XML

in 21 Tagen

Simon North
Paul Hermans

Deutsche Übersetzung:
Judith Muhr

XML

in 21 Tagen

Schritt für Schritt
Einstieg in Fähigkeiten
und Konzepte

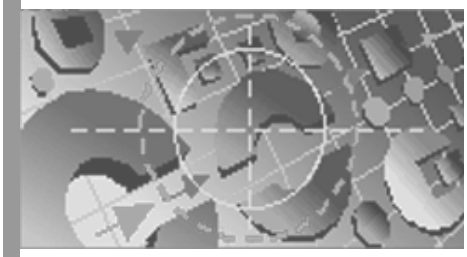
Praktischer Einsatz
anhand beispielhafter
XML-Applikationen

Bitte beachten Sie:

Der originalen Printversion liegt eine CD-ROM bei.

In der vorliegenden elektronischen Version ist die Lieferung einer CD-ROM nicht enthalten.

Alle Hinweise und alle Verweise auf die CD-ROM sind ungültig.



Markt+Technik Verlag

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist
bei der Deutschen Bibliothek erhältlich.

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen.
Trotzdem können Fehler nicht vollständig ausgeschlossen werden.
Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Autorisierte Übersetzung der amerikanischen Originalausgabe:
Teach Yourself XML in 21 Days © 1999 by SAMS Publishing

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Software-Bezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen oder sollten als solche betrachtet werden.

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.
Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

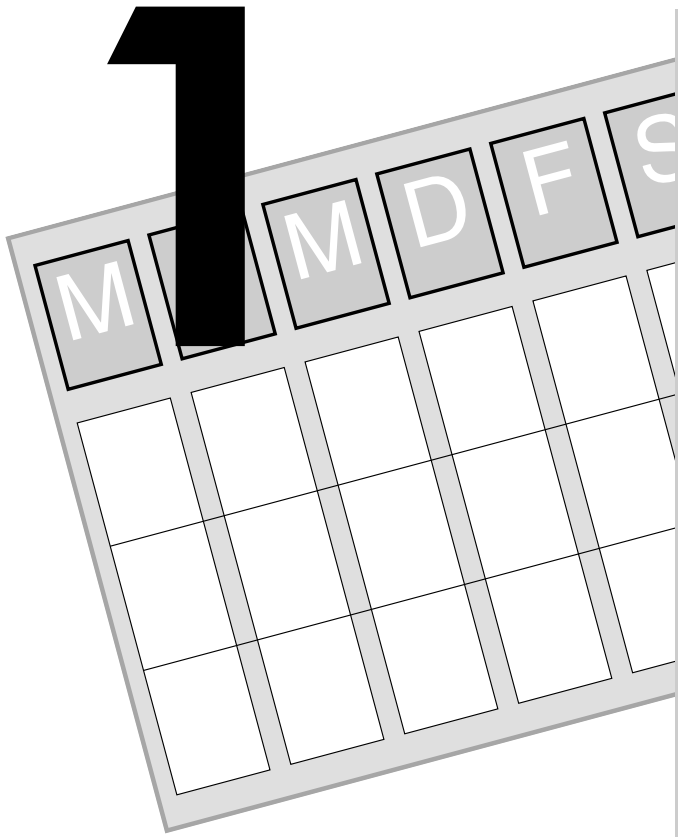
10 9 8 7 6 5 4 3 2 1

04 03 02 01 00

ISBN 3-8272-5687-9

© 2000 by Markt+Technik Verlag,
ein Imprint der Pearson Education Deutschland GmbH.
Martin-Kollar-Straße 10–12, D–81829 München/Germany
Alle Rechte vorbehalten
Übersetzung: Judith Muhr
Lektorat: Jürgen Bergmoser, jbergmoser@pearson.de
Herstellung: Claudia Bäurle, cbaeurle@pearson.de
Satz: reemers publishing services gmbh, Krefeld
Einbandgestaltung: Grafikdesign Heinz H. Rauner, München
Druck und Verarbeitung: Bercker, Kevelaer
Printed in Germany

Inhaltsverzeichnis



Auf einen Blick

Was ist XML, und warum sollte ich mich überhaupt damit beschäftigen?

Woche 1

Willkommen bei *XML in 21 Tagen*! Dieses Kapitel erklärt Ihnen, wozu Sie *XML* (*Extensible Markup Language*) brauchen. Heute werden Sie die folgenden Dinge kennenlernen:

- Die Bedeutung von XML in einem immer komplexeren Internet
- Schwächen von HTML, die es für den Internet-Kommerz ungeeignet machen
- SGML, die Standard Generalized Markup Language, und ihre Beziehung zu XML
- Schwächen anderer Tag- und Markup-Sprachen
- Wie SGML und HTML von XML profitieren
- Vorteile von XML für Nichtprogrammierer

Das Web wird erwachsen

Geliebt oder gehaßt – das Internet und das World Wide Web (WWW) werden uns erhalten bleiben. Was Sie auch tun, das Web wird eine immer größere Bedeutung in Ihrem Leben einnehmen.

