Dokument erstellen

Schreibe nun ein XML-Dokument, in welches du zur Probe drei Adressen einträgst. Denke dir die Adressen aus. Trage teilweise auch Faxnummer, E-Mail-Adresse und Webseite ein. Eine der drei Kontaktpersonen soll zwei Telefonnummern besitzen.

Nenne das Dokument *adressliste.xml*.

Setze einen Verweis auf die vorher erstellte Dokumenttyp-Definition.

## Übung B4: Dokument auf Gültigkeit überprüfen

Überprüfe das Dokument auf Gültigkeit.

# Lektion 5: Tags durch Attribute genauer bestimmen

In dieser Lektion lernst du folgendes:

- Baumstruktur mit Skizze planen
- Unterknoten einbauen
- Tags durch Attribute genauer bestimmen
- Attribute in DTD und XML-Dokument einbauen

In dieser Lektion üben wir die Arbeit mit XML an einem zweiten großen Projekt. Diesmal wird die Struktur noch weiter „verästelt".

Dabei zeige ich dir auch, wie hilfreich eine Skizze sein kann.

## Wir planen die Produktdatenbank

Richtig, XML eignet sich vor allem für gut strukturierte Daten. Nehmen wir diesmal einfach eine *Produktdatenbank* als Beispiel. Wir wollen folgendes festhalten:

- Produktnamen
- Nummer des Produkts
- Hersteller
- Preis

Den *Hersteller* wollen wir zusätzlich zerlegen in:

- Firma
- Ort

Hier soll jedoch gelten: Der Ort des Herstellers ist nicht immer bekannt. Er ist nicht zwingend vorgeschrieben und bleibt damit fakultativ.

Auch beim *Preis* gibt es eine Besonderheit. Es handelt sich zwar um ein Tag, jedoch um „zwei Geschmacksrichtungen". Wir wollen zwei verschiedene Preise darstellen:

- Privatkunde (Endpreis)
- Handel (60% des Endpreises)

Für diese Aufgabe stehen uns Attribute zur Verfügung. Das sind die Vorüberlegungen. Denken wir uns nun die dafür benötigten Feldnamen aus.

## XML-Elemente und Feldnamen

Ich wusste es! Spätestens an dieser Stelle ist mir doch tatsächlich das Wort *Feldname* „herausgerutscht".

Das zeigt jedoch, wie viel das Planen von XML-Strukturen eigentlich mit Datenbankdesign zu tun hat. Denn gerade bei Datenbanken muss man die Felder vorher genau festlegen. Und auch bei XML-Dateien ist es wichtig, sich vorher zu überlegen, welche Tags (bzw. Elemente) benötigt werden.

Wie nennen wir unsere Elemente?

Das Wurzelelement (also die alles umspannende Klammer) heißt im Beispiel

- produkte

Wir haben jedoch mehrere „Posten", sprich mehrere Produkte. Deshalb „umklammern" wir jedes einzelne Element mit

- posten

Jeder „Posten" (jedes Produkt) wird im Beispiel durch folgende Felder charakterisiert:

- name (für Produktnamen)
- nr (Nummer des Produkts)
- hersteller (Hersteller des Produkts)
- preis (durch Attribute untergliedert!)

Da wir das Feld *hersteller* untergliedern müssen, kommen noch die Unterfelder

- firma (Firmenname)
- ort (Firmensitz, nicht immer bekannt)

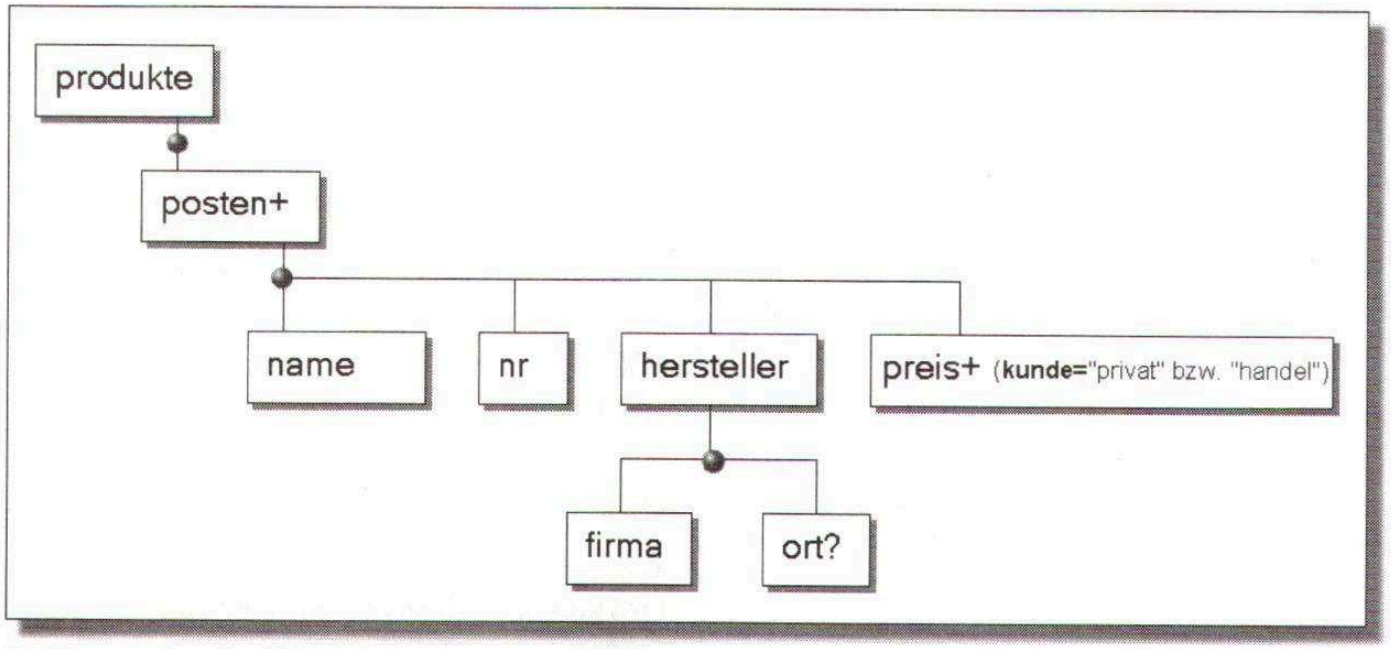
hinzu.

## Erst die DTD planen

Mit diesem Wissen können wir uns schon fast an die Planung der DTD heranwagen. Denn diesmal machen wir es so – wir planen zuerst die DTD. Damit es noch verständlicher wird, zeichne ich zuerst eine Skizze!

## Bessere Übersicht durch die grafische Darstellung

Schau dir meine Skizze an. Hier habe ich die Grundanordnung der eben von mir festgelegten Elemente optisch verdeutlicht. Außerdem habe ich – falls nötig – gleich die entsprechenden Zeichen in die Übersicht eingefügt. Die Kreise symbolisieren die entsprechenden Abzweige bzw. *Knoten*.




**Die Baumstruktur des XML-Dokuments ist in der Skizze gut zu erkennen**

Das Wurzelelement *produkte* steht in der Hierarchie an allerhöchster Stelle. Danach folgen die einzelnen *posten*. Das Plus-Zeichen hinter posten besagt lediglich, dass das Element wiederholt werden kann, und zwar mindestens einmal. Eine Produktdatenbank mit nur einem Produkt macht schließlich nicht sehr viel Sinn.

Auch das Fragezeichen ist sicher noch bekannt: Dieser Wiederholungs-Operator sorgt dafür, dass das Element optional ist. Sprich: Der Ort kann, muss aber nicht angegeben werden. Mit diesem Wissen können wir später die DTD aufbauen. Denn in diesem Beispiel „machen wir es gleich richtig" und beginnen mit der DTD. Doch vorher führe ich dich in die Geheimnisse der Attribute ein!

## Tags durch Attribute näher bestimmen

Attribute dienen dazu, Tags näher zu bestimmen. Du kennst das Prinzip sicher schon aus HTML.

Du möchtest in HTML einen Absatz einleiten? Ein normaler Absatz beginnt beispielsweise mit dem Paragraph-Tag <*p*>. Wenn du den Absatz rechtsbündig stellen willst, schreibst du dagegen die Zeichenfolge <*p align="right"*>, ein linksbündiger Absatz würde mit <*p align="left"*> umschrieben. Im Beispiel haben wir das Tag durch das Attribut *align* näher bestimmt, die Werte heißen *right* bzw. *left* (oder justify). In diesem Beispiel kann das Attribut verwendet werden, es muss aber nicht.

Auch in XML kannst du deine Tags selbstverständlich durch Attribute näher bestimmen. Im Beispiel soll das Tag <*preis*> zwei verschiedene „Geschmacksrichtungen" bekommen. Dafür habe ich mir das Attribut *kunde* ausgedacht. Dieses wiederum soll folgende Werte annehmen können:

- privat
- handel

Der Preis für den Privatkunden wird so umschrieben: <*preis kunde="privat"*>. Geht das Produkt an den Handel, gilt folgende Schreibweise: <*preis kunde="handel"*>. Wir wollen außerdem erreichen, dass das Attribut verwendet werden muss!

## Attribute in der DTD definieren

Wie die mit Attributen geschmückten Tags ausschauen, haben wir eben gesehen. Doch wie definierst du diese Attribute in der DTD? Dafür brauchst du folgende Syntax:

```
<!ATTLIST Tagname Attributname Typ #OPTION>
```

Dabei steht das Wort *ATTLIST* als Abkürzung für *Attributliste*. Dahinter notierst du Tag und Attributnamen, lediglich durch Leerzeichen getrennt. Das Wort *Typ* steht für den Typ des Attributs. Im Beispiel soll dieser nach folgendem „Typ-Muster" aufgebaut werden:

- `(Wert1|Wert2|usw.)`

Was verbirgt sich hinter diesem „Typ-Muster"? Nun, wir geben die Attributwerte einfach vor. In runden Klammern listest du die von dir selbst zu bestimmenden Attribut-Werte auf. Die Anzahl steht dir frei. Nimm zwei wie im Beispiel oder unendlich viele. Verwende als Trenner nicht das Komma, sondern immer nur den senkrechten Strich. Es kommt hier nicht auf eine Reihenfolge an.

Doch was bedeutet #OPTION? Das Wort steht als Platzhalter für eine Option wie z.B. #REQUIRED. Required heißt so viel wie „verlangt". Es bedeutet, dass das Attribut unbedingt erforderlich ist. Sprich: Es muss verwendet werden.

Sollte das Attribut nicht erforderlich sein, setzt du an diese Stelle das Wort #IMPLIED. Denke an das HTML-Beispiel mit <p>. Hier wären keine Attribute erforderlich.

Zurück zu unserem Beispiel. Das Tag *preis* soll das Attribut *kunde* erhalten. Die möglichen Werte sind *privat* und *handel*. Ein Attribut ist „required", also Pflicht. Dann schreibst du das folgendermaßen:

```
<!ATTLIST preis kunde (handel|privat) #REQUIRED>
```

## So sieht die DTD aus:

Mehr zu weiteren Attribut-Optionen erfährst du in der nächsten Lektion. Jetzt haben wir genug Wissen „gesammelt", um erfolgreich die DTD schreiben zu können. Notiere folgende Zeilen und sichere sie unter dem Namen *produkte.dtd*. Die Zeile mit der Attributdefinition habe ich grau hinterlegt:

```
<!ELEMENT produkte (posten+)>
<!ELEMENT posten (name,nr,hersteller,preis+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nr (#PCDATA)>
<!ELEMENT hersteller (firma,ort?)>
<!ELEMENT firma (#PCDATA)>
<!ELEMENT ort (#PCDATA)>
<!ELEMENT preis (#PCDATA)>
<!ATTLIST preis kunde (handel|privat) #REQUIRED>
```

## Die Anweisungen der DTD im Überblick

Am besten wir gehen die Anweisungen der DTD Schritt für Schritt noch einmal durch. Damit wiederholst und festigst du deine bisherigen Kenntnisse.

1. Die erste Zeile definiert das Wurzelelement, hier *produkte*. Diesem sind mehrere Posten untergliedert. Das zeigen wir an durch den Plus-Operator, den wir in diesem Fall übrigens auch hinter die schließende runde Klammer setzen könnten.

```
<!ELEMENT produkte (posten+)>
```

2. In den Posten sind die Tags *name*, *nr*, *hersteller* und *preis* enthalten. Dabei gilt: Der Preis muss mindestens zweimal vorkommen. Alle anderen Tags existieren innerhalb von *posten* nur ein einziges Mal. Das Komma gibt die exakte Reihenfolge vor, es sorgt für die Und-Verknüpfung.

```
<!ELEMENT posten (name,nr,hersteller,preis+)>
```

3. Die nächsten beiden Elemente sichern, dass die Tags beliebige Zeichen enthalten können, dafür sorgt die Anweisung *#PCDATA*.

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

4. Eine Zeile tiefer legst du die nächste Verschachtelungsebene fest: Das Tag *hersteller* soll die Tags *firma* und möglicherweise auch das Tag *ort* umschließen.

```
<!ELEMENT hersteller (firma,ort?)>
<!ELEMENT firma (#PCDATA)>
<!ELEMENT ort (#PCDATA)>
```

5. In den nächsten beiden Zeilen geht es um die Definition der Attribute für das Tag *preis*. Zuerst weisen wir dem Element das Schlüsselwort *#PCDATA* zu.

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

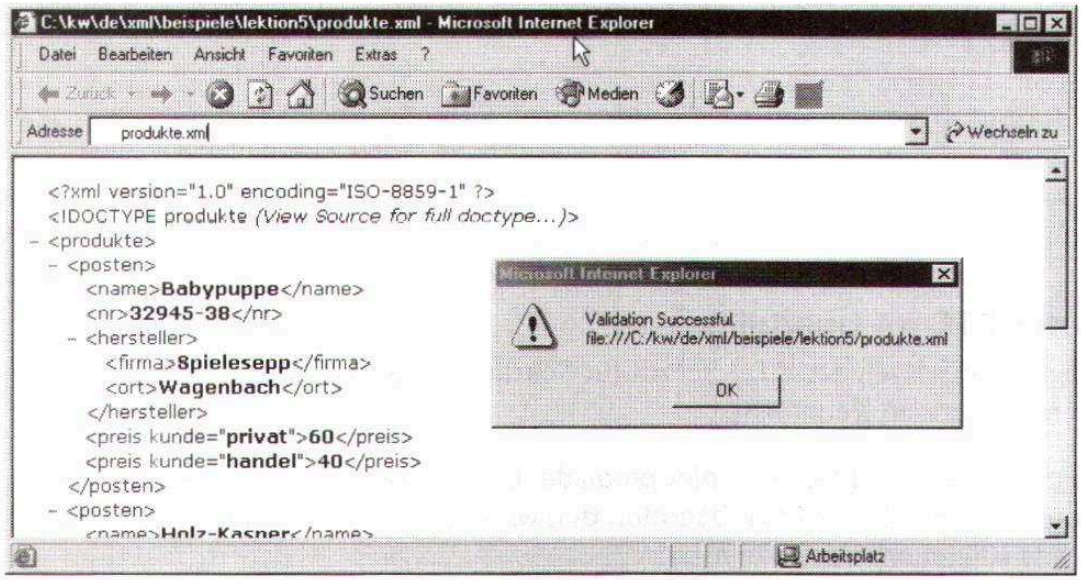
6. In der Zeile darunter geht es um die Definition des Attributs. Das Attribut *kunde* bekommt die Werte *handel* und *privat* zugewiesen. Mit dem Schlüsselwort *#REQUIRED* legen wir fest, dass die Attribute zwingend vorgeschrieben sind.

```
<!ATTLIST preis kunde (handel|privat) #REQUIRED>
```

## Das XML-Dokument

Die DTD ist fertig, fehlt noch die XML-Datei. Auf der nächsten Seite zeige ich dir das komplette Dokument im Überblick.

Denke nach dem Abschreiben auch an den entsprechenden Test zum Validieren. Denn nur so kannst du evtl. Verschreiber von vornherein ausschließen.




**Prüfe, ob das Dokument gültig ist. Hier erfolgt der Test per Kontextmenü-Befehl *Validate XML***

## Das XML-Dokument im Überblick

Schreibe den Quelltext ab und speichere ihn unter dem Namen *produkte.xml*. Setze gleich einen Verweis auf die eben erstellte DTD (grau hinterlegt).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE produkte SYSTEM "produkte.dtd">
<produkte>
   <posten>
      <name>Babypuppe</name>
      <nr>32945-38</nr>
      <hersteller>
            <firma>Spielesepp</firma>
            <ort>Wagenbach</ort>
      </hersteller>
      <preis kunde="privat">60</preis>
      <preis kunde="handel">40</preis>
   </posten>
   <posten>
      <name>Holz-Kasper</name>
      <nr>32945-41</nr>
      <hersteller>
            <firma>Spielesepp</firma>
            <ort>Wagenbach</ort>
      </hersteller>
      <preis kunde="privat">18</preis>
      <preis kunde="handel">12</preis>
   </posten>
   <posten>
      <name>Holzeisenbahn basic</name>
      <nr>32945-41</nr>
      <hersteller>
            <firma>Natura GmbH</firma>
      </hersteller>
      <preis kunde="privat">80</preis>
      <preis kunde="handel">60</preis>
   </posten>
</produkte>
```

Dieses Beispiel werden wir weiter hinten wieder aufgreifen. Ab Seite 58 zeige ich dir, wie du eine XML-Datei mit CSS gestalten kannst. Auf die Vorteile von XSL gehe ich dagegen ab Seite 63 ein.

# Lektion 6: Mit und ohne Vorgabewert? Mehr zu Attributen!

In dieser Lektion lernst du folgendes:

- 1. Attribut ohne Vorgabe definieren
- 2. Attribute mit Optionalwert einsetzen
- 3. Attributwert voreinstellen
- 4. Voreinstellung festschreiben

An dieser Stelle bist du schon mit Attributen vertraut. Schauen wir uns weitere Optionen für die praktischen „Bestimmer" an!

Die Beispieldateien für diese Lektion findest du im Ordner *lektion6* in nach Seiten benannten Unterordnern: www.jchanke.de/xml.

## 1. CDATA: Keinen Wert vorgeben

Was hast du bisher zu Attributen gelernt? Du kannst Werte vorher festlegen. Sorge dafür, dass diese Pflicht sind (#REQUIRED) oder aber „freiwillig" eingesetzt werden können (#IMPLIED). Der Haken: Bisher haben wir erstens immer nur ein Attribut gesetzt und zweitens den Attributwert oder die Attributwerte gleich mit vorgegeben.

### Beispiel: Tag zur „Bildbeschreibung"

Nehmen wir ein Beispiel aus dem Grafik-Bereich. Für irgendeine XML-Anwendung brauchst du ein Tag, welches einen Bildnamen speichert. Im Beispiel ist es das (imaginäre) Bild *rose.tif*. In diesem Tag sollen per Attribut die Breite und Höhe der Grafik angegeben werden.

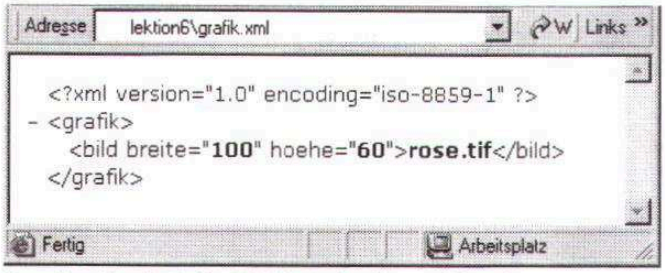
Das Tag sieht im Beispiel folgendermaßen aus:

```
<bild breite="100" hoehe="60">
rose.tif</bild>
```

Beachte, dass natürlich auch in Attributnamen keine Sonderzeichen erlaubt sind!

- Wir geben zwei Attribute an, im Beispiel heißen sie *breite* und *hoehe*.
- Wir geben keinen Wert vor.

Denke an die Größe der Grafiken. Diese besitzen eine ganz bestimmte Breite und Höhe. Es macht aber wenig Sinn, alle Werte vorzugeben, da sowohl Pixelwerte von 1 bis 600, 800 oder sogar 1024 denkbar sind.




**Das Tag besitzt die Attribute breite und hoehe**

Schreibe das Beispiel ab, wenn du möchtest. Speichere es unter dem Namen *grafik.xml*. Setze außerdem einen Link (grau hinterlegt) auf die noch zu erstellende DTD namens *grafik.dtd* ein:

```
<?xml version="1.0" encoding=
"iso-8859-1"?>
<!DOCTYPE grafik SYSTEM "grafik.dtd">
<grafik>
    <bild breite="100"
    hoehe="60">rose.tif</bild>
</grafik>
```

### Die DTD zur Attributdefinition

Schauen wir uns nun die dazugehörige DTD an. Schreibe das Beispiel ab und speichere unter dem Namen *grafik.dtd*.

```
<!ELEMENT grafik (bild)>
<!ELEMENT bild (#PCDATA)>
<!ATTLIST bild
breite CDATA #REQUIRED
hoehe CDATA #REQUIRED>
```

Besonders interessant sind für uns die letzten drei Zeilen, eigentlich ist es ja nur eine Zeile.

- Aus Gründen der Übersichtlichkeit habe ich Zeilenumbrüche eingefügt: So steht jede Attributdefinition auf einer eigenen Zeile.
- Hinter *!ATTLIST* wird zuerst der Tag-Name angegeben, hier *bild*.
- Statt der runden Klammern bei fest vorgegebenen Werten notieren wir lediglich CDATA, die Abkürzung für character data.

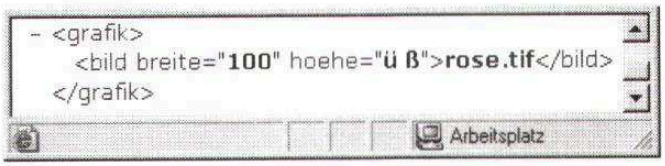
CDATA bedeutet, dass der Wert eine beliebige *Zeichenkette* sein kann.

Teste durch Validieren! Nimm ein Attribut aus dem <*bild*>-Tag heraus. Da beide Attribute „required" sind, erscheint eine Fehlermeldung.

## Vor und Nachteile von CDATA

Reden wir noch einmal kurz über die CDATA-Anweisung. Auf diese Weise haben wir ein Attribut erstellt, welches beliebige Werte annehmen kann. Das ist ein Vorteil.

Auch Leer- und Sonderzeichen sind dabei erlaubt.




**Quatsch mit Soße – aber gültig!**

Wie du siehst, wird auch dieser blödsinnige Attributwert klaglos akzeptiert, hält sogar einer Validierung stand.

## Schlüsselwort NMTOKEN

Wenn du Sonderzeichen und Leerzeichen ausschließen möchtest, greifst du statt dessen auf das Schlüsselwort NMTOKEN zurück. Statt

```
breite CDATA #REQUIRED
```

schreibst du dann

```
breite NMTOKEN #REQUIRED
```

Jetzt sind nur noch folgende Zeichen erlaubt:

- Buchstaben
- Ziffern
- Punkt (.), Doppelpunkt (:)
- Bindestrich (-), Unterstrich (_)

Leider gibt es keine „elegante" Möglichkeit, die Anzeige nur auf Zahlen oder auf Zahlen zwischen beispielsweise 10 und 260 zu begrenzen. Du müsstest dann alle diese Werte in runden Klammern vorgeben.

Spätestens an dieser Stelle zeigen sich die Nachteile der Verwendung des DTD-Konzepts als „Musterschablone". Denn DTDs stammen noch aus alten „SGML-Tagen". Deshalb hat man erst 2001 ein neues, ziemlich komplexes Prinzip namens Schema eingeführt. Aber auch in dieses Konzept führe ich dich ein, und zwar ab Seite 69.

An dieser Stelle wollen wir uns jedoch weiter mit den DTDs „herumärgern", da sie momentan noch „Stand der Technik" sind.

## 2. #IMPLIED: Attribut ist optional

Erweitern wir die DTD ein wenig. Ich füge ein weiteres Attribut names *lzw* ein. (Das ist ein Kompressionsverfahren für Tif-Grafiken). Dieses soll die Werte *yes* oder *no* annehmen können, je nachdem ob die Grafik komprimiert ist oder nicht.

An diesem Beispiel zeige ich dir, wie du mit dem schon erwähnten Schlüsselwort #IMPLIED dafür sorgst, dass das Attribut optional ist. Die entsprechende Zeile habe ich grau hinterlegt.

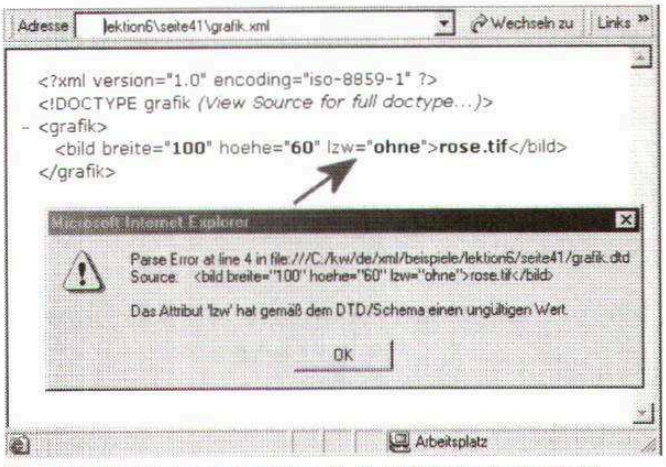
```
<!ELEMENT grafik (bild)>
<!ELEMENT bild (#PCDATA)>
<!ATTLIST bild
breite CDATA #REQUIRED
hoehe CDATA #REQUIRED
lzw (yes|no) #IMPLIED>
```

## Übung macht den Meister!

*Frage*: Entspricht die bisherige XML-Datei (siehe Vorseite) immer noch dem „Schema" der DTD? Wenn ja, warum? Validiere!

*Frage*: Doch was passiert, wenn ich in das Tag *<bild>* das Attribut-Werte-Paar *lzw="ohne"* einfüge? Probiere es aus! Validiere auch hier.

Warum gibt es eine Fehlermeldung?




**Das Attribut ist zwar optional, der Wert jedoch nicht**

*Antworten:* Im ersten Fall entspricht die XML-Datei immer noch der „Schablone" aus der DTD. Schließlich war das Attribut nur „implied", optional. Im zweiten Fall entspricht die XML-Datei nicht mehr der DTD, da das Attribut einen falschen Wert besitzt.

## 3. Vorgabe: Wert voreinstellen

Zurück zu den Attributen. Du weißt, wie du Attribute pflichtweise oder fakultativ vorgibst, wie du auf Wunsch die entsprechenden Werte festlegst oder mit CDATA „alles offen lässt".

Du weißt aber noch nicht, wie du ein Attribut nebst Wert vorgibst. Wir wollen das Grafik-Beispiel so ändern, dass das Attribut-Werte-Paar *lzw="yes"* vorgegeben wird.

Dazu änderst du die Zeile

```
lzw (yes|no) #IMPLIED>
```

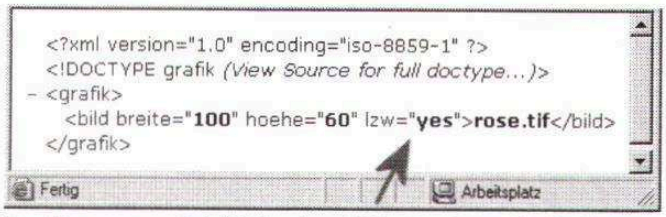
wie folgt:

```
lzw (yes|no) "yes">
```

Du notierst den Vorgabewert einfach an Stelle von #IMPLIED bzw. #REQUIRED. Sobald du mit einen Vorgabewert arbeitest, darfst du diese Schlüsselwörter nicht mehr verwenden!

**Setze den Vorgabewert in Gänsefüßchen!**

Was passiert nun? Probiere es aus! Lösche in der XML-Datei einfach das Attribut *lzw*.




**Der Browser setzt Attribut und Wert automatisch ein**

Der Internet Explorer setzt nun das Attribut mitsamt vorgegebenem Wert ganz automatisch ein.

Doch was passiert, wenn du statt *yes* außer der Reihe den Wert *no* haben möchtest? Dann notiere einfach *lzw="no"*.

**Der von dir eingegebene Wert überschreibt den Vorgabewert!**

## 4. FIXED: Vorgabe festschreiben

Du kannst diese Vorgabe jedoch auch „fest einstellen". Dazu notierst du hinter der Angabe des Attributwerts bzw. der Attributwerte erst das Schlüsselwort

- `#FIXED`

und schreibst dahinter den fest vorgegebenen Wert in Gänsefüßchen. Für unser Beispiel wäre also folgende Änderung durchaus korrekt:

```
lzw (yes|no) #FIXED "yes">
```

Doch denke einmal nach. Macht es Sinn, einerseits *yes* und *no* zur Auswahl zu stellen und dann als „fixed" Wert nur *yes* zuzulassen?

Noch klarer ist deshalb folgende Zeile.

```
lzw CDATA #FIXED "yes">
```

Die vollständige DTD sieht für unser Beispiel so aus:

```
<!ELEMENT grafik (bild)>
<!ELEMENT bild (#PCDATA)>
<!ATTLIST bild
breite CDATA #REQUIRED
hoehe CDATA #REQUIRED
lzw CDATA #FIXED "yes">
```

Probiere immer wieder aus, ob das Ergebnis immer noch gültig ist.


**Beruhigend: Validation Successful**

# Lektion 7: Entitäten als „Platzhalter“ nutzen

In dieser Lektion lernst du folgendes:

- 1. Entitäten als interne Platzhalter nutzen
- 2. externe Dateien durch Entitäten einbinden
- 3. „Parameter-Entities“ für die DTD

Erinnerst du dich an unsere Betrachtungen der Entitäten? Auf Seite 23 hattest du Entitäten als eine Art „Platzhalter“ für Sonderzeichen kennen gelernt.

Nun zeige ich dir, wie man diese „Entities“ als Platzhalter für beliebigen Text verwenden kann. Zuerst geht es um verschiedene „interne Entitäten“. Diese wirken vor allem in der XML-Datei. Danach schauen wir uns die Kandidaten an, die nur in der DTD zum Einsatz kommen.

## 1. Mehr Komfort: Interne Entitäten

Womit kannst du Entitäten vergleichen? Vielleicht mit dem Prinzip des Seriendrucks! Da gibt es *eine* „Briefvorlage“, die für hunderte Empfänger gilt. An ganz speziellen Stellen arbeiten Seriendruckfelder und fügen beispielsweise den jeweiligen Namen ein.

Programmierer kennen ein ähnliches Konzept namens Variable. Eine Variable ist eine Art Platzhalter für eine bestimmte Zeichenfolge. Man muss die Variable nur einmal deklarieren. Der Inhalt kann beliebig sein.

### XML-Datei für einen Brief

Als Beispiel schreibst du eine XML-Datei für einen Brief. Nenne sie *brief.xml*. Binde gleich die noch zu erstellende DTD *brief.dtd* ein.

```
<?xml version="1.0" encoding=
"iso-8859-1"?>
<!DOCTYPE brief SYSTEM "brief.dtd">
<brief>
    <an>Peter</an>
    <betreff>Konzertbesuch</betreff>
    <text>Hallo Peter, wann treffen
    wir uns?</text>
</brief>
```

Bei diesem Beispiel stellst du fest, dass der Empfänger *Peter* immer wieder auftaucht.

### DTD für den Brief erstellen

Als nächstes erstellst du die DTD *brief.dtd* für den Brief. Versuche es selber und vergleiche:

```
<!ELEMENT brief (an,betreff,text)>
<!ELEMENT an (#PCDATA)>
<!ELEMENT betreff (#PCDATA)>
<!ELEMENT text (#PCDATA)>
```

Wo steckt die Entität? Die fügen wir gleich ein!

### Entität einfügen

Im Beispiel arbeiten wir mit einer Entität namens `&name;`. An allen Stellen wo *Peter* steht, soll `&name;` notiert werden. Denn nur so kann die XML-Datei als Vorlage für viele verschiedene Briefe an mehrere Empfänger dienen!

**Eine Entität beginnt mit & und endet immer mit einem Semikolon ;.**

Bisher kassierst du nur eine Fehlermeldung. Der Internet Explorer beschwert sich über den „Verweis auf eine nicht definierte Entität“.

### Entität definieren

Das Definieren einer Entität geschieht ganz einfach in der DTD, die Syntax sieht so aus.

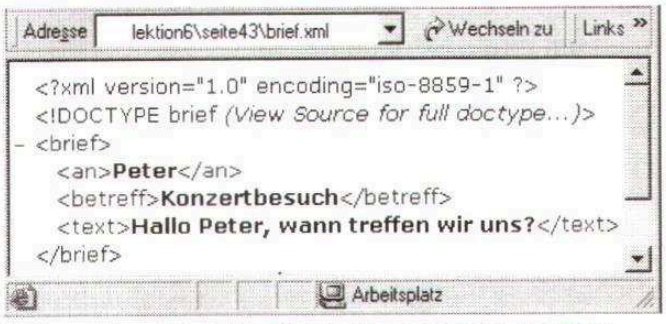
```
<!ENTITY entityname "inhalt">
```

Der Platzhalter *entityname* steht für den Namen der Entität. Lasse hier jedoch **&** und **;** weg!

**Setze die Definition der Entität am besten unterhalb der Tag-Definition ein.**

Notiere nun folgende Zeile am Ende der DTD:

```
<!ENTITY name "Peter">
```

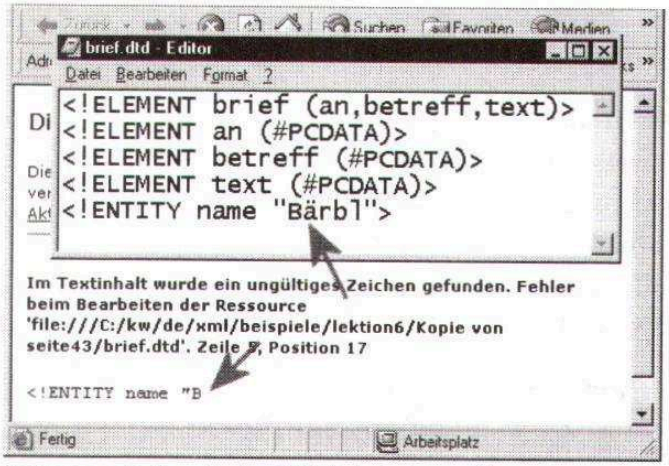



**Hokuspokus: Statt der Platzhalter erscheint der Text**

## Die Tücken der Praxis: Umlaute?

Gar nicht schlecht, dieses Konzept der Entities. Doch die Browser-Praxis macht uns wieder einen Strich durch die Rechnung. Sobald du beim Inhalt der Entität mit Umlauten oder Sonderzeichen arbeitest, beginnen die Probleme.

Kurz: Ein Name wie *Bärbl* wäre tödlich für die Anzeige. Es kommt zur Fehlermeldung.




**Schluss mit der „Umlaut-Diskriminierung“**

Die Lösung: Füge am Anfang der DTD ausnahmsweise einen XML-Prolog ein, in welchem du per `encoding="iso-8859-1"` unseren Latin-1-Zeichensatz festlegst.

## 2. Geht auch: Externe Dateien

Mit diesen raffinierten Entitäten kannst du auch externe (XML-)Dateien einbinden. Vorab die Syntax:

```
<!ENTITY entityname SYSTEM "datei.end">
```

Dabei steht *entityname* für den Namen der Entität. Danach notierst du ein Leerzeichen und das Schlüsselwort SYSTEM. Nach einem weiteren Leerzeichen gibst du in Klammern den Namen mit Endung (und ggf. den Pfad) der Datei an.

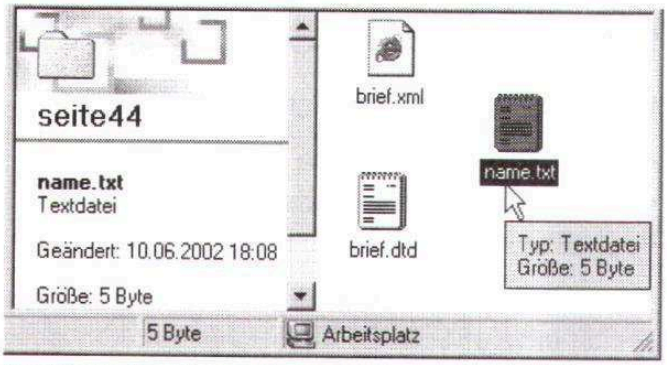
Dabei kann es sich um eine XML-Datei, aber auch um eine Textdatei handeln.

### Die DTD im Überblick

Probieren wir es einfach aus. Im Beispiel arbeiten wir mit einer Textdatei.

- Die XML-Datei bleibt unverändert bestehen
- Der Name wird nun in der Textdatei *name.txt* gespeichert

Dafür muss lediglich die DTD geändert werden.




**Speichere den Namen in einer externen Textdatei**

Und so sieht der leicht veränderte Quelltext der DTD aus:

```
<!ELEMENT brief (an,betreff,text)>
<!ELEMENT an (#PCDATA)>
<!ELEMENT betreff (#PCDATA)>
<!ELEMENT text (#PCDATA)>
<!ENTITY name SYSTEM "name.txt">
```

Vergleiche mit den Dateien aus den betreffenden Ordner unter *lektion7*. (www.jchanke.de/xml)

Allerdings besteht hier wieder das „Umlautproblem“. Abgesehen davon kann es mit dieser Art der „externen Entitäten“ derzeit noch zu Problemen kommen.

## 3. Parameter-Entities in der DTD

Last but not least zeige ich dir die dritte „Gattung“ der Entitäten. Du kannst auch innerhalb einer DTD mit Platzhaltern arbeiten.

Diese so genannten Parameter-Entities machen vor allem dann Sinn, wenn du mit langen DTDs arbeitest und dir das häufige Tippen immer gleich lautender Abschnitte ersparen willst.

### Beispiel Inventarliste

Im Beispiel habe ich mir eine Inventarliste ausgedacht. In einem Dokument soll die Anzahl verschiedener Büromöbel aufgezeigt werden. Dafür verwende ich die Elemente

- tisch
- stuhl
- schrank usw. usf.

Jedes Tag soll das Attribut *marke* besitzen, welches als Werte verschiedene vorgegebene Marken zuweisen soll. Und hier beginnt die Erleichterung bei der Schreibarbeit ...

## Die XML-Inventarliste im Überblick

Zuerst zeige ich dir die XML-Inventarliste. Wie du siehst ist die Liste ganz einfach aufgebaut:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE inventar SYSTEM "inventar.dtd">
<inventar>
    <tisch marke="Desktop">2</tisch>
    <stuhl marke="Orkus">3</stuhl>
    <schrank marke="Troll">1</schrank>
    <schrank marke="Orkus">1</schrank>
    <regal marke="Ikea">3</regal>
</inventar>
```

Erst die DTD wird für uns wirklich interessant. Hier lege ich fest, dass die vier Tags in beliebiger Reihenfolge verwendet werden dürfen (Oder-Verknüpfung). Sie sollen beliebig oft auftreten können – wer weiß wie viele Schränke es in deinem Büro gibt – aber nicht müssen. Das erreichst du mit dem Sternchen-Operator. Schau nach unten zur DTD, zur zweiten Zeile: Hier habe ich das Sternchen diesmal außerhalb der runden Klammern notiert, damit es gleich auf die ganze Gruppe wirkt.

## Syntax der Parameter-Entitäten

Nutzen wir endlich die Vorteile der Parameter-Entitäten. Sprechen wir deshalb zuerst über die Syntax!

**Eine Parameter-Entity wird nicht von den Zeichen & ; sondern von % ; eingehüllt. Außerdem muss diese Entität – Variablen lassen grüßen – ganz zu Beginn der DTD notiert werden.**

Erzeugt wird eine Parameter-Entity in folgender Syntax: `<!ENTITY % entityname "Inhalt">`. Beachte die Leerzeichen zwischen dem Prozentzeichen, dem Namen der Entität und dem eigentlichen Inhalt. Zugreifen auf den Inhalt der Entität kannst du dann durch `%entityname;` wobei sich Prozentzeichen und das „Ende-Semikolon“ ohne Leerzeichen an den Namen „herankuscheln müssen“.

## Das Beispiel DTD

Im Beispiel soll die Entität *attribut* heißen. Der dort gespeicherte Inhalt besteht aus dem Attributnamen *marke* mit den Werten *Desktop, Orkus, Troll, Ikea*. Diese sollen „required“ sein. Also setze ich für *Inhalt* folgende Zeile ein: *marke (Desktop|Orkus|Troll|Ikea) #REQUIRED*. Beachte, dass der *Inhalt* unbedingt von Gänsefüßchen eingehüllt werden muss.

Hier zeige ich dir nun endlich das Skript der Datei *inventar.dtd*. Die Zeile mit der Entity-Definition habe ich grau hinterlegt, die Entitäten selber dagegen fett formatiert:

```
<!ENTITY % attribut "marke (Desktop|Orkus|Troll|Ikea) #REQUIRED">
<!ELEMENT inventar (tisch|stuhl|schrank|regal)*>
<!ELEMENT tisch (#PCDATA)>
<!ATTLIST tisch %attribut;>
<!ELEMENT stuhl (#PCDATA)>
<!ATTLIST stuhl %attribut;>
<!ELEMENT schrank (#PCDATA)>
<!ATTLIST schrank %attribut;>
<!ELEMENT regal (#PCDATA)>
<!ATTLIST regal %attribut;>
```

Dieses Beispiel zeigt, dass du dir mit Parameter-Entities durchaus einige Schreibarbeit sparen kannst. Vergleiche mit den Dateien aus meinem entsprechend benannten Unterordner unter *lektion7*.

# Übungsteil C: Attribute, DTD und Entitäten

Du weißt jetzt auch, wie man:

- Attribute verwendet und in der DTD definiert
- Attributwerte vordefiniert und festschreibt
- Entitäten als Platzhalter verwendet

Wie wäre es mit einer Runde „Gehirn-Jogging"? XML macht Spaß und Übung den Meister!

## Übung C1: XML-Dokument mit Attributen

Schreibe folgendes XML-Dokument ab und speichere es unter dem Namen *bauwerke.xml*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bauwerke>
    <bauwerk stil="Gotik">Erfurter Dom</bauwerk>
    <bauwerk stil="Romanik">Stiftskirche Quedlinburg</bauwerk>
    <bauwerk stil="Klassizismus">Schauspielhaus Berlin</bauwerk>
    <bauwerk>Palast der Republik Berlin</bauwerk>
    <bauwerk stil="Jugendstil">Stadttheater Cottbus</bauwerk>
</bauwerke>
```

## Übung C2: Dokumenttyp-Definition schreiben

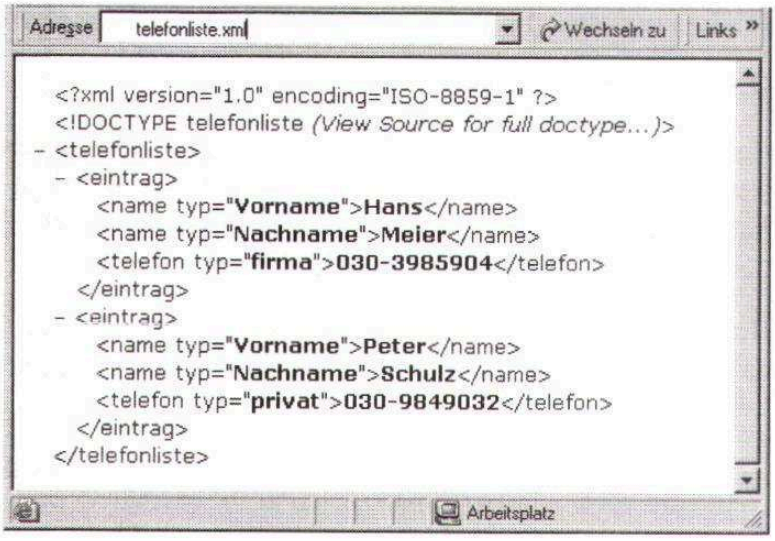
Erstelle für dieses XML-Dokument nun die entsprechende Dokumenttyp-Definition, nenne diese *bauwerke.dtd*. Berücksichtige dabei die Attribute, die Reihenfolge ist egal. Denke nach, sind diese Attribute immer Pflicht? Berücksichtige das in der DTD! Vergiss nicht den Verweis auf die DTD!

## Übung C3: Attributwerte vorgeben

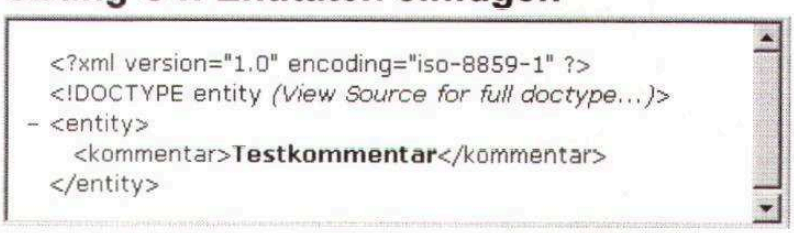
Erstelle diese Telefonliste namens *telefonliste.xml*. Richte eine DTD ein. Diese soll folgende Merkmale besitzen:

Das Tag *name* soll ein- oder mehrmals vorkommen, das Tag *telefon* dagegen nur einmal.

Das Tag *name* soll obligatorisch entweder das Attribut *typ="Vorname"* oder *typ="Nachname"* besitzen. Das Tag *telefon* soll das Attribut *typ="firma"* bzw. *typ="privat"* erhalten. Dabei soll *typ="firma"* voreingestellt, aber überschreibbar sein.




## Übung C4: Entitäten einfügen




Erstelle folgende Datei *entity.xml* nebst DTD *entity.dtd*. Für den Inhalt des Tags *kommentar* soll die Entität *&comment;* sorgen. Definiere die Entität in der DTD und füge sie in die XML-Datei ein.

# Lektion 8: Schachtelungen und Klammerspiele in der DTD

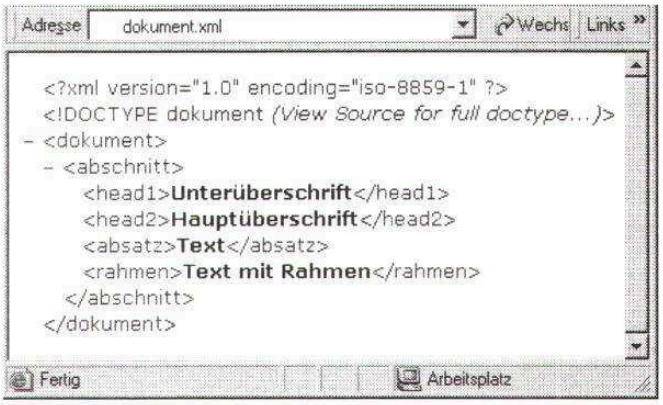
In dieser Lektion lernst du folgendes:

- Wiederholungsoperatoren richtig einsetzen
- Zusammenhang verstehen zwischen Elementreihenfolge und Gruppierung

Zuerst geht es um Klarstellung und Festigung des vorhandenen Wissens. Dieses Kapitel trägt wieder Workshopcharakter. Mach mit!

## Eine Dokumentstruktur darstellen

Machen wir ein Experiment! Dafür schreibst du dieses Beispiel ab und speicherst es unter dem Namen *dokument.xml*. Es handelt sich um die Grundstruktur eines Dokumentes, beispielsweise einer Webseite. Wir gliedern den Text in zwei Überschriften, einen Absatz und einen Rahmen.




**Verlinke auf die noch zu erstellende DTD**

Setze gleich einen Link auf die noch zu erstellende DTD namens *dokument.dtd*.

## Die dazugehörige DTD dokument.dtd

Notiere nun die dazugehörige DTD. Wir werden an diesem Beispiel experimentieren!

```
<!ELEMENT dokument (abschnitt+)>
<!ELEMENT abschnitt
(head1,head2,absatz,rahmen)>
<!ELEMENT head1 (#PCDATA)>
<!ELEMENT head2 (#PCDATA)>
<!ELEMENT absatz (#PCDATA)>
<!ELEMENT rahmen (#PCDATA)>
```

Besonderes Augenmerk legen wir auf den Bereich (es ist eine Zeile!), den ich grau hinterlegt habe. Hier wird schließlich durch die runden Klammern *erstens* eine Gruppe erstellt und *zweitens* die Reihenfolge genau festgelegt.

## Gruppen und Operatoren

Wenn du validierst stellst du fest – alles perfekt. Das XML-Dokument ist in dieser Fassung gültig, die Reihenfolge und Anordnung stimmt: Vergleiche noch einmal mit dieser Passage:

```
(head1,head2,absatz,rahmen)
```

Es gibt je ein Element namens *head1*, *head2*, *absatz* und *rahmen*. Die festgelegte Reihenfolge wurde auch eingehalten. Zur Erinnerung: Man spricht hier von der *Und*-Verknüpfung.

## Oder-Verknüpfung im Detail

Wenn du die Reihenfolge der Tags im XML-Dokument beliebig festlegen möchtest? Auch kein Problem. Dann ersetzt du das Komma einfach durch den senkrechten Strich. Dieses Zeichen steht schließlich für eine Oder-Verknüpfung. Probiere erst einmal das hier:

```
(head1|head2|absatz|rahmen)
```

Oder-Verknüpfung bedeutet, dass du den einen *oder* den anderen Wert verwenden kannst.

Bis jetzt liest sich diese Anweisung so:

- Nimm das Element *head1*
- *oder* head2 *oder* absatz *oder* rahmen

Der Haken: Du kannst so nur ein einziges Element verwenden. Würdest du das nebenstehende XML-Dokument gegen die veränderte DTD validieren, gäbe es eine Fehlermeldung.

## Wiederholungs-Operator Plus (+)

Notiere deshalb den Wiederholungs-Operator Plus, und zwar *außerhalb der schließenden geschweiften Klammer*.

```
(head1|head2|absatz|rahmen)+
```

Dieser wirkt auf die ganze Gruppe. Das Plus bedeutet, dass das Element einmal oder mehrmals vorhanden sein muss. Mit Element ist nun die gesamte „Gruppenanweisung" gemeint.

- Eins der vier Elemente muss auf jeden Fall auftreten, egal welches.
- Es kann eine beliebige Anzahl weiterer Elemente vorkommen.

Jetzt könntest du im XML-Dokument gerne zuerst einen *absatz* notieren, dann *head2*, dann *head1* usw. Und zwar in beliebiger Reihenfolge und beliebiger Häufigkeit. Vergleiche mit meinem Beispiel *dokument2.xml* mit der DTD *dokument2.dtd*, wo ich genau das gemacht habe.

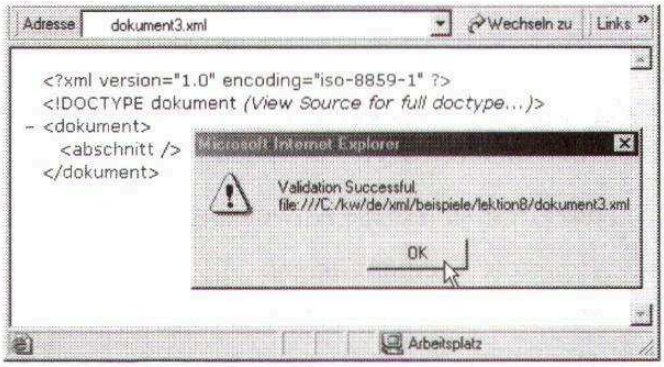
Du könntest jedoch auch alle Elemente bis auf eines weglassen. Eins muss mindestens bleiben.

## Operator Sternchen (*)

Du möchtest völlig freie Hand bekommen? Du willst die einzelnen Tags so oft und so durcheinander verwenden wie du möchtest? Du wünschst dir jedoch auch die Option, völlig auf die Tags verzichten zu können? Dann nimm den Sternchen-Operator:

```
(head1|head2|absatz|rahmen)*
```

Dieses Beispiel findest du im Dokument *dokument3.xml*.




**Validation successful trotz weggelassener Tags!**

Bei allen diesen Beispielen wird der Operator außerhalb der runden Klammern notiert.

## Und-Verknüpfung und Gruppe

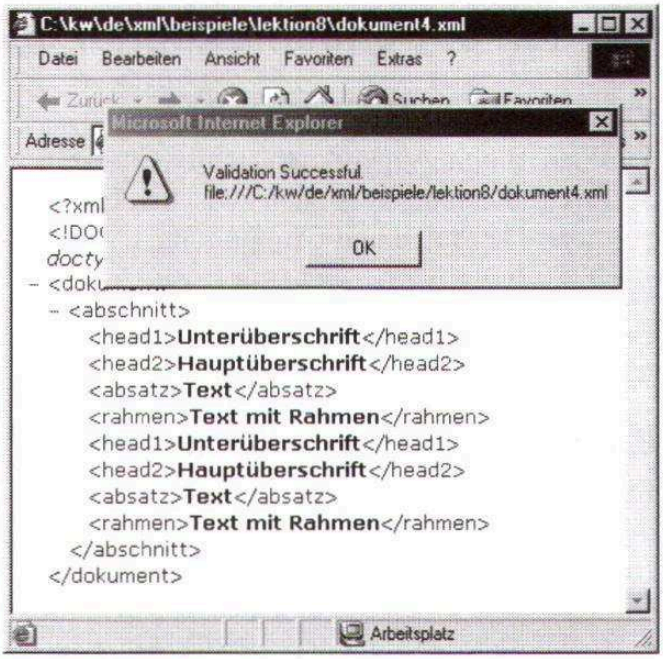
Experimentieren wir weiter! Was würdest du mit dieser Anweisung erreichen?

```
(head1,head2,absatz,rahmen)+
```

Könntest du die Tags jetzt auch beliebig anordnen? Nein! Du musst sie nun unbedingt in folgender Reihenfolge auflisten:

- *head1 - head2 - absatz - rahmen*

Du kannst sie zwar beliebig oft aufreihen, jedoch immer wieder nur in dieser Reihenfolge!




**Die Und-Verknüpfung legt die Reihenfolge fest**

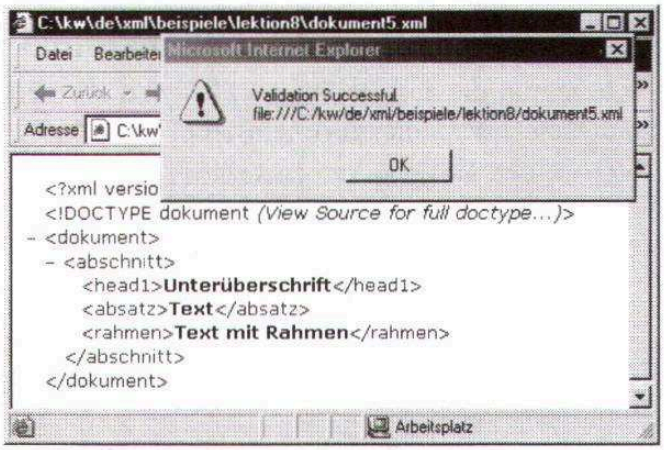
Das Beispiel findest du in der Datei *dokument4.xml*.

## Untergruppen erstellen

Das Spielchen lässt sich übrigens noch weitertreiben. Erstelle auf Wunsch sogar Untergruppen! Schau dir einmal diese Option an:

```
(head1,(head2|absatz),rahmen)+
```

Was erreichst du damit? Richtig! Nach *head1* muss entweder *head2* ***oder*** *absatz* folgen. Danach ist *rahmen* zwingend vorgeschrieben. Genau diese Reihenfolge kann nun wieder beliebig oft wiederholt werden.




**Lasse entweder head2 oder absatz weg!**

Du findest das Beispiel in der Datei *dokument5.xml* mit der entsprechenden DTD.

# Lektion 9: Von Namensräumen und Dateninseln

In dieser Lektion lernst du folgendes:

- Namensräume verstehen, definieren und deklarieren
- HTML-Tags in XML einbinden
- Dateninseln am Beispiel XML in HTML

Gerade das Konzept der Namensräume ist eine wichtige Grundlage für das Verständnis vieler auf XML basierender Sprachen. Sei gespannt!

## Räume für Tags: name spaces

Was hatte ich dir am Anfang vorgeschwärmt? In XML kann man sich seine Tags selber aussuchen! Und genau so ist es auch.

### Problem der Doppeldeutigkeit

Doch wenn du zwei XML-Dokumente von unterschiedlichen Autoren zusammenführen möchtest? Und sich diese Autoren nicht abgesprochen haben?

Angenommen, der Hersteller verwendet ein Tag namens *<produkt></produkt>* und meint damit die Anzahl des Produkt. Der Vertrieb verwendet das gleiche Tag und will dieses als „Sinnträger" für eine Produktbeschreibung verstanden wissen.

Heraus kommt das herrlichste Chaos!

### Doppeldeutigkeiten reparieren

Im richtigen Leben können wir diese „Doppeldeutigkeiten" leicht reparieren. Nehmen wir als weiteres Beispiel einfach Kollegen mit gleichem Vornamen. Du unterscheidest sicher mühelos zwischen Hans von der Versandabteilung und dem „Langweiler" Hans aus der Buchhaltung.

Ähnlich einfach bei den Produkten. So sieht der Leser schon aus dem Inhalt des Tags *<produkt>*, dass das eine Mal die Anzahl und das andere Mal die Beschreibung gemeint sein muss!

XML wird jedoch vorrangig von Maschinen ausgelesen. Maschinen sind nicht so intelligent.

Eine eindeutige Zuordnung muss her!

### Namensraum als „Raum für Namen"

Das haben auch die Leute vom World Wide Web Consortium gemerkt und 1999 schnell noch die Namensräume erfunden.

Namensräume heißen auf Englisch name spaces.

Namensräume sind, wie der Name schon sagt, „Räume für Namen". Weise jedem der beiden XML-Dokumente einen eigenen Raum zu. Verwende den einen „Hans" im „Kontext" von Versand, den anderen im Sinne von Buchhaltung. Für diesen Kontext denken wir uns ein „Kürzel" aus. Den Versand-Typen beschreiben wir so:

```
<versand:name>Hans</versand:name>
```

Der „Buchhalter"-Kollege kann mit

```
<buch:name>Hans</buch:name>
```

definiert werden. Auf ähnliche Weise lassen sich auch unsere beiden gleichen *<produkt>*-Tags durch unterschiedliche Namensräume unterscheiden. Das eine Tag wird so angesprochen:

```
<hst:produkt>2</hst:produkt>
```

Das zweite dagegen so:

```
<vtb:produkt>Stuhl</vtb:produkt>
```

### Namensraum-Präfix: Syntax der Tags

Und schon kennst du die recht simple gestrickte Syntax für die Namesraum-Definition!

Das Element vor dem Doppelpunkt (unser Kürzel) nennt sich Namensraum-Präfix.

Den Präfix-Namen kannst du dir frei ausdenken, solange du auf Leer- und Sonderzeichen verzichtest und nicht mit einer Zahl beginnst.

Im „Hans-Beispiel" heißen die beiden Namensräume *versand* bzw. *buch*, im „Produkt-Beispiel" *hst* und *vtb*. In der Kürze liegt hier die Würze, denn kurze Präfixe sparen Schreiberei!

Danach folgen der eben erwähnte Doppelpunkt und das gewöhnliche Tag. So weit, so gut. Doch wie deklariert man einen Namensraum?

## So deklarierst du einen Namensraum

Du weißt was Namensräume sind und wie du gleichlautende Tags durch unterschiedliche Präfixe und Doppelpunkt „in ihren jeweiligen Raum sperrst". Doch wie erschaffst du diese Räume?


**Hansdampf in allen Gassen: Es sind verschiedene Personen in „verschiedenen Räumen"**

Das gelingt in einem „übergeordneten Tag". Sinnvollerweise nehmen wir das Wurzelelement. Erstelle also im Wurzelelement eine Deklaration, in welcher das jeweilige Präfix vereinbart wird. Im Hans-Beispiel deklarieren wir gleich zwei Namensräume auf einen Schlag:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<firma xmlns:versand="http://www.jchanke.de/versand"
   xmlns:buch="http://www.jchanke.de/buch">
   <personen>
      <versand:name>Hans</versand:name>
      <buch:name>Hans</buch:name>
   </personen>
</firma>
```

Du findest das Beispiel im Ordner *lektion9* in der Datei *namensraum.xml*.

## Das Attribut xmlns und der URI

Schau dir die hervorgehobene Zeile an: Hier werden die Namensräume deklariert. Das Attribut *xmlns* sorgt dabei für die Referenz auf den jeweiligen Namensraum. Das Attribut kommt im Beispiel zweimal vor, da es gleich um zwei Namensräume geht. (In der Praxis geht es oft nur um einen einzigen.)

Mit xmlns kürzt man die Wortgruppe *XML name space* ab!

Der eigentliche Namensraum wird nun im Beispiel durch eine Web-Adresse „referenziert". Doch was bedeutet diese Web-Adresse? Was bedeutet dieser URI, wie man im Fachjargon sagt, der Uniform Ressource Identifier (URI=weltweit eindeutiger Bezeichner)?

## Die mysteriöse Web-Adresse

Halt, stopp – zurück! Du brauchst diese Adresse nicht in deinen Browser einzutippen, du findest dort nichts! Keine Sorge, auch der Browser wird sich beim Anzeigen der Datei nicht mit dem Web verbinden. Diese Web-Adresse alias URI dient einfach nur als weltweit eindeutiger Bezeichner, wie der Name schon sagt. Da es eine bestimmte Web-Adresse nur ein einziges Mal geben kann, greift man mit Vorliebe auf eine Web-Adresse zurück. In diesem Beispiel darfst du dir den URI selber ausdenken und musst diese Benennung dann allerdings konsequent beibehalten.

Bei schon definierten Namensräumen (z.B. zur Darstellung von HTML-Tags in XML) ist diese Adresse jedoch fest vorgegeben. Dann musst du den einmal festgelegten URI verwenden.

Keine Angst, wir werden keine weiteren XML-Dokumente zusammenführen. Die Beispiele dienten als Muster für die Deklaration von Namensräumen und als Lernschritte für das Verständnis schon vordefinierter Namensräume. Und nur diese werden uns im weiteren Kursverlauf interessieren.

## HTML-Datei in XML einbinden

Vordefinierte Namensraumbeispiele, die Erste: Binde doch einfach eine HTML-Datei in XML ein. Wie das geht? Mit einem *name space*, einem „eigenen Zimmer" für die artfremden HTML-Tags.

**Dieser Namensraum wurde vom W3C fest vorgegeben.**

Das Tag mit Namensraum-Deklaration – in der Regel das Wurzelelement – muss zwingend so aussehen:

```
<Tag xmlns:html="http://www.w3.org/TR/REC-html40">
```

Die Angabe der Web-Adresse in Gänsefüßchen ist im Browser fest verankert. Da das „Zimmer" vom W3C vordefiniert wurde, klappt es nur dann, wenn du genau diese Adresse verwendest. Vor allem Netscape 6.x wird beim kleinsten Verschreiber „herumzicken". Hier ein simples Beispiel:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<seite xmlns:html="http://www.w3.org/TR/REC-html40">
    <xmltag>Ich gehöre noch keinem Namensraum an.</xmltag>
    <html:h1>Ahoi HTML!</html:h1>
    <html:h2>Eine echte Überschrift 2!</html:h2>
    <html:marquee>Unfassbar, sogar Lauftext!</html:marquee>
    <html:p>Ich bin ein Satz in einem HTML-Absatz.</html:p>
    <html:hr/>
</seite>
```

Jetzt kannst du, wie du siehst, XML mit HTML mischen. Verwende beliebige Tags aus HTML. Selbst das berühmt-berüchtigte *<marquee></marquee>* wird zumindest im Internet Explorer korrekt angezeigt. Allerdings musst du dich an die strengen XML-Regeln halten. Jedes Tag benötigt ein End-Tag!

### Linie intern abschalten!

Wunderst du dich über die Linie? Warum schreibe ich *<html:hr/>* statt *<html:hr>*? Das liegt daran, dass es keine leeren Tags in XML geben darf. In HTML gibt es diese jedoch, denke an <hr>, <img>, <input> usw. Damit es keine Probleme gibt, schaltet man solche Tags intern ab. Man setzt zusätzlich einen Slash an das Ende des Tags.

### Browservorschau

Rufe das Dokument im Netscape-Browser oder in Opera auf. Du siehst die HTML-Struktur. Selbst das oberhalb des „HTML-Bereichs" notierte reine XML-Tag erscheint, jedoch unformatiert.


**Nur der Internet Explorer stellt den Lauftext dar**

Apropos **Internet Explorer**. Bei dir ist bisher noch die Baumstruktur sichtbar? Der Internet Explorer lässt sich nur mit einen Trick zur „korrekten" Anzeige des HTML-Dokuments überreden. Füge einfach einen Verweis auf eine Layoutdatei ein, binde also eine so genannte CSS-Datei ein. Setze also noch diese Zeile an zweite Position, genau unter den Prolog und über das Wurzelelement:

```
<?xml-stylesheet href="dummy.css" type="text/css"?>
```

Der Clou bei der Geschichte: Die eigentliche Layout-Datei muss dafür überhaupt nicht vorhanden sein! Vergleiche mit der Datei *htmlinhtml.xml* aus dem Ordner *lektion9*. Eine Einführung in das Gestalten von XML-Dokumenten mit CSS bekommst du ab Seite 58.

## Von Hiddensee nach Sylt: Dateninseln in HTML

Der Vollständigkeit halber erwähne ich noch den umgedrehten Weg: Du kannst auch XML-Code in HTML-Dokumente „einbauen". Dafür erzeugst du lediglich eine so genannte Dateninsel.

Diese Insel beginnt mit *<xml>* und endet mit *</xml>* Vergib eine selbst wählbare Kennung. Nutze dafür das Attribut *id*.

Das Attribut id steht als Abkürzung für identifier, dahinter verbirgt sich eine eindeutige Kennung. Im Beispiel habe ich mir die id *xmlinsel* ausgedacht.


Hier ein Beispiel:

```
<html>
<head>
   <title>Dateninsel</title>
</head>
<body>
   <h1>XML-Dokument in HTML einbetten</h1>
      <xml id="xmlinsel">
            <posten>
                  <name>KnowWare</name>
                  <preis>4</preis>
            </posten>
      </xml>
   <p>Welcome back to HTML!</p>
</body>
</html>
```

Ohne Layoutanweisungen (CSS-Datei) ist nur der Netscape-Browser 6.x zur Anzeige der XML-Daten fähig. Wenn du das Beispiel im Internet Explorer aufrufst, wird der XML-Teil „unterdrückt", obwohl er natürlich vorhanden ist. Probiere es aus!

Mein Musterdokument heißt *dateninsel.html* und befindet sich im Ordner *lektion9*.

# Lektion 10: HTML lebt: XHTML als Neufassung von HTML

In dieser Lektion lernst du folgendes:

- Eigenschaften von XHTML
- Aufbau eines XHTML-Dokuments
- DTDs von XHTML
- XHTML in der Praxis

## XHTML als Neufassung von HTML

Bisher hatten wir festgestellt, dass XML für das Web derzeit nicht geeignet ist. Der alte neue Standard heißt weiterhin HTML.

**HTML wird auch in den nächsten Jahren der gültige Web-Standard bleiben.**

Da es seit 1998 XML gibt, hat sich das World Web Consortium entschlossen, nun auch HTML 4 (1997 eingeführt) „neu zu formulieren“. Das Wort „Neuformulierung“ trifft den Nagel auf den Kopf, denn es geht tatsächlich vorrangig um „formale Dinge“.

HTML wurde inzwischen so „umgebaut“, dass es den strengen XML-Regeln entspricht. Herausgekommen ist XHTML. XHTML 1.0 wurde im Jahre 2000 „herausgebracht“.

**XHTML ist HTML im XML-Gewand!**

## Strenge Regeln von XML

Bist du mit klassischem HTML vertraut? Dann wird dir der Umstieg nicht schwer fallen. Beachte einfach folgende „XML-Regeln“:

### Halte dich an Kleinschreibung

Bei HTML war Groß- und Kleinschreibung egal. In XHTML gilt: Tags und Attributnamen (nicht die Werte!) dürfen nur noch klein geschrieben werden. Diese Regel geht über XML hinaus, da dort auch Großschreibung in den Tags erlaubt ist.

### Jedes Tag benötigt ein End-Tag

Während man in HTML recht schlampig sein durfte, benötigt *jedes* Tag in XHTML ein End-Tag. Das Weglassen von </p> oder </li> usw. ist inzwischen streng verboten. Schreibe also:

```
<p>Das ist ein Absatz</p>
```

### Tags ohne End-Tag intern schließen

In XML werden alle Tags grundsätzlich geschlossen. Wenn ein Tag leer gelassen wird, schließt man es intern, beispielsweise so:

```
<adresse/>
```

Das passiert sogar automatisch! Probiere es aus. Rufe ein XML-Dokument auf. Lösche zur Probe den Inhalt eines Tags. Schaue dir das Ergebnis im Internet Explorer an. Der Browser zeigt ein intern geschlossenes Tag!

Zurück zu XHTML: Neuerdings müssen auch hier alle Tags geschlossen werden, auch wenn sie keine End-Tag besitzen. Eine Linie schrieb man früher so: `<HR>`. Für XHTML muss man unbedingt diese Schreibweise wählen: `<hr />`

Dabei setzt man ein Leerzeichen und danach innerhalb des Tags einen Schrägstrich, den Slash. Doch warum ein Leerzeichen? Das Leerzeichen ist deshalb nötig, damit ältere Browser (z.B. Netscape 4.x) vom „komischen Slash“ nicht verwirrt werden.

**Diese Browser behandeln den Slash als zusätzliches, unbekanntes Attribut, welches sie ignorieren.**

Es wäre nach XHTML aber „korrekter“, eine Linie beispielsweise so zu schreiben: `<hr/>`

### Setze Attributwerte in Gänsefüßchen

Was in XML selbstverständlich ist, gilt nun auch für XHTML. Neuerdings muss man auch hier alle Attributwerte in Gänsefüßchen setzen, beispielsweise so: `<table border="1">`

### Verschachtele korrekt!

Da die korrekte Verschachtelung schon in HTML 4 Pflicht war, gilt sie in X(HT)ML erst recht: Ein untergeordnetes Tag muss immer korrekt im übergeordneten platziert werden.

**Ein Element ist im anderen enthalten!**

Falsch: `<p><i></p></i>`

Richtig: `<p><i></i></p>`

### Vermeide Attribut-Kurzformen

Bei vielen Tags hat man in HTML statt des vollen Attribut-Werts bisher völlig ungeniert eine Kurzform eingesetzt.

Mit *<table border>* legte man Rahmenlinien fest, mit *<hr noshade>* konnte man den evtl. Schatten einer Linie unterdrücken und

```
<option value="seite1.htm"
selected>Seite 1</option>
```

sorgte dafür, dass ein Options-Feld ausgewählt wurde.

Kurzformen sind verboten, schreibe jetzt den vollen Attributwert. In unseren Beispielen wären das: `<table border="1">` bzw.

```
<hr noshade="noshade"> bzw.
```

```
<option value="seite1.htm"
selected="selected">Seite 1</option>
```

### Attribut name wird durch id ersetzt

Kennst du das Attribut *name* in HTML? Es dient zum Benennen von Ankern, Formularen usw. Dieses wurde nun vollständig durch das Attribut *id* (identifier) ersetzt.

Leider können ältere Browser mit *id* noch nichts anfangen. Deshalb setze am besten beide Attribute und vergib die gleichen Werte (oder lasse *id* weg).

Angenommen, du möchtest ein Formular benennen. Dann tust du es folgendermaßen.

```
<form name="frage" id="frage">
... Formularinhalt</form>
```

### Skripte und CSS in SGML „einhüllen“

Neuerdings sollen Skripte und CSS-Verweise in SGML-Anweisungen „gehüllt“ werden, beispielsweise so:

```
<script language="JavaScript"
type="text/javascript">
<![CDATA[
hier steht das eigentliche Skript
]]>
</script>
```

Der Grund sind Sonderzeichen wie & oder >, die XML-fähige Browser verwirren könnten.

Davon rate ich dir in der Praxis jedoch ab, da es meines Wissens noch keinen Browser gibt, der davon verwirrt wird.

### Dokumenttyp-Deklaration setzen

Was in HTML bisher fakultativ war, ist in XHTML Pflicht: Das XHTML-Dokument muss in Bezug auf eine von drei DTDs gültig sein.

Was eine DTD ist, weißt du inzwischen. Schließlich haben wir in diesem Kurs schon einige DTDs selber geschrieben.

Bei XHTML brauchst du dir zum Glück nichts selbst auszudenken, es handelt sich um öffentliche DTDs.

Die Anweisungen liegen öffentlich (public) auf dem Server des W3C.

Sowohl die HTML- als auch die XHTML-Norm wurde vom W3C in drei Dokumenttyp-Definitionen standardisiert. Die DTD definiert, wie du inzwischen selber ausprobiert hast, Anweisungen zur Struktur des Dokuments. Eine DTD wird in der Sprache SGML abgefasst.

## Die drei Fassungen der DTD

Selbermachen ist bei XHTML out. Dafür haben die Jungs vom W3C schon zu viel Arbeit in ihre eigenen Monster-DTDs gesteckt.

Es gibt drei zulässige Geschmacksrichtungen für XHTML (und auch für HTML). Da wären zum einen eine strenge und zum anderen eine „gemäßigte“ Fassung. Die strenge Fassung (*strict DTD*) definiert nur die Elemente, die derzeit gültig und nicht veraltet sind.

Die gemäßigte oder auch Übergangs-DTD (*transitional DTD*) wiederum entspricht den derzeit gebräuchlichen XHTML- (bzw. HTML-) Befehlen.

Außerdem gibt es noch eine DTD für Framesets.

## Die drei DTDs für XHTML

Der Verweis auf die „strict DTD“, die strenge Variante, sieht für XHTML so aus:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Diese DTD beinhaltet strenge Restriktionen: So darf Text nicht mehr einzeln notiert werden sondern muss stets innerhalb von so genannten Blockelementen wie z.B. Absätzen oder * *-Tags stehen.

Du entscheidest dich für die gemäßigte DTD, die „Übergangsfassung“ mit mehr Freiheiten? Dann schreibst du am Anfang deines Dokuments einfach folgende Dokumenttyp-Deklaration:

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

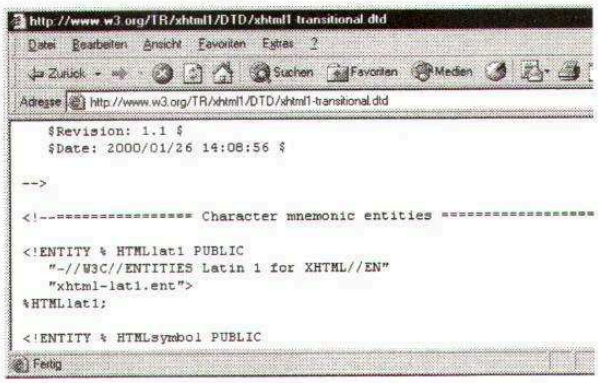
Auch für Framesets (Das ist die Vorschrift zum Einteilen von Seiten in Rahmen) gibt es eine DTD. Benutze für deine „XHTML-Frame-Definitionen“ einfach folgende Dokumenttyp-Deklaration:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

## Die DTD der Übergangsfassung

Bisher haben wir die Dokumenttyp-Deklarationen besprochen. Wo liegen nun die eigentlichen DTDs, also die Definitionen? Wo liegt beispielsweise die „Übergangsfassung“?

Schaue dir zum Beispiel die Stelle *"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"* einmal ganz genau an. Das ist der Verweis! Tippe diese Adresse in deinen Browser ein! Schon siehst du den SGML-Code dieser DTD!




**Die DTD für die „gemäßigte XHTML-Fassung“**

Der Browser (oder ein menschliches SGML/DTD-Genie) könnte diesem Dokument entnehmen, welche Tags und Attribute in welcher Reihenfolge zugelassen sind. Wir verzichten jedoch darauf!

## Der Kompatibilitäts-Modus (Quirks-Modus)

Bis vor kurzem galt: Der Browser „sieht“ zwar den Verweis auf die DTD, ignoriert ihn aber. Er kümmert sich (noch nicht) um das Validieren! Genau wie bei XML war/ist es dem Browser egal, was in der DTD „Nettes“ geschrieben steht; es sei denn, du validierst dieses Dokument von Hand.

Die neuesten Browser (6er Versionen) erkennen jedoch die Angaben der Dokumenttyp-Deklaration und richten sich tlw. auch danach. Sie validieren zwar (noch) nicht gegen die entsprechende DTD. Trotzdem kann ein Regelverstoß von dir zu Fehldarstellungen führen, vor allem beim sehr strengen Netscape 6.x. Ich denke hier an die Höhen- und Breitenangaben in CSS! Der Browser „weiß“ intern: Aha, der Anwender nutzt XHTML strict, dann werde ich keinen Fehler mehr durchgehen lassen.

Wenn du diesen DOCTYPE-Verweis weglässt, schaltet der Browser in einen als „Quirks-Modus“ bezeichneten „Kompatibilitäts-Modus“ um. Der Browser simuliert dann das aus heutiger Sicht „fehlertolerante Anzeige-Verhalten“ älterer Browser und versucht anzuzeigen, was möglich ist.

Wenn du ungezogen bist, fasst du die ersten Zeilen auch weiterhin zum Tag *<html>* zusammen. Das ist zwar nicht standardgerecht, führt aber bei komplexen Layouts evtl. zum richtigen Ergebnis.

Du kannst dein XHTML-Dokument natürlich auch prüfen, um Fehler von vornherein zu vermeiden bzw. um diese überhaupt erst aufzuspüren! Validiere XHTML! Dafür gibt es mehrere Methoden.

## Testen: W3C-Validation Service und TIDY

Surfe zum sehr strengen HTML Validation Service unter validator.w3.org. Hier kannst du eine Web-Adresse angeben oder eine HTML/XML-Datei zum Testen hochladen. Ebenfalls gut geeignet für einen Test ist das HTML-Kit. In diesen HTML-Editor wurde das Prüfprogramm TIDY vom W3C gleich eingebaut. Das derzeit noch kostenlose HTML-Kit gibt's auf www.chami.com/html-kit.

## Ein Musterdokument in XHTML

Hier zeige ich dir ein einfaches „HTML-Dokument“ in der „gemäßigten“ XHTML-Fassung:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>KnowWare-Titel</title>
</head>
<body>
   <h2>PHP f&uuml;r Einsteiger</h2>
   <ul>
      <li>Autor: Johann-Christian Hanke</li>
      <li>Verlag: KnowWare</li>
      <li>Beschreibung: Webserver-Programmierung leicht gemacht</li>
      <li>Preis 4 Euro</li>
   </ul>
   <hr />
   <img src="extra12.gif" width="200" height="275"
      alt="PHP f&uuml;r Einsteiger" />
</body>
</html>
```

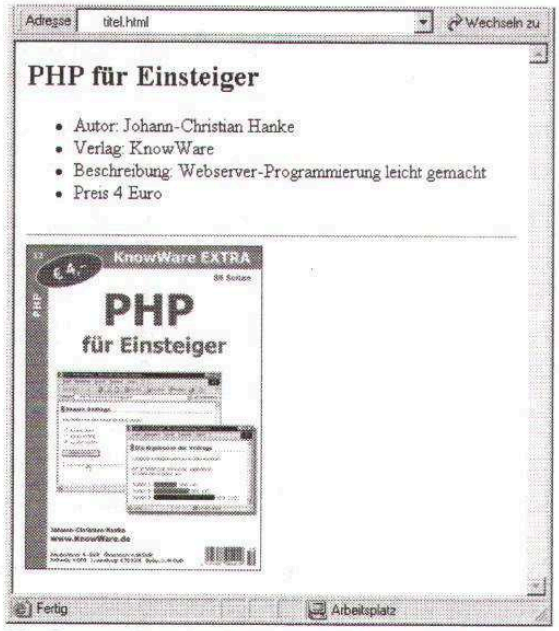
## Der Quelltext kurz erklärt

Das Dokument wird durch die Zeile *<?xml version="1.0" encoding="ISO-8859-1"?>* eingeleitet. Es handelt sich um den XML- Prolog. Dieser ist für XHTML zwar nicht zwingend vorgeschrieben, der Einsatz wird aber empfohlen.

Die zweite Zeile (grau hinterlegt) verweist auf die schon besprochene Dokumenttyp-Definition (DTD). Ich empfehle die „gemäßigte“ Übergangsfassung. In der dritten Zeile wird zusätzlich das *<html>*-Tag „aufgebohrt“, und zwar durch das „Namensraum-Attribut“ *xmlns*, siehe S. 49.

Die Zeile *xmlns="http://www.w3.org/1999/xhtml"* besagt nichts weiter, als dass es sich bei diesem HTML-Dokment um ein Dokument nach der XHTML-Spezifikation handelt. (Auf ein Präfix wird hier übrigens generell verzichtet.)

Ansonsten arbeite ich mit einer Überschrift, einer „unordered list“ (Aufzählung), einer Linie und einer Grafik.




**Die Darstellung im Browser ändert sich nicht**

Beachte die internen Ausschalt-Slashs bei Linie und Grafik. Die entsprechenden Tags habe ich wieder durch Grauhinterlegung markiert.

# Übungsteil D: Klammersetzung, Namensräume und XHTML

Du weißt jetzt auch, wie man

- Elemente in der DTD richtig gruppiert
- Namensräume und Dateninseln setzt
- XHTML-Dateien schreibt

## Übung D1: Gruppierung und Wiederholungsoperatoren

Blättere zurück zu unserem Workshop von Seite 47. Kopiere die dortige *dokument.xml* und *dokument.dtd* in den *aufgaben*-Ordner. Ändere die zweite Zeile der DTD wie folgt:

```
<!ELEMENT abschnitt (head1?,head2,(absatz|rahmen)*)+>
```

Ist die Urfassung der *dokument.xml* noch gültig? Finde weitere Varianten für die XML-Datei!

## Übung D2: Namensraum für HTML in XML erstellen

Binde folgenden HTML-Ausschnitt als Namensraum in ein XML-Dokument namens *london.xml* ein. Nenne das Wurzelelement *text*; füge hier Namensraumdeklaration ein. Vermeide leere Tags!

```
<h1>London und Berlin im Vergleich</h1>
<h2>Die Stadtentwicklung von Berlin</h2>
<p>Wenn man sich mit der Stadtentwicklung Berlins befasst, wird man mit dem
Problem der Mietskaserne konfrontiert. Folgende Themen sind interessant:</p>
<ol>
    <li>soziale Struktur</li>
    <li>Architektur</li>
    <li>Regierung</li>
</ol>
<hr>
```

## Übung D3: XML-Dateninsel in HTML erzeugen

Erstelle ein einfaches HTML-Dokument mit folgender „H1“: *Orte in Hiddensee*. Speichere unter dem Namen *hiddensee.html*. Binde folgende XML-Struktur als Dateninsel ein, verpasse dieser die id *insel*.

```
<hiddensee>
    <ort>Vitte</ort>
    <ort>Kloster</ort>
    <ort>Neuendorf</ort>
</hiddensee>
```

## Übung D4: HTML-Dokument in XHTML umwandeln

Wandle dieses HTML-Dokument in XHTML um. Verwende die „gemäßigte DTD“. Sichere das Ergebnis unter dem Namen *formular.html*. Beachte, dass das <*input*>-Tag ausgeschaltet werden muss.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head><title>Ein Formular</title></head>
<body>
<h1>Geben Sie Ihren Namen ein!</h1>
<form><input type="text"><input type="submit"></form>
<hr>
</body>
</html>
```

# Lektion 11: XML-Dateien formatieren – Schnelleinstieg in CSS

In dieser Lektion lernst du folgendes:

- Kurzeinführung in die CSS-Grundlagen
- Link zur CSS-Datei setzen
- Überblick über die wichtigsten Attribute und Werte von CSS

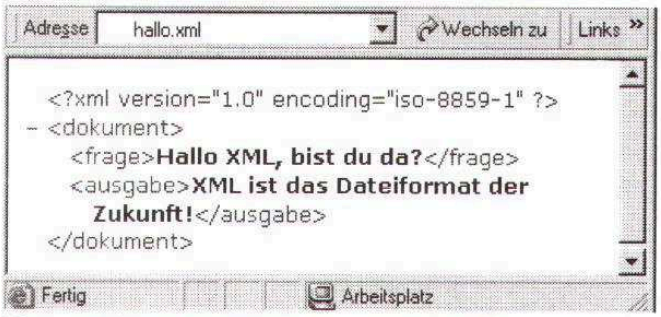
XML sichert nur die logische Struktur. Die Daten sind hierarchisch angeordnet, und zwar als „Baum". Der Internet Explorer zeigt diese baumartige Struktur auch an. Alles das ist dir an dieser Stelle längst vertraut.

Doch ehe du „vor lauter Bäumen den Wald nicht mehr siehst", wollen wir unsere XML-Dateien layouten; also fit machen für die Ausgabe im Browser. Hier bietet sich das Formatieren mit einer speziellen „Layout- und Formatiersprache" an. Die am weitesten verbreitete Sprache heißt CSS.

In dieser Lektion führe ich dich in CSS ein und zeige dir, wie einfach du XML-Dokumente gestalten kannst. Und damit es gleich perfekt gelingt, beginnen wir mit dem Mini-Workshop.

## Mini-Workshop zur Einführung

Krame doch einmal dein allererstes Dokument hervor, die Datei *hallo.xml*. Wenn du sie gerade nicht zur Verfügung hast, schreibe sie fix ab:




**Dieses Dokument besteht nur aus drei Tags**

Das Dokument besteht aus drei Tags. Davon steht das Wurzelelement *dokument* an höchster Hierarchiestelle, es enthält *frage* und *ausgabe*.

Richte dir für diesen Workshop unter deinem Ordner *xmlkurs* einen extra Ordner namens *css* ein. Kopiere die *hallo.xml* hier hinein. Meine Beispiele findest du jedoch unter *lektion11*.

## CSS-Datei erstellen

Die Layoutanweisungen selbst stecken wir in eine CSS-Datei. Das ist – du ahnst es schon – nichts weiter als eine externe Textdatei, der wir pflichtgemäß die Endung *css* verpassen.

Nenne die Datei im Beispiel hallo.css!

Wir wollen dem Dokument die Schriftart *Arial* zuweisen und als Alternative *Helvetica* angeben.

1. Schreibe zuerst den Namen des Elements, im Beispiel *dokument*. Diese Angabe wird als Selektor bezeichnet. Setze danach ein Paar geschweifte Klammern. Dort sollen die Eigenschaften notiert werden.

```
dokument {
...
}
```

2. Du möchtest die Schriftart zuweisen? Notiere das Schlüsselwort *font-family*, setze dahinter einen Doppelpunkt.

```
dokument {
font-family:
}
```

3. Trage die Schriftart ein, hier *Arial*. Wie du siehst, stehen die eigentlichen Anweisungen innerhalb der geschweiften Klammern.

```
dokument {
font-family: Arial
}
```

3. Da du nicht genau weißt, ob das jeweilige Betriebssystem auch Arial anbietet, gibst du hinter Kommas die alternative(n) Schriftart(en) an; *sans-serif* steht dabei für eine ganz allgemeine sachliche Schrift.

```
dokument {
font-family: Arial, Helvetica,
   sans-serif;
}
```

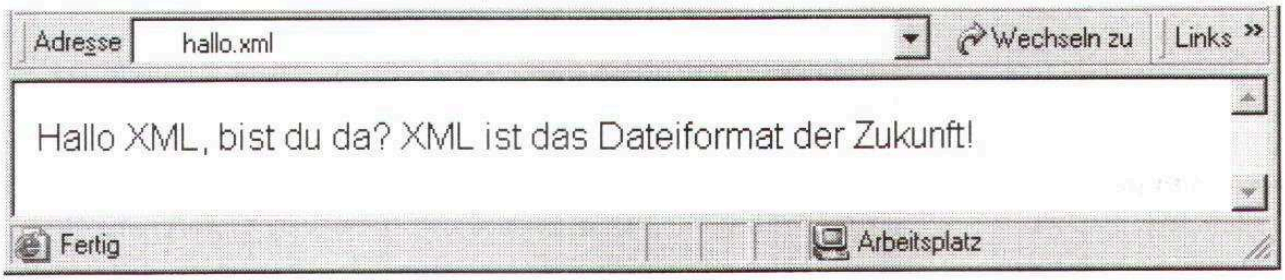
Schließe diese „Zeile" durch ein Semikolon ab. Damit ist deine erste CSS-Regel fertig.

## Verweis auf CSS-Datei setzen

Setze nun einen Verweis auf die eben erstellte CSS-Datei! Die Syntax entnimmst du dem Beispiel: Ich zeige dir das gesamte kurze Dokument und hebe die Stelle mit dem Link auf die CSS-Datei hervor:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="hallo.css" type="text/css"?>
<dokument>
   <frage>Hallo XML, bist du da?</frage>
   <ausgabe>XML ist das Dateiformat der Zukunft!</ausgabe>
</dokument>
```

Speichere das XML-Dokument und aktualisiere die Ansicht im Browser.




**Schriftart Arial: So präsentiert sich die Datei nun im Internet Explorer**

## Der Tiefere gewinnt: Das Kaskadenprinzip

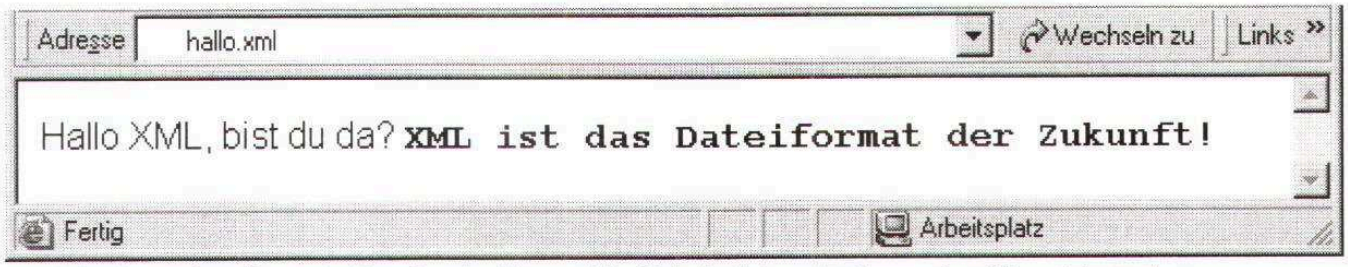
Wir haben die Schriftart dem Wurzelelement zugewiesen. Das ist das in der Hierarchie am höchsten stehende Element. Deshalb werden die Eigenschaften auf alle darunter liegenden Elemente vererbt.

Dieses Prinzip wird – der Name kaskadierende Formatvorlagen sagt es schon – Kaskadenprinzip genannt. Wie bei einer Kaskade werden die Eigenschaften vom in der Hierarchie am höchsten liegenden Element an tiefer liegende („enthaltene") Elemente vererbt. Du kannst jedoch in tiefer liegenden Tags andere Eigenschaften verwenden. Diese überschreiben die „von oben" kommenden Eigenschaften.

Probiere das gleich einmal aus! Wir wollen dem Element *ausgabe* zusätzlich noch die Schriftart *Courier New* bzw. *monotype* zuweisen (wobei *monotype* für eine ganz allgemeine Schrift mit fester Zeichenbreite steht). Außerdem soll der Inhalt des Tags mit *font-weight: bold* fett erscheinen. Notiere also noch folgende Regel und damit folgende Zeilen in der *hallo.css*.

```
ausgabe {
font-family: 'Courier New', monotype;
font-weight: bold;
}
```

Speichere die Datei *hallo.css* erneut und aktualisiere die Ansicht der XML-Datei.




**Die tiefer liegenden Eigenschaften *Courier New* und *bold* sind stärker als die aus dem Wurzelelement**

Warum habe ich die Schriftart *Courier New* eigentlich in Gänsefüßchen gesetzt? Denn normalerweise sind Gänsefüßchen in CSS nicht gestattet! Das liegt daran, dass hier ausnahmsweise im Schriftartnamen ein Leerzeichen zu überbrücken ist. In diesem Fall setzt du einfache Gänsefüßchen.

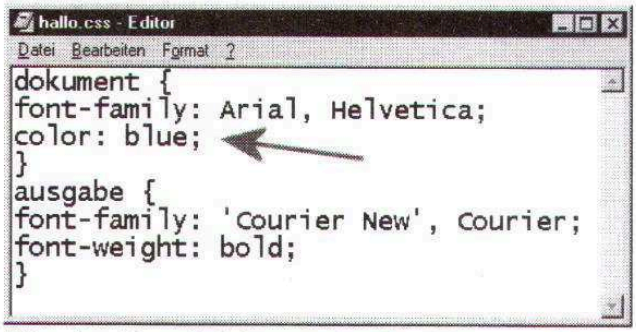
## Grundlagen von CSS

Mit CSS kannst du beispielsweise Schriftart und -größe festlegen, Formate wie fett oder kursiv einschalten, Vorder- und Hintergrundfarben zuweisen und Ränder und Rahmen definieren.

Die Style-Sheet-Syntax wurde an die von Programmiersprachen wie C, C++, Java und Java-Script angelehnt. Genau wie in diesen Programmiersprachen werden auch in CSS alle zusammengehörenden Anweisungen in geschweifte Klammern gehüllt und damit zu einem Block zusammengefasst. Auch in CSS wird jede Zeile durch ein Semikolon (;) abgeschlossen.

### Selektoren

Die zu bearbeitenden Tags heißen „Selektoren". Alle Attribute und Werte (z.B. *font-family: Arial*) gehören jeweils in das geschweifte Klammernpaar. Du möchtest die Schrift zusätzlich blau färben? Dann verwendest du das Attribut *color* und vergibst einen der erlaubten englischen Farbnamen wie *red*, *blue*, *yellow* usw. Probiere es aus! Weise dem Wurzelelement zusätzlich die Farbe blau zu: `color: blue;` Das wirkt sich nun auf alle Elemente aus!




**Mit color: blue wird der gesamte Text blau eingefärbt**

Wenn du zwei Tags gleichzeitig mit einer Eigenschaft versehen willst (Und-Verknüpfung!), reihst du sie mit Komma versehen auf, beispielsweise so: `frage, ausgabe { Eigenschaften; }`

## Die wichtigsten Attribute und Eigenschaften von CSS

Hier liste ich dir die wichtigsten Attribute und Eigenschaften von CSS auf:

| Attribut | verantwortlich für | mögliche Werte (Auswahl) | Notationsbeispiel |
|---|---|---|---|
| font-family | Schriftart | Schriftartname | font-family: Arial, sans-serif; |
| font-size | Schriftgröße | Punkt (pt), Pixel (px) | font-size: 14pt; |
| color | Farbe | engl. Farbname oder Hexadezimalwert | color: red; *oder* color: #ff0000; |
| background-color | Hintergrundfarbe | engl. Farbname oder Hexadezimalwert | background-color: silver; |
| font-weight | „Schriftgewicht" wie fett | bold, light, normal usw. | font-weight: bold; |
| font-variant | Schriftvariation (Kapitälchen) | small-caps | font-variant: small-caps; |
| font-style | Schriftstil (z.B. kursiv) | italic | font-style: italic; |
| line-height | Zeilenhöhe | 1 bzw. 1.2 (Eins-Punkt-Zwei) | line-height: 1.2; |
| margin | Rand, Außenabstand (äußerer Leerraum) | Pixel (px), Zentimeter (cm) | margin: 10px; |
| padding | Innenabstand (Füllung bzw. Polsterung) | Pixel (px), Zentimeter (cm) | padding: 20px; |
| width | Breite | Pixel (px), Zentimeter (cm) | width: 400px; |
| border-style | Rahmenstil | z.B. solid (durchgezogen) | border-style: solid; |
| border-width | Rahmenbreite | Pixel (px), Zentimeter (cm) | border-width: 120px; |

Du suchst einen ausführlichen Einstieg in CSS? Dann empfehle ich dir mein neues Heft „Homepages mit HTML und CSS". Dort lernst du die Sprache CSS bis ins kleinste Detail kennen und verstehen.

**Kurzreferenz zu HTML und CSS**: Eine ausführliche Liste mit den wichtigsten Tags und Attributen auch zu CSS findest du in der Mitte meines Titels „Homepages für Fortgeschrittene". Ich biete dir diese Liste aber auch auf www.jchanke.de/xml zum Download an. Sie liegt im PDF-Format vor.

# Lektion 12: Produktliste mit CSS gestalten

In dieser Lektion lernst du folgendes:

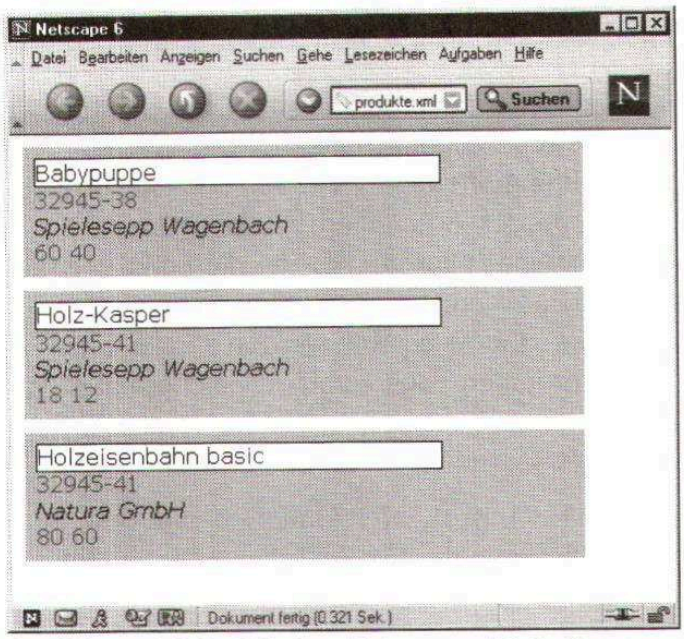
- komplexe Formatierungen mit CSS
- Umbruch steuern mit *display: block*

## Attraktive Gestaltung mit CSS

Wagen wir uns nun an ein komplexes Beispiel. Die Produktliste aus *Lektion 5* soll attraktiv gestaltet werden. Blättere am besten zurück zur Seite 36 und zur Seite 39, um dir den Aufbau des Beispiels noch einmal vor Augen zu führen.

Schaue auf die Abbildung. So soll die Liste im Endeffekt aussehen. Internet Explorer 6 und Netscape 6.x führen hier zum gleichen Ergebnis.

Mach mit! Kopiere zuerst die Datei *produkte.xml* in deinen separaten *css*-Ordner.




**Attraktiv gestaltet mit Farben, Rahmen und Schattierung**

(Ich gehe davon aus, dass du dir diesen Ordner unter dem Projektordner *xmlkurs* eingerichtet hast.)

## Die CSS-Datei erstellen

Füge jetzt schon die Verknüpfung auf die noch zu erstellende CSS-Datei *produkte.css* ein. Besitzt das Dokument eine DTD bzw. ein Schema? Dann setze den CSS-Link stets unter den DTD-Link.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE produkte SYSTEM "produkte.dtd">
<?xml-stylesheet href="produkte.css" type="text/css"?>
<produkte>
   <posten>
      <name>Babypuppe</name>
      ...
   </posten>
</produkte>
```

## Wir erstellen die CSS-Datei in Kompaktschreibweise

Lege nun die Datei *produkte.css* an. Zuerst zeige ich dir den kompletten Quelltext. Auf der nächsten Seite besprechen wir die einzelnen Punkte im Detail. Erst dann schreibst du alles Zeile für Zeile ab und beobachtest das Ergebnis. Apropos Zeilennummern: Die schreibst du natürlich nicht mit ab.

```
1  produkte { font-family: Verdana, sans-serif; font-size: 12pt;
   line-height: 1.2; }
2  posten { display: block; background-color: silver; margin: 10px;
   padding: 8px; width: 400px; }
3  name { display: block; background-color: white; border-style: solid;
   border-width: 1px; width: 300px; }
4  hersteller { display: block; font-style: italic; }
5  nr, preis { color: red; }
```

*Hinweis*: Diesmal habe ich die CSS-Datei in der so genannten Kompaktschreibweise notiert. Statt untereinander werden die eingeklammerten Blöcke einfach nebeneinander notiert. Das spart Platz.

## CSS-Datei Schritt für Schritt erklärt

Schauen wir uns nun die einzelnen Zeilen an, und zwar Schritt für Schritt. Schreibe mit!

1. Schreibe die erste Zeile ab. Diese Zeile definiert die Eigenschaften für das Wurzelelement *produkte*. Hier legen wir quietschvergnügt die Schriftart *Verdana* nebst „allgemeiner Alternativschrift", die Schriftgröße 12 Punkt und eine Zeilenhöhe von 1,2 fest. Diese Eigenschaften werden an alle untergeordneten Tags vererbt. Speichere das Dokument und schaue dir die Vorschau im Browser an.

Nanu?! Noch läuft der Text hintereinander weg wie ein Bandwurm. Einen Zeilenumbruch gibt es lediglich an der rechten Browserkante.


**Der Text läuft hintereinander weg, ohne Umbrüche**

Das ist der Ärger bei XML: Es gibt kein Tag, mit dem man Absätze oder Zeilenumbrüche einfügen könnte. Es gibt keine Tags „auf Blockebene", wie in HTML. Es gibt kein <*p*>, kein <*br*> und kein <*div*>!

Was machen wir da? Wir arbeiten mit einem Trick!

## Umbruch mit display: block

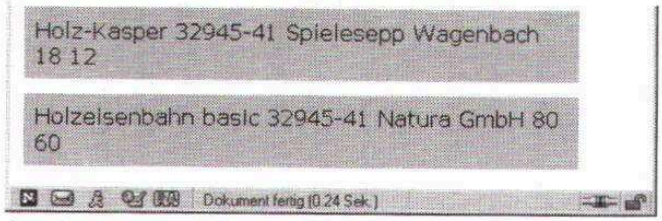
Erzeuge einen Umbruch mit CSS. Dafür sorgt die Anweisung *display*. Sie kann die Attribute *inline* und *block* annehmen. Inline ist voreingestellt, muss also nicht notiert werden. Erst

```
display: block;
```

sichert, dass die jeweilige Passage als eigener Block, praktisch als „Absatz" dargestellt wird.

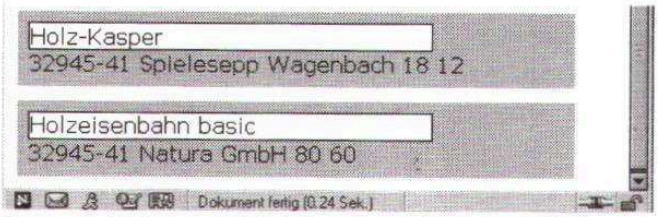
Das Element wird zum „Blockelement". Davor und danach wird ein Umbruch erzeugt.

2. Ran an die zweite Zeile! Jeder „Posten" soll nun einen eigenen Block erhalten. Dann legen wir die Hintergrundfarbe *silver* fest (hellgrau). Der Rand (*margin*) soll 10 Pixel betragen, der Innenrand (*padding*) dagegen 8 Pixel. Das ganze Gebilde bekommt eine Länge von 400 Pixeln spendiert. Notiere die Zeile und begutachte die Vorschau.




**Jetzt besitzt jeder Posten einen eigenen Block!**

3. Die nächste Zeile ist die längste: Auch das Element *name* wird „als Block" formatiert. Diesmal schalte ich die Hintergrundfarbe auf weiß. Außerdem wird mit `border-style: solid;` ein Rahmen erzeugt, dem ich mit `border-width: 1px;` eine Dicke von einem Pixel spendiere. Mit einer Breite von 300 Pixeln ist dieser Rahmen 100 Pixel schmaler als der „übergeordnete" silbergraue Farbteppich.




**Der neue Rahmen hebt sich gut vom Untergrund ab**

4. Die vierte Zeile besitzt kaum Erklärungsbedarf: Der „Hersteller" wird kursiviert und natürlich „als Block" formatiert

5. Doch was passiert in Zeile 5? Hier weise ich den Tags `nr` und `preis` **gleichzeitig** die Farbe rot zu. Dazu liste ich sie mit Komma getrennt auf (Und-Verknüpfung).

Warum verzichte ich hier auf *display: block;*? Und trotzdem wird `nr` als Block dargestellt?! Dafür sorgen die schon als Block formatierten umgebenden Tags! Außerdem sollen die zweimal auftauchenden Preise (*preis*) nicht unter-, sondern nebeneinander dargestellt werden.

# Lektion 13: Mehr Möglichkeiten mit XSL bzw. XSLT

In dieser Lektion lernst du folgendes:

- Grundlagen von XSL bzw. XSTL
- XML mit XSLT nach HTML transformieren

In dieser Lektion führe ich dich *als ersten Einstieg* in die komplizierte Stilsprache XSL ein. Wenn du mit dem Internet Explorer arbeitest, verwende die Version 6 oder installiere MSXML 4, siehe S. 72.

## Einführung in XSL und XSTL

Jetzt kennst du CSS und weißt, wie du deinen XML-Dateien ein attraktives Layout verpasst. Doch auch CSS ist wegen der begrenzten Möglichkeiten nicht das Maß aller Dinge. Ich rede nicht allein über das mit *display: block* mehr schlecht als recht gelöste Problem mit dem Zeilenumbruch. Andere „Mängel" wiegen weit schwerer: Wie erstellst du Tabellen? Wie fügst du Hyperlinks in deine Dokumente ein? Und was ist mit Grafiken und Mediadateien? Hast du darüber schon einmal nachgedacht?

### CSS allein ist überfordert: Ohne HTML geht es nicht!

CSS allein ist hier völlig überfordert. Das Hausformat der gängigen Browser ist und bleibt nun einmal HTML. Denn nur mit HTML-Tags kannst du die Dokumente so im Browser ausgeben, wie du das möchtest! Deshalb holt sich CSS-Verstärkung: Das „Dreamteam" CSS und HTML schafft den Job!

## XSLT wie Transformation

Damit wir CSS und HTML verknüpfen können, brauchen wir den neuen Standard XSL (Extensible Style Sheet Language, erweiterbare Style-Sheet-Sprache). XSL ist seit Oktober 2001 offizielle Empfehlung des World Wide Web Consortiums. XSL selber existiert in mehreren Geschmacksrichtungen. Hier ist vor allem die Unterspezifikation XSLT für uns interessant.

Das *T* steht für *T*ransformations, für Umformung. Dieser Regelsatz ist verantwortlich für das Umformen von XML in HTML, in WML (Sprache für Wap-Handys), in SVG (Scalable Vector Graphics) und dank des „Unterstandards" XSL-FO auch in RTF und PDF.

Keine Bange, soweit treiben wir es nicht. Wir gehen die ersten Schritte und wandeln in HTML um!

## Schritt für Schritt: Erstes Beispiel

Erinnerst du dich an das erste XML-Dokument aus Lektion 1? Dieses wollen wir nun in eine HTML-Struktur transformieren. Wie arbeiten erst mit Überschriften, später mit Tabelle und Grafik.

Kopiere das Dokument in einen separaten Ordner namens *xslt*, damit du nicht durcheinander gerätst.

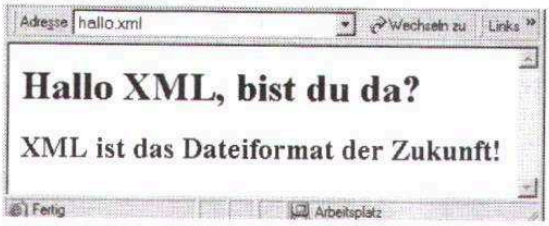
Und so sieht die veränderte *hallo.xml*-Datei aus. Ich habe gleich den Link auf die noch zu erstellende XSL-Stylesheet-Datei *hallo.xsl* eingefügt (grau hinterlegt).

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<dokument>
   <frage>Hallo XML, bist du da?</frage>
   <ausgabe>XML ist das Dateiformat der Zukunft!</ausgabe>
</dokument>
```

Erstellen wir nun gemeinsam unsere erste XSL-Datei! Dabei stricken wir ein komplettes HTML-Gerüst rund um XML. Das Tag <*frage*> soll als H1 und <*ausgabe*> als H2 ausgegeben werden.

## So sieht XSL aus: Die Datei hallo.xsl

Der erste Kontakt mit XSL ist alles andere als logisch und verständlich. Soviel vorweg: Auch XSL-Dokumente werden in der Sprache XML verfasst!

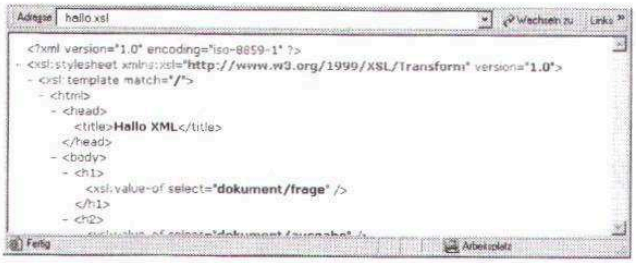


**Im Endeffekt entsteht eine HTML-Datei**

Zuerst zeige ich dir die komplette XSL-Datei.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
    <html><head><title>Hallo XML</title></head>
    <body>
    <h1>
      <xsl:value-of select="dokument/frage"/>
    </h1>
    <h2>
      <xsl:value-of select="dokument/ausgabe"/>
    </h2>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Schreibe das Dokument ab und betrachte die Vorschau im Browser! Und nun untersuchen wir gemeinsam die Bedeutung der einzelnen Zeilen!



**Auch die XSL-Datei ist ein XML-Dokument**

## Der Quelltext Schritt für Schritt

1. Die erste Zeile beherbergt den für XML-Dokumente obligatorischen Prolog.

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

2. Die nächste Zeile beginnt mit *<?xsl:stylesheet ...* Das ist das Wurzelelement dieser Datei. Beachte: Es wird in der letzten Zeile mit *</xsl:stylesheet>* wieder geschlossen. Als Attribut im Wurzelelement folgt xmlns:xsl. Das Schlüsselwort *xmlns* steht für einen der schon besprochenen Namensräume. Der Präfix *xsl* ist dabei zwingend vorgeschrieben. Als nächstes folgen die eindeutig festgelegte „Adresse" des Namensraums und die Versionsbezeichnung von XSL. Das ist bisher Vorschrift und muss praktisch auswendig gelernt (oder abgeschrieben) werden.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

### Das Element xsl:template und die X-Path-Syntax

3. Als nächstes folgt das Element *xsl:template*. Das Wort *template* bedeutet Vorlage. An dieser Stelle wird eigentlich eine Vorlage für die Transformation hinterlegt. Für uns nicht nötig. Denn wir beziehen uns mit dem Attribut-Werte-Paar match="/" auf das Wurzelelement unserer XML-Datei. Denn das ist schließlich unsere Vorlage und die binden wir auf diese Art ein.

```
<xsl:template match="/">
```

Hinter dieser Pfadangabe verbirgt sich eine weitere Sprache namens X-Path. Der Slash (/) verweist dabei auf das Wurzelement. Beachte, dass auch das *template*-Element am Schluss wieder abgeschlossen werden muss.

4. Besonders interessant sind die nächsten Zeilen. Zuerst wird mit dem typischen HTML-Kopf ein HTML-Grundgerüst erzeugt. Mit *<h1></h1>* legst du dann fest, dass das XML-Element *frage* eine Überschrift erster Ordnung werden soll. Dieses Element befindet sich zwischen *<h1></h1>*!

```
<h1>
   <xsl:value-of select="dokument/frage"/>
</h1>
```

**X-Path-Syntax**: Mit *<xsl:value-of select="dokument/frage"/>* legst du den „Fokus" auf *dokument/frage*. Du gibst also den Pfad zum Element *frage* an. Du beginnst beim übergeordneten Tag *dokument* und hangelst dich bis zu *frage* durch. Schließe das Tag wieder intern mit dem Slash ab.

5. Auf gleiche Art und Weise „erreichst" du das Element *ausgabe*. Dafür sorgen die nächsten drei Zeilen. Auch hier wird mit *dokument/ausgabe* der Fokus auf das zu gestaltende Tag gelegt:

```
<h2>
   <xsl:value-of select="dokument/ausgabe"/>
</h2>
```

6. Vergiss nicht, „Template-Tag" und Wurzelelement am Ende der Datei wieder zu schließen.

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

Was lernst du aus diesem Beispiel? XSLT ist ziemlich kompliziert. Doch das war erst der Beginn!

**Tabellenstruktur? Grafik? Kein Problem!**

## Tabellenstruktur um das Dokument legen

Ich hatte davon gesprochen, dass XML mit XSLT *transformiert* wird, und zwar nach HTML: Stricke nun ein komplettes HTML-Dokument mit Tabelle und Grafik um die Elemente drum herum. HTML kannst du? Sehr gut! Im Beispiel habe ich alle Dateien von hallo in *holla* umbenannt. Hier nun der Code der XSL-Datei *holla.xsl*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
   <html><head><title>Tabelle</title></head>
   <body>
   <table border="1">
      <tr><th><xsl:value-of select="dokument/frage"/></th></tr>
      <tr><td><xsl:value-of select="dokument/ausgabe"/></td></tr>
      <tr><td><img src="lighthouse.gif"/></td></tr>
   </table>
   </body></html>
</xsl:template>
</xsl:stylesheet>
```


**Auch das geht: ein HTML-Hyperlink**

Beachte, „leere Tags" wie *img* usw. stets intern zu schließen, da XML geschlossene Tags verlangt!

***Übung***: XSLT erlaubt das Einbinden beliebiger HTML-Elemente! Mache die Grafik anklickbar: Binde also zusätzlich einen HTML-Hyperlink ein, der auf KnowWare verweist! Benenne XML-Dokument und XSL-Stylesheet in *hello* um. Meine Musterdateien findest du im Ordner *lektion13*.

# Lektion 14: XSLT für Profis: Gestalten und Sortieren

In dieser Lektion lernst du folgendes:

- komplexe XSTL-Vorlagen erstellen
- XML-Dokumente mit XSLT sortieren

Unser XSLT-Beispiel aus der vorigen Lektion hat bisher einen großen Haken. Es ist zu simpel gestrickt! Dort kann jedes Tag nur ein einziges Mal vorkommen. Bei mehrfachem Vorkommen genügt *value-of select* nicht mehr!

## Beispiel Titelleiste (titel.xml)

Nehmen wir ein sinnvolles Beispiel aus der Praxis. Erinnerst du dich an unsere Titelleiste vom Anfang des Kurses?

Vergleiche mit Kapitel 2 und 3 ab Seite 24: Es geht um die *titel.xml*, die du mitsamt der dazugehörigen DTD-Datei *titel.dtd* verwenden kannst. Kopiere die zwei Dateien in einen separaten Ordner namens *xslt*. Füge in die *titel.xml* unter dem DTD-Link einen Link zur noch zu erstellenden XSL-Datei *titel.xsl* ein.

```
<?xml-stylesheet href="titel.xsl"
type="text/xsl"?>
```

## Gestalten mit CSS

Ab jetzt „spielt die Musik" in der XSL-Datei, siehe Nebenseite. Zum Formatieren greifen wir auf eine externe CSS-Datei namens *titel.css* zurück. Diese wird per Link in die HTML-Struktur der *titel.xsl* eingebunden, siehe Zeile 7!

Wer schon mit HTML und CSS vertraut ist, wird mit dieser Syntax keine Schwierigkeiten haben.

## For-each-Schleife

Doch schauen wir zur XML-Struktur.

Interessant wird es ab Zeile 10. Wir legen den Fokus auf *titelliste/heft* und bearbeiten alle Tags auf diesem „Unterast". Doch wie? Schließlich müssen wir berücksichtigten, dass die Elemente mehrmals vorkommen. Programmierer aufgepasst: Wir arbeiten mit einer Schleife!

Das Schlüsselwort *xsl:for-each* signalisiert, dass es sich um eine Schleife handelt. Hinter dem select-Attribut wird die X-Path-Pfadangabe *titelliste/heft* notiert.

Das bewirkt, dass nun alle Elemente unter diesem Pfad betroffen sind. Das Auslesen wird so lange wiederholt, bis alle Daten aus der XML-Datei „abgearbeitet" wurden.

Die Schleife wird in Zeile 30 beendet!

## Elemente einbinden

Als nächstes werden die Elemente *titel*, *autor*, *verlag*, *beschreibung* und *preis* eingebunden. Dazu dient das Schlüsselwort *value-of select*.

Da wir schon auf dem richtigen „Pfad" sind (*titelliste/heft*), müssen wir nur noch den Namen des Tags angeben. Ganz eindeutig ist das z.B. für den Titel in Zeile 13, den wir zusätzlich in die HTML-Überschrift *<h3></h3>* „gewickelt" haben. Schließe das Tag intern!

## Element kommt mehrfach vor

Erinnerst du dich? Das Element *autor* kann einmal, aber auch mehrmals vorkommen!

Zum „Zugriff" auf Autor/Autoren dient daher wieder eine *for-each*-Schleife. Diese wird in Zeile 15 eingeleitet und in Zeile 20 geschlossen.

Das eigentliche Auslesen des Tag-Inhalts (Zeile 17) geschieht durch ein weiteres Element der X-Path-Syntax, den Punkt (.). Dieser steht als Platzhalter für beliebig viele Elemente.

Ohne diese Schleife würde nur ein einziges Vorkommen von *autor* angezeigt werden!

## Daten sortieren

Zum Schluss zeige ich dir, wie einfach man Daten ordnet. Im Beispiel sollen die Hefte nach Namen des Titels sortiert werden. Nutze die Anweisung *xsl:sort*. Setze also zusätzlich folgende Zeile unter die Einleitung der ersten xsl:for-each-Schleife, siehe auch Zeile 11:

```
<xsl:sort select="titel"/>
```

Die voreingestellte Sortierreihenfolge ist *ascending*, also aufsteigend. Für eine absteigende Sortierung ergänzt du das Tag um *order="descending"*, schreibe dann:

```
<xsl:sort select="titel"
order="descending"/>
```

## Quelltext der Datei titel.xsl im Überblick

Ich habe versucht, dir mit optischen Hervorhebungen den Durchblick zu erleichtern.

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
3  <xsl:template match="/">
4  <html>
5  <head>
6  <title>KnowWare-Titel</title>
7  <link rel="stylesheet" type="text/css"  href="titel.css"/>
8  </head>
9  <body>
10 <xsl:for-each select="titelliste/heft">
11 <xsl:sort select="titel"/>
12 <h3>
13 <xsl:value-of select="titel"/>
14 </h3>
15 <xsl:for-each select="autor">
16 <i>
17 <xsl:value-of select="."/>
18 </i>
19 <br/>
20 </xsl:for-each>
21 <div>
22 <xsl:value-of select="verlag"/>
23 </div>
24 <i>
25 <xsl:value-of select="beschreibung"/>
26 </i>
27 <div>
28 <xsl:value-of select="preis"/>
29 </div>
30 </xsl:for-each>
31 </body>
32 </html>
33 </xsl:template>
34 </xsl:stylesheet>
```


**Alphabetisch sortiert mit der xsl:sort-Anweisung**

## Quelltext der Datei titel.css im Überblick

Hier zeige ich die noch den erfreulich kurzen Quelltext der externen CSS-Datei *titel.css*. Du kannst die Stilanweisungen gerne erweitern, um der Titelliste ein noch attraktiveres Aussehen zu verleihen!

```
body { font-family: Arial, Helvetica; }
h3 { font-size: 14pt; color: blue; }
```

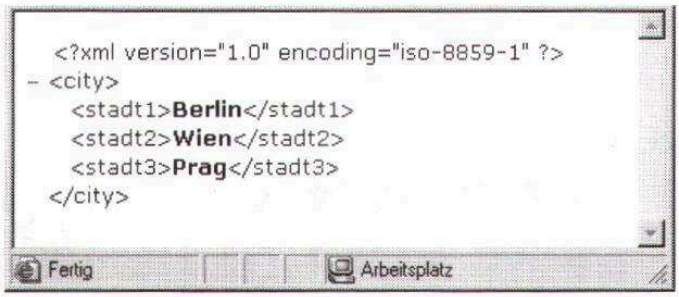
Das soll's an dieser Stelle gewesen sein als Schnelleinstieg in XSLT. Diese Sprache ist so mächtig und umfassend, dass man gut und gerne ein ganzes Heft zu XSTL verfassen könnte. Vieles geht hier schon in Richtung Programmierung. Erfahrungsgemäß ist das Interesse an solchen Themen jedoch gering, oder irren wir uns? Melde dich, wenn Interesse besteht (www.jchanke.de/xml), dann würden wir das Thema evtl. in einem weiterführenden Titel vertiefen.

# Übungsteil E: Übungen zu CSS und XSLT

Du weißt jetzt auch, wie man:

- Dateien mit CSS gestaltet
- Umbrüche in CSS erstellt
- XML-Dateien mit XSLT nach HTML transformiert

## Übung E1: XML-Dokument mit CSS layouten




**Ein einfaches XML-Dokument**

Schreibe folgendes Dokument ab und speichere es unter dem Namen *city.xml*. Erstelle eine CSS-Datei namens *city.css*.

Alle Städte sollen in der Schriftart Arial, Schriftgröße 14 Punkt, Schriftfarbe grau (gray) dargestellt werden. Denke auch an die Angabe einer allgemeinen Alternativschrift.

## Übung E2: Städte untereinander darstellen

Sorge dafür, dass die Städte nicht mehr neben- sondern untereinander dargestellt werden. Sieh außerdem einen Rand von 10 Pixel um jede Stadt herum vor.

## Übung E3: Link zur CSS-Datei mit Link zur DTD verbinden

Erstelle für das Dokument *city.xml* zusätzlich die DTD *city.dtd*. Binde diese DTD per Link ein und achte auf die richtige Reihenfolge der Verknüpfungen.

## Übung E4: Kommentar setzen – Dokument auf Gültigkeit prüfen

Prüfe das Dokument auf Gültigkeit. Es gibt die Fehlermeldung *Kein XML-Dokument*? Dann musst du den Link auf die CSS-Datei kurzzeitig „auskommentieren". Dafür verwendest du die gleiche Kommentarsyntax, die du schon aus HTML kennst, und zwar in der kurzen Form:

```
<!-- Hier stehen auskommentierte Passagen -->
```

Entferne den Kommentar am Schluss wieder!

## Übung E5: Dokument mit XSLT gestalten

Erstelle eine Kopie des Dokumentes, die du *city2.xml* nennst. Gestalte dieses Dokument mit XSLT. Erstelle dafür eine XLS-Datei namens *city.xsl*: Jede Stadt soll lediglich durch einen eigenen Absatz hervorgehoben werden. Es sind keine weiteren Formatierungen nötig.

## Übung E6: Mehrere gleichnamige Tags durch Schleife ansprechen

```
- <city>
    <stadt>Berlin</stadt>
    <stadt>Wien</stadt>
    <stadt>Prag</stadt>
  </city>
```

Verändere die XML-Datei! Speichere zuerst die Ursprungsdatei *city.xml* unter dem Namen *cityneu.xml*. Ändere jetzt die Tags *stadt1*, *stadt2* und *stadt3* jeweils nur in *stadt*. Erstelle eine XSL-Datei namens *cityneu.xsl*. Wie erreichst du hier, dass wieder *jedes* XML-Tag in einen eigenen Absatz gestellt wird?

## Übung E7: DTD cityneu.dtd einbinden

Wie muss nun die zur Datei *cityneu.xml* passende DTD namens *cityneu.dtd* aussehen? Erstelle diese Datei und binde sie ein!

# Lektion 15: Einführung in das Konzept von XML-Schema

In dieser Lektion lernst du folgendes:

- Warum Schema statt DTD?
- Referenz auf Schema-Datei setzen
- einfaches Schema erstellen
- Schema validieren

Hier geht es vorrangig um ein „Hineinschnuppern" in ein noch (zu?) junges Thema.

## Warum Schema statt DTD?

Das nervt! Kaum hatte ich den Umgang mit DTDs halbwegs verstanden, hieß es: April, April, alles veraltet. Jetzt gibt es ja das neue Schema-Konzept. Doch Frust beiseite, ganz so schlimm ist es nicht. Momentan sind die DTDs noch Stand der Dinge. Das Schema-Konzept geriet u.a. wegen seiner Kompliziertheit schon in die Kritik. Zur Zeit arbeitet man an einer fehlerberichtigten (besseren?) Version 1.1.

**Während DTDs Jahrzehnte alt sind, wurde der Standard Schema erst 2001 eingeführt.**

Aber warum brauchte man etwas Neues? Das liegt an den Nachteilen der DTDs:

Wenn du eine DTD schreibst, kannst du nicht genau festlegen, welchem Datentyp deine Zeichen entsprechen sollen. Der Anweisung #PCDATA „ist es egal", ob du mit Buchstaben oder Zahlen arbeitest. Die Beschränkung auf Zahlen, auf das Währungsformat oder sogar auf Ganzzahlen (integer) ist nicht möglich.

Auch die Wiederholungsoperatoren +, * oder ? haben nur eine beschränkte Aussagekraft. Wenn das Element nur dreimal oder zwischen fünf- und zehnmal vorhanden sein soll?

Außerdem wäre es wünschenswert, dass eine „Schablone" für XML nun auch in XML verfasst wird und nicht in SGML wie die DTDs.

## Beispiel Titelliste

Wie immer greifen wir auf ein Beispiel zurück. Damit du vergleichen kannst, verwenden wir wieder die Titelliste aus Lektion 2 und 3. Soviel schon vorweg: XML-Schema ist komplizierter und „tippaufwändiger" als die klassische DTD!

Blättere zum Verständnis zum Abschnitt ab Seite 24 zurück! Hier zur Erinnerung ein Ausschnitt aus der XML-Datei:

```
<?xml version="1.0" encoding=
"ISO-8859-1"?>
<titelliste>
   <heft>
      <titel>Start mit ...</titel>
      <autor>Pia Hardy</autor>
      <autor>Kare Thomsen</autor>
      <verlag>KnowWare</verlag>
      <preis>4</preis>
   </heft>
...
</titelliste>
```

Für das nächste Beispiel richtest du dir unter deinem Projektordner *xmlkurs* einen Ordner namens *schema* ein. Kopiere die Ursprungsfassung der *titel.xml* in diesen Ordner.

Die Arbeit, die wir uns in Kapitel 3 mit der DTD gemacht haben, stecken wir nun in das Schema.

## Referenz auf Schema-Datei setzen

Gleich erstellen wir die Schema-Datei.

**Eine Schema-Datei trägt die Endung *xsd*!**

Da wir die Schema-Datei ebenfalls *titel* nennen, heißt sie *titel.xsd*. Doch vorher setzt du in der *titel.xml* schon einmal eine Referenz auf diese Datei. Wende dich also der *titel.xml* zu.

Blenden wir zurück. Wie war das bei den DTDs? Bei der DTD hattest du einfach unter dem Prolog einen Link zur DTD platziert. Diesmal geht das nicht. Wir müssen eine Referenz auf die Schema-Datei setzen, und zwar direkt im Wurzelelement.

Die „offizielle" Referenzierung ist kompliziert. Statt einer simplen Syntax hat man eine „ganze Sprache" für das Referenzieren erfunden, bei der sich alle möglichen Attribut-Werte-Paare ein munteres Stelldichein geben. Damit der Frust nicht schon beim Referenzieren beginnt, kürzen wir auf zulässige Weise ab. Bitte blättere um!

### Die einfache Variante

So wird die Schema-Datei in XML „referenziert“:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<titelliste xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="titel.xsd">
   <heft>
   ...
   </heft>
</titelliste>
```

Was passiert in der für uns interessanten grau hinterlegten „Referenz-Zeile“? Zuerst erstelle ich den vorgegebenen Namensraum für ein Präfix namens *xsi*. Das Kürzel *xsi* steht als Abkürzung für XML-Schema instance. Darunter befindet sich ein Attribut, über dessen Namen du dir keine allzu großen Gedanken machen solltest. Wichtig ist, dass wir in Gänsefüßchen Namen (und ggf.) Pfad zur entsprechenden Schema-Datei angeben. Hier ist es die *titel.xsd*, die wir gemeinsam erstellen!

## Die Schema-Datei titel.xsd

Zuerst zeige ich dir diese Datei im Gesamtüberblick. Der Quellcode sieht, da stimmst du mir sicher zu, recht gruselig aus. Schreibe ihn ab und lies dir meine Erklärungen auf der nächsten Seite durch!

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3  <xsd:element name="titelliste">
4   <xsd:complexType>
5    <xsd:sequence>
6     <xsd:element name="heft" maxOccurs="unbounded">
7      <xsd:complexType>
8       <xsd:sequence>
9        <xsd:element name="titel" type="xsd:string"/>
10       <xsd:element name="autor" type="xsd:string" maxOccurs="2"/>
11       <xsd:element name="verlag" type="xsd:string"/>
12       <xsd:element name="beschreibung" type="xsd:string" minOccurs="0"/>
13       <xsd:element name="preis" type="xsd:decimal"/>
14      </xsd:sequence>
15     </xsd:complexType>
16    </xsd:element>
17   </xsd:sequence>
18  </xsd:complexType>
19 </xsd:element>
20 </xsd:schema>
```

## XSD ist XML: Prolog und Namensraum

So, und jetzt gehen wir ins Detail. Nach dem Motto: Wiedersehen macht Freude findest du in Zeile 1 unseren geliebten Prolog. Bester Beweis, dass die XSD-Datei eine waschechte XML-Datei ist.

> Rufe die Datei doch einmal im Internet Explorer auf: Schon siehst du die XML-Struktur! (Gegebenenfalls ziehst du die Datei bei gedrückter linker Maustaste in das Internet Explorer-Fenster hinein.)

Zeile 2 legt den Namensraum für unser Schema fest. Dieser sieht (für die derzeitige Schema-Variante) exakt so aus und darf nicht verändert werden. Es handelt sich hierbei um das Wurzelelement der XML-Schema-Datei. Dieses wird in Zeile 20 wieder geschlossen.

## Tag definieren mit xsd:element

In der nächsten Zeile wird das erste Element aus der XML-Datei definiert. Es ist das Element *titelliste*, das Wurzelelement der XML-Datei.

Das Attribut `name` verweist auf das zu definierende Element. Weitere Attribute sind an dieser Stelle (noch) nicht nötig. Erst in den untergeordneten Elementen wird es interessant.

Das Element *titelliste* wird in Zeile 19 wieder abgeschaltet. Alles dazwischen ist in diesem Tag „enthalten". Ich habe diese „Tag-Klammer" durch Grauhinterlegung optisch hervorgehoben.

## Eine Ebene hinabsteigen

Schaue jetzt auf Zeile 4 und Zeile 5.

```
<xsd:complexType>
    <xsd:sequence>
```

Was bedeuten diese merkwürdigen Tags? Merke dir nur soviel: Mit *<xsd:complexType>* steigst du eine Hierarchieebene nach unten. Mit *<xsd:sequence>* dagegen leitest du die „Sequenz" des oder der dort vorhandenen Element-Definitionen ein.

**Diese Tags müssen ebenfalls wieder abgeschaltet werden!**

Das Abschalten geschieht in Zeile 17 und 18.

## Häufigkeit des Vorkommens

Nun bist du in Zeile 6 gelandet, wo erst einmal das Element *heft* definiert wird. Doch was besagt das zusätzliche Attribut *maxOccurs* mit dem Wert *unbounded*? Damit gibst du das maximal erlaubte Vorkommen von *heft* an; *unbounded* bedeutet dabei unbegrenzt. Schließlich sollen in unserer Titelliste unendlich viele Hefte vorkommen dürfen!

Notiere gerne auch Zahlen: *maxOccurs="10"* würde nur 10 Heftvorkommen erlauben. Das Gegenteil von *maxOccurs* ist *minOccurs*.

Auch das *heft*-Element muss wieder abgeschaltet werden, und zwar in Zeile 16. Es umklammert schließlich Unterelemente. Auch diese „Klammer" habe ich optisch markiert.

Mache dir doch einmal den Jux und vergleiche mit der DTD von Seite 27! Wofür wir auf Seite 27 noch eine simple Zeile benötigten, müssen wir hier sechs komplizierte Reihen opfern.

## Datentyp definieren

In Zeile 7 und 8 stiefeln wir wieder eine Hierarchieebene nach unten. Dort legen wir nun die übrig gebliebenen Elemente fest.

Diese Definitionen werden interessanterweise intern abgeschlossen, beachte also den Slash am Ende. Warum? Da wir nun „ganz unten" sind, gibt es in diesem Fall keine weiteren Elemente, die umklammert werden müssen.

Doch was bedeutet das *type*-Attribut? Damit kannst du den Datentyp festlegen. Mit *type="xsd:string"* gehen wir auf Nummer sicher und erlauben eine Folge „beliebiger in XML zulässiger Zeichen".

Daneben gibt es unzählige weitere Datentypen von *xsd:integer* bis *xsd:boolean*. Wer eine „höhere Programmiersprache" wie C oder C+ „spricht" wird alle diese „Typen" freudig wieder erkennen. Ein Anfänger gruselt sich davor. Neben *xsd:integer* gibt es beispielsweise:

- *xsd:decimal* (Ganz- oder Dezimalzahl wie -1, 2,5 oder 22)
- *xsd:integer* (negative oder positive Ganzzahl wie -225 oder 7)
- *xsd:date* (Datum wie 2002-09-10, also Jahr-Monat-Tag)
- *xsd:boolean* (Wahrheitswert wie true oder false bzw. 1 oder 0)

## Repetitio est mater studiorum

Ja, ja, die alten Lateiner ... Bis jetzt entspricht eigentlich noch alles der DTD von Seite 27! Im XML-Schema sind wir schon in Zeile 10, dort tummeln wir uns noch mit *autor+* in Zeile 2.

Spätestens an dieser Stelle gehen wir mit dem Schema über die Möglichkeiten einer DTD hinaus. Zeile 10 bestimmt, dass der Autor mindestens einmal, aber nicht mehr als zweimal erscheint: KnowWare-Hefte mit mehr Autoren gibt es nicht! Eine DTD kann so etwas nicht.

## maxOccurs und minOccurs

Dafür sorgt das schon bekannte *maxOccurs*-Attribut, dem ich den Wert 2 mit auf den Weg gegeben habe. Doch warum prahle ich erst an dieser Stelle mit dem Attribut herum?

Wenn du *maxOccurs* oder *minOccurs* weglässt, gilt jeweils die Voreinstellung 1! Soll das Element also nur einmal vorkommen, lässt du dieses Attribut unbedingt weg!

Aus diesem Wissen ergeben sich interessante Konstellationen: Wenn du lediglich *minOccurs="0"* angibst, wird *maxOccurs* automatisch auf 1 geschaltet. Und das entspricht dann ziemlich genau unserem Fragezeichen-Wiederholungsoperator aus der DTD.

Bitte vergleiche mit Zeile 12 und mit dem Element *beschreibung?* aus der DTD von Seite 27.

## type="xsd:decimal"

Die nächste kleine „Verbesserung" gegenüber der DTD steckt in Zeile 13. Mit der Anweisung *type="xsd:decimal"* lege ich fest, dass der Preis eine Ganz- oder Dezimalzahl sein muss.

## Bitte validieren!

Nachdem du nun dein erstes Schema derart erfolgreich hinter dich gebracht hast, möchtest du doch sicher einmal prüfen, ob du alles richtig gemacht hast.

Sprach's und klickte mit rechts in die Datei. Befehl *Valdiate XML* - Validation successful. Wow, so gut bin ich also. Fehler eingebaut in das Schema – Validation successful. Hier stimmt doch etwas nicht!


**Alles Lüge! XML-Schema wird gar nicht validiert!**

Und tatsächlich sind weder XMLINT noch der Internet Explorer 6 von Hause aus zum Validieren einer XML-Datei gegen das XML-Schema in der Lage. Informationsstand: Juni 2002.

## Und es geht doch ...

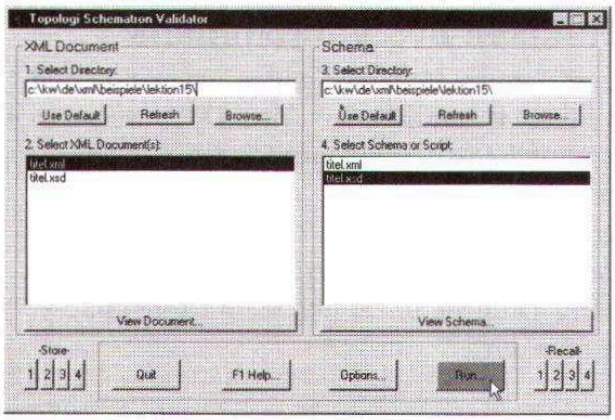
... um frei nach Galilei zu sprechen! Tue dies:

- Besorge dir **MSXML** (derzeit) **Version 4.0;** ein Tool, welches die XML-Fähigkeiten des Internet Explorers „modernisiert".
- Installiere **Schematron**, ein Tool, welches in Zusammenspiel mit MSXML gegen ein Schema validieren kann: www.topologi.com

Surfe zu http://msdn.microsoft.com/xml und klicke im rechten Bereich auf den Link zu MSXML 4.0. Lade die MSXML, Version 4 herunter, derzeit aktuell ist die Version „mit Service Pack 2". Dieses Release benötigt den Installer in der Version 2.0. Dieser ist erst ab Windows XP enthalten. Falls es beim Installieren deshalb eine Fehlermeldung gibt, musst du dir auch noch den neuesten Installer von Microsoft herunterladen, die Datei *Instmsi.exe*. Suche danach auf http://msdn.microsoft.com.

Besorge dir nun von www.topologi.com das Tool *Schematron*, welches du nach Registrierung im „free download" bekommst. Im Endeffekt besitzt du ein Zip-Archiv namens *TSV usw.zip*. Packe es aus und starte die Exe- bzw. Msi-Datei. Das Tool ist schnell installiert!

*Starte das Tool über Start/Programme/Topologi/ Schematron Validator usw.*




**Der Klick auf Run startet das Validieren**

Im linken Bereich stellst du die XML-Datei ein, rechts suchst du das passende Schema heraus.

Dass der Hersteller von Schematron eigentlich eine andere, „konkurrierende" Schema-Variante namens Schematron „propagiert", sei nur am Rande erwähnt. Beim Thema *Schema* herrscht offenbar noch Uneinigkeit und Chaos!

# Lektion 16: Hyperlinks mit XLink

In dieser Lektion lernst du folgendes:

- einfache Hyperlinks erstellen mit XLink

Last but not least zeige ich dir, wie man in XML Hyperlinks darstellt. Dass diese Teile bisher nur in Netscape 6.x funktionieren, darüber sehen wir wohlwollend hinweg.

XLink ist seit Juni 2001 offizieller Standard.

## Hyperlinks in HTML und XML

Im Beispiel setzt du einen einfachen Hyperlink. Das HTML-Muster ist dir bekannt?

```
<a href="http://www.knowware.de"
title="KnowWare-Verlag"
target="_blank">KnowWare</a>
```

Der Link führt zur KnowWare-Seite, wird (in neueren Browsern) mit dem Titel *KnowWare-Verlag* geschmückt und ruft ein neues Browserfenster auf. Das setzen wir nun in XLink um.

### Das XLink-Beispiel

Schreibe folgende Zeilen ab. Speichere das Dokument unter dem Namen *hyper.xml*.

```
<?xml version="1.0" encoding=
"ISO-8859-1"?>
<link xmlns:xlink=
"http://www.w3.org/1999/xlink">
 <verweis xlink:type="simple"
   xlink:href="http://knowware.de"
   xlink:title="KnowWare-Verlag"
   xlink:show="new">
KnowWare
 </verweis>
</link>
```

Beachte, dass der Umbruch in den ersten beiden Zeilen nur den engen Spalten geschuldet ist. Du kannst es so abschreiben (keine Zeichenketten innerhalb der Gänsefüßchen auseinanderreißen), oder du gönnst dir dafür eine ganze Zeile.

Das Untereinander-Notieren der Attribute des Elements *verweis* ist dagegen Absicht.

Du möchtest das Beispiel ausprobieren? Auch das Verbinden mit einer externen CSS-Datei bringt beim Internet Explorer keinen Erfolg!


**Nur Netscape 6.x interpretiert derzeit den Hyperlink!**

Nun zum Quellcode: Es handelt sich, wie du siehst, um eine nach dem Motto „supersimpel" gestrickte XML-Datei. Ich habe mir lediglich die Tags *link* (Wurzelelement) und *verweis* (für einen Testlink) ausgedacht.

## Namensraum definieren

Im Wurzelelement *link* definiere ich den Namensraum. Das hierfür zwingende vorgeschriebene Namensraum-Präfix heißt *xlink*. Diese Zeile ist Pflicht, der Namensraum ist fest vorgegeben.

Hinweis: Mehr zum Thema Namensräume erfährst du ab Seite 49.

### Attribut xlink:type

Das Attribut *xlink:type* beschreibt den Typ des Hyperlinks. Mit dem Wert *simple* legen wir einen einfachen Link fest.

### Attribute xlink:href und xlink:title

Das Attribut *xlink:href* ist selbsterklärend. Hier tragen wir – wie von HTML gewohnt – den Pfad zur entsprechenden Seite ein, auf die wir verweisen wollen.

Auch *xlink:title* wird dir sicher klar. Damit zaubern wir den o.g. „Titel" zum Hyperlink dazu. Netscape zeigt freundlicherweise eine gelbe Kurzinfo an, siehe Abbildung!

### Attribut xlink:show

Das Attribut *xlink:show* kann z.B. die Werte *new* und *replace* annehmen. Mit *new* sorgen wir dafür, dass das „Linkergebnis" in einem neuen Browserfenster erscheint. Mit *replace* wird dagegen der Inhalt der aktuellen Seite ersetzt.

# Informieren, Lernen, Nachfragen: Ressourcen zu XML

An dieser Stelle sind wir fast schon am Schluss angelangt. Zumindest am Schluss dieses Heftes, denn über XML gäbe es noch weit mehr zu berichten.

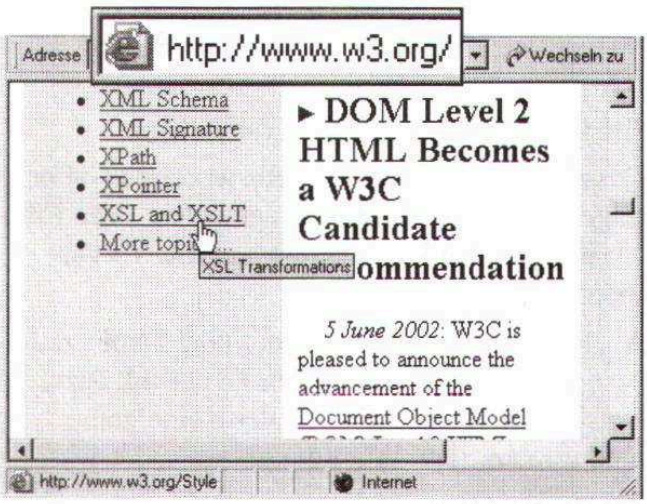
In einem weiterführenden Kurs müsste man stärker auf XSLT und XML-Schema eingehen und XML im Zusammenhang mit Skripting und dem so genannten Document Objekt Model (DOM) besprechen. Hast du Lust, diesen Kurs zu schreiben? Oder kennst du jemanden, der das könnte? Wir suchen Autoren! Bitte schaue zu *Autoren gesucht* unter www.knowware.de!

Doch vor allem empfehle ich dir: Schaue dich ein wenig um im Web. Viele Informationen gibt es kostenlos!

## Informieren: Seiten des W3C

Du suchst Informationen aus allererster Hand? Du möchtest dich genauestens über die jeweiligen Standards informieren? Deine Englisch-Kenntnisse sind sehr gut und du scheust dich nicht vor „Tech-Talk"?

Dann sind die Seiten des W3C der richtige Anlaufpunkt.




**Hier findest du alle Infos zu den neuen Sprachen**

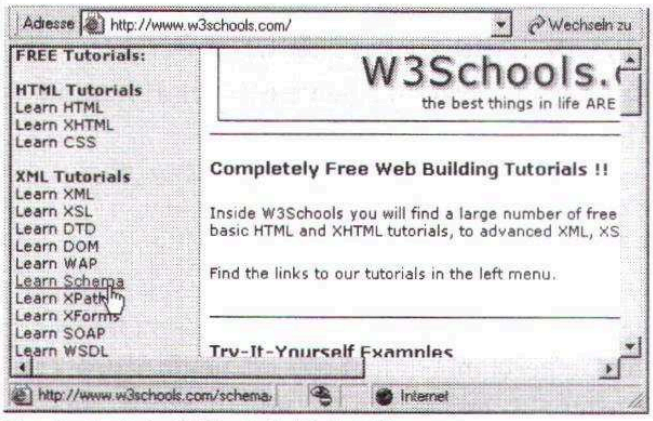
Surfe zu www.w3.org und klicke die entsprechende Rubrik an. Allerdings schwankt die Qualität der Dokumente von kompliziert bis unverständlich. Durchbeißen!

## Lernen: W3Schools und Co.

Du stehst statt technischer Informationen eher auf Schritt-für-Schritt-Anleitungen? Dann „google" doch einfach nach deinem Wunschthema. Suche beispielsweise nach

```
xml-schema tutorial
```

wenn du beispielsweise Informationen zu XML-Schema finden möchtest.




**Fundierte Schritt-für-Schritt-Anleitungen**

Sehr gut gefallen haben mir die W3Schools Online-Tutorials, unter anderem zu DTD, DOM, WAP, X-Path oder dem Standard für Formulare X-Form. Der Haken: Auch diese Seiten sind wieder auf Englisch: www.w3schools.com

## Nachfragen: Usegroups zu XML

Für einen guten Einstieg empfehle ich natürlich auch die Newsgruppen. Hier helfen sich XML-Freunde gegenseitig und du kannst Fragen stellen. Neben der Gruppe

- `comp.text.xml`

gibt es auch den deutschsprachigen Vertreter

- `de.comp.text.xml`

Surfe zu www.google.de, gehe auf die Registerzunge *Groups* und tippe den gewünschten Gruppennamen in die Suchen-Zeile ein. Hier kannst du bequem mitlesen und selber „posten".

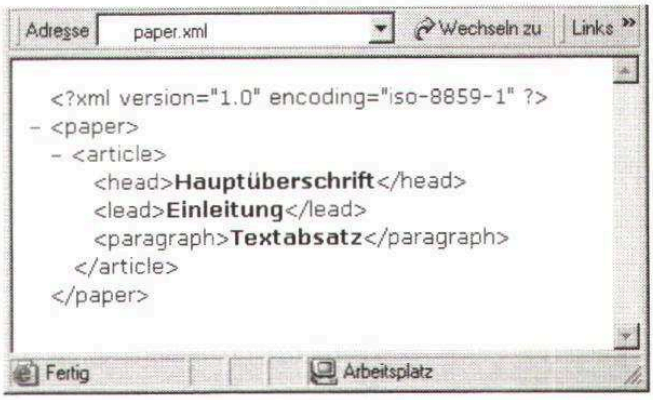
Wenn du die Gruppe regelmäßig verfolgen möchtest, empfiehlt sich ein Newsreader wie Outlook Express. Genaue Usenet-Anleitungen findest du in unserem Outlook-Express-Titel.

# Übungsteil F: Übungen zu Schema und Hyperlinks

Du weißt jetzt auch, wie man:

- XML-Schema-Dateien schreibt
- Im XML-Dokument eine Referenz auf das Schema anbringt
- XML-Dateien gegen ein Schema validiert
- Mit XML Hyperlinks setzt

## Übung F1: XML-Datei für Zeitschriftenartikel erstellen




**So sieht die Grundstruktur für die Artikel aus**

Eine Zeitung stellt auf XML um. Alle Redakteure müssen die Artikel im neuen Format abliefern. Dafür stellt der Chefredakteur diese „Artikel-Vorlage" zur Verfügung. Schreibe das Dokument ab und speichere unter dem Namen *paper.xml*.

*Beachte:* Das Tag *article* darf unbegrenzt oft vorkommen. Die Elemente *head* und *lead* dürfen nur einmal vorkommen, das Elemente *paragraph* dagegen zehnmal (mehr Absätze sollen die Autoren nicht schreiben dürfen). Die Reihenfolge ist *head - lead - paragraph*.

## Übung F2: DTD erzeugen

Hast du die *paper.xml* erzeugt? Speichere eine Kopie dieser Datei unter dem Namen *paper1.xml*. Was willst du hier erreichen? Versuche die oben gestellten Forderungen mit einer klassischen DTD (*paper.dtd*) umzusetzen. Was wird nicht möglich sein? Validiere das Dokument!

## Übung F3: Referenz auf Schema-Datei setzen

Erzeuge eine zweite Version der Ursprungsdatei, die du unter dem Namen *paper2.xml* speicherst. Füge hier eine Referenz auf die noch zu erstellende Schema-Datei *paper.xsd* ein.

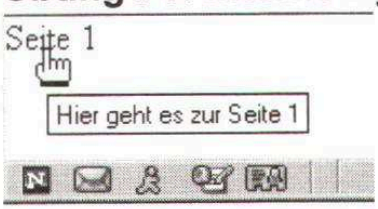
## Übung F4: Schema-Datei schreiben

Erstelle nun die entsprechende Schema-Datei *paper.xsd*. Versuche dabei, die in Übung F1 gestellten Forderungen umzusetzen.

## Übung F5: XML-Datei gegen Schema validieren

Validiere die XML-Datei gegen das von dir selbst erstellte Schema!

## Übung F6: Interne Hyperlinks setzen




Erstelle zwei einfache XML-Dateien (im gleichen Ordner). Die erste soll *seite1.xml* heißen, die zweite dagegen *seite2.xml*. Erstelle in jeder Seite einen Hyperlink, der jeweils zur anderen Seite führt. Die Linkbeschreibung (also der sichtbare Text) soll *Seite 1* bzw. *Seite 2* heißen. Als „Titel" wählst du z.B. *Hier geht es zur Seite 1* usw.

Achte darauf, dass kein neues Browserfenster erscheint, sondern der Inhalt im alten Fenster „ersetzt" wird. Teste die Hyperlinks im Netscape-Browser (falls möglich).

# Stichwortverzeichnis

# Was kann KnowWare für dich tun?

## Über KnowWare

KnowWare ist Erfinder der „Computer-Softbook-Reihe“ und Marktführer in Deutschland. Regelmäßig geben wir neue Hefte zu Computerthemen heraus,

- preiswert, sympathisch und verständlich.

Wir vertreiben unsere Titel über Zeitschriftskioske und ausgewählte (Bahnhofs-)Buchhändler. Alle lieferbaren Titel sind auch noch nach Jahren erhältlich. Sie werden regelmäßig nachgedruckt und ggf. aktualisiert.

Dein Händler ist dir beim Beschaffen älterer Titel nicht behilflich? Dann bestelle einfach selber! Auf www.knowware.de bekommst du alle lieferbaren Hefte. Wenn du mehr als zehn Hefte bestellst, entfallen die Versandkosten.

KnowWare gibt es seit nunmehr zehn Jahren in Deutschland. Für diesen Erfolg möchten wir uns bei dir, lieber Leser, herzlich bedanken!

## Autor werden

KnowWare sucht stets gute Autoren zu allen interessanten Themen. Du bist ausgewiesener Spezialist auf einem bestimmten Gebiet? Du hast einen Vorschlag? Her damit! Schreibe eine E-Mail an unseren Lektor Karl Antz, lektorat@knowware.de.

Bitte vergiss nicht folgende Angaben in Mail:

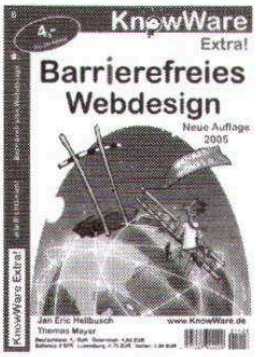
- Themenwunsch
- Terminvorstellung
- Name, Anschrift, Telefon, E-Mail-Adresse
- Alter
- Vorerfahrungen

Als Betreff der Mail schreibst du *Bewerbung als Autor für ... (Thema einsetzen)*.

P.S. Wir ermutigen auch Anfänger, sich bei KnowWare zu bewerben! Entscheidend sind nicht die Zahl der Veröffentlichungen oder das Alter, sondern fachliches Know-how, Engagement und natürlich ein sehr guter Schreibstil.

## Weitere Titel zum Thema

Kennst du schon folgende Hefte?




**Barrierefreies Webdesign, 88 Seiten**

Jan Eric Hellbusch

Für Leser, die sich um die Zugänglichkeit ihrer Seite Gedanken machen. Und für alle, die sich für HTML und/oder Grundsätze der übersichtlichen Gestaltung interessieren. Positiv rezensiert in c't 27/2002. Neuauflage!


**Homepages für Profis, 72 Seiten**

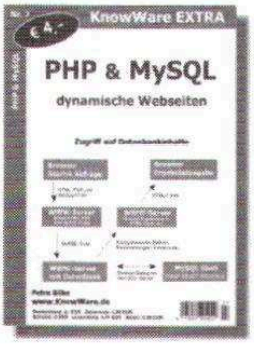
Johann-Christian Hanke

Die ideale Ergänzung zu „Erfolg mit der Homepage“: Hosting, Content Management, Server Side Includes (SSI), Usability (Benutzerfreundlichkeit), Errorpage, Weblog, Forum, Newsfeed, Shopsysteme u.v.a.m.


**Homepages mit HTML und CSS, 88 Seiten**

Johann-Christian Hanke

Anspruchsvoller Grundkurs zu HTML und vor allem zur „Stilsprache“ CSS (Cascading Style Sheets). Ideal geeignet als Unterrichtsmaterial. Mit vielen Beispielen und Übungen. Nachgedruckt und aktualisiert.




**PHP & MySQL, 72 Seite**

Petra Bilke

Datenbankprogrammierung mit PHP und MySQL. Am Beispiel. Mit kurzer Einführung in PHP. (Für eine ausführliche PHP-Einführung empfehlen wir die Lektüre von „PHP für Einsteiger“ von Joh.-Christian Hanke.)

Viel Spaß beim Lesen und viel Erfolg am PC wünscht das gesamte KnowWare-Team!

## 120 leicht verständliche Titel ab 4,- €

Übertragen in das Faxbestellformular a. d. Rückencover -->
**Oder gleich online bestellen: www.knowware.de**

| Linux | Nr. | € |
|---|---|---|
| **Umsteigen! ... auf Linux** | E20 | 4,40 |
| Linux für **Poweruser** | E22 | 4,40 |
| **Linux im Windows-Netzwerk** leicht & verst | M07 | 4,40 |
| **PowerPoint** | | |
| Start mit PowerPoint 7 | 140 | 4 |
| PowerPoint 2000 für Einsteiger, 3. Aufl. 2005 | S01 | 4 |
| **PowerPoint** XP/2002/2003 Einst., 2. Aufl. | P32 | 4,40 |
| **Programmierung** | | |
| **C++ leicht und verständlich (neu 2005)** | S08 | 5,20 |
| **C++** für Einsteiger | E06 | 4 |
| CGI & Perl für Einsteiger | P15 | 4 |
| Java2 für Einsteiger | P19 | 4 |
| Spiele in Flash | 174 | 4 |
| **Delphi** für Einsteiger, 2. Auflage | E21 | 4,40 |
| **Windows, WLAN und Netzwerke** | | |
| Start mit Windows 3.1 | 105 | 4 |
| Windows 2000 für Einsteiger | E05 | 4 |
| Windows 2000 für Fortg. | P17 | 4 |
| Windows ME/98 für Einsteiger | 166 | 4 |
| Windows-**Netzwerke** für Einsteiger, 5. Aufl. | E14 | 4 |
| Windows Super User | P25 | 4 |
| Windows Tips und Tricks (Version 95/98) | P02 | 4 |
| **WLAN** für Einsteiger | E19 | 4 |
| **Windows XP** leicht und verständlich | P38 | 4,40 |
| Windows Tipps & Tricks XP-2000 | 180 | 4 |
| Windows-Tuning m. d. Registry, Aufl. 2005 | P01 | 4 |
| **Windows XP Crash- und Systemlösung** | 187 | 4,40 |
| **Word (Textverarbeitung)** | | |
| **Word in der Praxis** (Workshops) | S13 | 5,20 |
| **Gestalten mit Word** (Layouten, Drucken) | S11 | 5,20 |
| **Word 2000-2003 im Schnellkurs** | P39 | 4 |
| Word 7 für Anfänger | 129 | 4 |
| Word 97 für Anfänger | E03 | 4 |
| Word 7 für Fortgeschrittene | 132 | 4 |
| Weiter mit Word 97/2000 | 160 | 4 |
| Word für Studenten Ver. 95-2003, 6. Aufl. | 138 | 4,40 |
| Word 2000 für Einsteiger | 164 | 4 |
| **Word 2003**/2002 leicht & verständlich | P37 | 4 |
| Word für Profis (Satz, Layout), 2. Aufl 2005 | M03 | 4,40 |
| **Office** | | |
| **Office 2003** für Einsteiger (SSL) | P33 | 4 |
| Office 2000 für Sekretäre/innen | S06 | 4 |
| Open- & Starwriter für Einsteiger | 175 | 4 |
| Staroffice 5.x für Einsteiger | P09 | 4 |
| **Publisher leicht & verständlich** (XP/2003) | S15 | 5,20 |
| **Sonstiges** | | |
| **InDesign** leicht & verständlich (Ver. 2/CS) | S09 | 5,20 |
| Eltern und ComputerKids | E17 | 4 |
| **Unternehmensdaten gekonnt auswerten** | S07 | 5,20 |
| Acrobat und PDF für Einsteiger | E10 | 4 |
| Rund um den PC | 143 | 4 |
| MindManager X5 für Einsteiger | M02 | 4,40 |
| **Lebenshilfe Life21** | | |
| Dein Recht auf Wohngeld | LB01 | 4 |
| Dein Weg aus der Schuldenfalle | LB02 | 4 |
| Dein Recht auf BAföG | LB03 | 4 |
| Arbeitslos? ... nicht mit mir! | LC01 | 4 |
| Unterhalt für Kinder und Eltern | LC02 | 4 |
| Arbeitslosengeld II einf. erkl. (Hartz IV) | LC03 | 4 |
| Gesünder Wohnen | LC04 | 4 |
| **KnowWare Tools, Handbuch u. Software** | | |
| **HackDetect**: Hacker finden a. d. Homepage | T01 | 7,80 |
| | | |

| Access und Datenbanken | Nr | € |
|---|---|---|
| Start mit **Access** 2 | 107 | 4 |
| Start mit Access 7/97 | 146 | 4 |
| Access 2000 für Einsteiger | 162 | 4 |
| Access 97/2000 für Fortg. | 154 | 4 |
| Access **2003**/2002 leicht ver... (Neuaufl. v. 172) | 173 | 4,40 |
| Access 97/2000: Formulare u. Berichte | P18 | 4 |
| **Access 2002/3**: Formulare u. Berichte | 182 | 4,40 |
| Access mit Makros automatisieren | P29 | 4 |
| Datenbanken und SQL leicht & verständlich | 131 | 4,40 |
| **Excel (Tabellenkalkulation)** | | |
| **Excel** 2000-2003 **Schnellkurs** ... Übungen | 186 | 4,40 |
| **Excel** VBA Makro-Programmierung (Ver. 5/95) | 126 | 4 |
| VBA-Programmierung mit **Excel** (ab 2000) | M06 | 4,40 |
| Excel 2000 leicht & verständlich | 169 | 4,40 |
| Excel 2000 für Fortgeschrittene | P20 | 4 |
| Excel 2002 leicht & verständlich | 179 | 4,40 |
| **Excel 2003** (02/XP) für Fortgeschrittene | P35 | 4,40 |
| Diagramme in Excel (2003, 2002/XP) | 185 | 4,40 |
| **Bildbearbeitung und Digitalfotografie** | | |
| **Bildbearbeitung** für Einsteiger (2005, farbig) | P16 | 4,40 |
| CorelDraw 7-10 für Einsteiger | P23 | 4 |
| Paint Shop Pro 5/6 für Einsteiger | P10 | 4 |
| PhotoShop LE für Einsteiger | S05 | 4 |
| **PhotoShop 6/7** leicht. & verst, **Neuauflage** | E15 | 5,20 |
| WebDesign mit **Fireworks** | P31 | 4 |
| **Digitalfotografie** und Bildbearbeitung, 2. Aufl. | P36 | 4,40 |
| **Brennen, Hardware, Musik und Video** | | |
| **Nero 6 Reloaded**: Daten, Audio und Video | E23 | 4,40 |
| CD & DVD brennen mit Nero | 176 | 4 |
| **Hardware** aufrüsten | P26 | 4 |
| Musik bearbeiten am PC | E11 | 4 |
| Video am PC | E18 | 4 |
| **Homepage und Internet** | | |
| Barrierefreies Webdesign (2. Aufl. 2005) | E08 | 4 |
| Dreamweaver 3/4 für Einsteiger | P14 | 4 |
| Dreamweaver MX für Einsteiger | P27 | 4 |
| Erfolg mit der Homepage | P30 | 4 |
| **Flash kompakt & ActionScript** (Flash MX-8) | S12 | 5,20 |
| Frontpage 2000 für Einsteiger | 159 | 4 |
| **Frontpage 2003** (2002) leicht & verständlich | 184 | 4 |
| GoLive 5 für Einsteiger | P21 | 4 |
| **HomePages** für Einsteiger (6. Auflage 2005) | 161 | 4,40 |
| Homepages mit HTML und CSS (5. Aufl. 2006) | 168 | 5,20 |
| Homepages für Fortgeschrittene (Aufl. 2005) | P12 | 4 |
| Homepages für Profis | 178 | 4 |
| Intranet, HTML und Java | 133 | 4 |
| JavaScript für Einsteiger | P06 | 4 |
| JavaScript für Fortgeschrittene | P24 | 4 |
| **PHP5 leicht & verständlich** | S10 | 5,20 |
| PHP und MySQL für Einsteiger | E07 | 4 |
| PHP und MySQL auf der Homepage, 2. Aufl. 05 | M04 | 4,40 |
| **ASP.net leicht & verständlich** | S14 | 5,20 |
| XML für Einsteiger (3. Auflage) | E13 | 4,40 |
| Anonym im Internet | E16 | 4 |
| **Sicherheit im Internet** | 183 | 4 |
| E-Mail mit Outlook Express 5/6, 5. Aufl. 05 | P08 | 4 |
| **Internet leicht & verständlich:** ab 5/2006 | 177 | 5,20 |
| **Online bewerben (Ratgeber zum Erfolg)** | **M08** | 7,80 |
| Outlook 98/2000/2002 Einsteiger | 165 | 4 |
| **Outlook 2003** leicht & verständl. (Neuauflage) | P34 | 4,40 |
| Lotus Notes 6 für Einsteiger | M01 | 4,40 |
| Viren, Hacker, Firewalls (2. Auflage 2005) | 170 | 4 |
| **Start ins Internet** | 157 | 4 |
| Internet-Surfen für Einsteiger | 181 | 4 |
| Kaufen & Verkaufen im Internet (ebay, osc) | M05 | 4,40 |
| **Voice over IP** - Billig telefonieren m.d. Internet | P40 | 4,40 |
| | | |

# KnowWare Bestellseite

ab 4,- €

Bestellfax: **0541 33145-33** oder direkt: **www.knowware.de**


**Excel 2000-2003 im Schnellkurs, 3. Auflage**
**64 Seiten, 4,40 €**
**ISBN 87-91364-26-4**
**Johann-Christian Hanke**
Praxisorientiertes Lernen anhand von Beispielen und Übungen. Komplett farbig.

**Heft-Nr. 186**
**..... Bestellmenge**


**Word 2003/2002 leicht & verständlich**
**88 Seiten, 4,- €**
**ISBN 87-91364-47-7**
**Johann-Christian Hanke**
Unser erfolgreicher Grundkurs zu Microsoft Word in brandneuer Auflage!

**Heft-Nr. P37**
**..... Bestellmenge**


**Unternehmensdaten gekonnt auswerten,**
**80 Seiten, 5,20 €**
**ISBN 87-91364-53-1**
**Petra und Steffen Bilke**
Data Mining und Data Warehouse mit Excel und Access! Highlight!

**Heft-Nr. S07**
**..... Bestellmenge**


**Excel 2003 (2002/XP) für Fortgeschrittene**
**72 Seiten, 4,- €**
**ISBN 87-91364-39-6**
**Thomas Barkow**
Ein echtes Profi-Heft, welches in die Tiefe geht. Funktionen, Import, Pivot-Tabellen ...

**Heft-Nr. P35**
**..... Bestellmenge**


**InDesign leicht & verständlich**
**80 Seiten, 5,20 €**
**ISBN 87-91364-58-2**
**Kerstin Vorwerk**
Druck machen mit InDesign, dem Satz- & Layoutprogramm: Version 2 und CS.

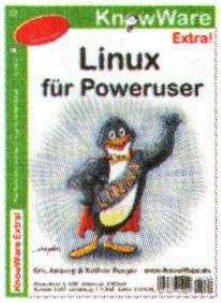
**Heft-Nr. S09**
**..... Bestellmenge**


**Windows XP leicht & verständlich, 3. Auflage**
**88 Seiten, 4,40 €**
**ISBN 87-91364-52-3**
**Johann-Christian Hanke**
Grundkurs zu Windows in 3. Auflage. Noch mehr Tipps! Top-Bestseller!

**Heft-Nr. P38**
**..... Bestellmenge**


**Gesünder Wohnen**
**64 Seiten, 4,- €**
**ISBN 87-91364-30-2**
**Brigitte Harste**
Wohnqualität steigern und gesund bleiben! Mit Fotos, Diagrammen und Tabellen. Kompetent geschrieben!

**Heft-Nr. LC04**
**..... Bestellmenge**




**Linux für Poweruser**
**72 Seiten, 4,40 €**
**ISBN 87-91364-42-6**
**Eric Amberg, Kathrin Reeger**
Die Fortsetzung des Kult-Hefts "Umsteigen! .. auf Linux". Verständlich geschrieben!

**Heft-Nr. E22**
**..... Bestellmenge**


**Sicherheit im Internet**
**80 Seiten, 4,- €**
**ISBN 87-91364-38-8**
**Eric Amberg**
Nieder mit Viren, Hackern, Spam und Spyware! Eric zeigt dir, wie du deinen Rechner zuverlässig schützt.

**Heft-Nr. 183**
**..... Bestellmenge**


**Arbeitslosengeld II einfach erklärt**
**64 Seiten, 4,- €**
**ISBN 87-91364-29-9**
**Stefan Igelmann**
Stefan erklärt dir Chancen und Risiken der Hartz IV-Reformen. Bestseller!

**Heft-Nr. LC03**
**..... Bestellmenge**

**Versandkosten** (für Deutschland):
bis Bestellwert von 12,- € . . . . 2,50 €
bis Bestellwert von 28,- € . . . . 2,90 €
bis Bestellwert von 40,- € . . . . 5,50 €
**darüber versandkostenfrei**
Lieferung in 2-3 Werktagen auf Rechnung!

**weitere Titel bestellen***

| Menge | Titel des Heftes | Heft-Nr. | Preis |
|---|---|---|---|
| . . . . . x | . . . . . | | € |
| . . . . . x | . . . . . | | € |
| . . . . . x | . . . . . | | € |
| . . . . . x | . . . . . | | € |
| . . . . . x | . . . . . | | € |
| . . . . . x | . . . . . | | € |
| . . . . . x | . . . . . | | € |
| . . . . . x | . . . . . | | € |
| . . . . . x | . . . . . | | € |

*siehe Vorseite

**Wohin sollen wir die Bestellung schicken?**

Name: .....
Str: .....
PLZ, Ort: .....
Telefon: .....
E-Mail: .....
Datum/Unterschrift: .....

Bestellung per Fax: 0541 33145-33
Bestellung per Telefon: 0541 33145-20
Bestellung per Post an: KnowWare-Vertrieb
Postfach 3920
D-49029 Osnabrück
Oder online bestellen: **www.knowware.de**

**Die Heftpreise enthalten die MwSt. und gelten innerhalb von Deutschland. Auslandskonditionen siehe www.knowware.de.**

**XML leicht & verständl., E13 | ISBN: 87-90785-89-4**

9 788790 785895 00440

**120 preiswerte KnowWare-Titel! Sofort lieferbar! Bestellinformationen für den Handel auf Seite 2.**