Das Internet ist aus einem kleinen Experiment einiger Naturwissenschaftler entstanden und hat sich zu einem der phänomenalsten Ereignisse der gesamten Computergeschichte entwickelt. Man könnte sich das Ganze als Äquivalent zur Industriellen Revolution vorstellen: als Anbruch des Informationszeitalters.

In dem ursprünglichen Vorschlag für das CERN-Management (das europäische Atomforschungszentrum) beschrieb Tim Berners-Lee (der anerkannte Erfinder des Web) 1989 seine Vision eines

»... universell verbundenen Informationssystems, dessen Allgemeinheit und Portierbarkeit wichtiger sein sollen als verrückte Grafiken und komplexe Zusatzfunktionen.«

Das Web hat sich in den letzten zehn Jahren in alle Richtungen weiterentwickelt, und ich frage mich manchmal, was Berners-Lee von seiner Erfindung im derzeitigen Stadium hält.

Und dennoch steckt das Web immer noch in den Kinderschuhen. Seine Nutzung geht langsam über die bereits so vertrauten Web-Seiten hinaus, aber noch hat sich die Realisierung des E-Kommerzes (elektronischer Kommerz) im Internet nicht wirklich durchgesetzt. Mit E-Kommerz meine ich nicht nur die Möglichkeit, Dinge von einer Web-Seite aus zu bestellen, wie beispielsweise Bücher, Kassetten, CDs oder Software. Diese Art Kommerz gibt es schon seit mehreren Jahren, und einige Unternehmen – insbesondere Amazon.com – haben große Erfolge damit erzielen können. Meine Defini-

tion von E-Kommerz geht viel tiefer. In den letzten Jahren haben sich verschiedene neue Initiativen gegründet, die die Perspektive vieler Unternehmen in Hinblick auf das Web ändern werden. Unter anderem geht es dabei um die folgenden Dinge:

- Nutzung des Internet, um die verschiedenen Niederlassungen verteilter Unternehmen zu einer Einheit zu verbinden.
- Nutzung des Internet für den Informationsaustausch über Finanztransaktionen (Kreditkarten, Überweisungen usw.).
- Austausch von medizinischen Transaktionsdaten zwischen Patienten, Krankenhäusern, Ärzten und Versicherungsgesellschaften über das Internet.
- Verteilung von Software über das Web, unter anderem mit der Möglichkeit, Software zu entwickeln, für die keine Installation mehr nötig ist, und die einzelnen Funktionen in riesigen Programmen wie Microsoft Word zu modularisieren, so daß Sie nur die Teile laden, nutzen und bezahlen, die Sie wirklich brauchen.

Immer wenn Sie eine Web-Site besuchen, die Java, JavaScript oder eine andere Scripting-Sprache unterstützt, führen Sie letztlich ein Programm über das Web aus. Nachdem Sie damit fertig sind, befinden sich im Cache Ihres Web-Browsers sehr wahrscheintlich nur noch ein paar Codeabschnitte. Mehrere Softwareunternehmen – unter anderem Microsoft – haben vor, ihre Software auf diese Weise zu veröffentlichen. Damit erzielen sie ständige Gewinne mit ihrer Software, und Sie profitieren davon, weil Sie nur die Software zahlen, die Sie genutzt haben, und nur für die Zeitdauer, in der Sie sie benötigten.

Während die meisten dieser Anwendungen mit *HTML* (*Hypertext Markup Language*) nicht möglich sind, macht XML all das (und noch mehr) zur Realität. Man könnte fast sagen, XML ist die Technologie, die uns zu einer völlig neuen Form der Internet-Gesellschaft führt. XML ist wahrscheinlich das wichtigste, was dem Web seit der Einführung von Java passiert ist.

Warum aber kann XML Dinge, die HTML nicht kann? Lesen Sie einfach weiter.

Wo HTML nicht mehr reicht

Bevor wir hier die Schwächen von HTML darlegen, wollen wir eines klarstellen: HTML war und ist immer noch eine phantastische Sache.

Als einfache Tag-Sprache für die Anzeige von Text in einem Web-Browser entworfen, hat HTML seine Arbeit großartig gemeistert und wird das sicher auch in den nächsten Jahren noch tun. Ohne Übertreibung kann man sagen, es gäbe gar kein Web, wenn

es kein HTML gäbe. Vor HTML gab es zwar Gopher, WAIS und Hytelnet, aber nichts davon bot dieselbe Kombination aus Leistungsfähigkeit und Einfachheit wie HTML.

HTML wird zum Teil immer noch als die Killer-Internet-Anwendung betrachtet, aber viele Unternehmen beschweren sich darüber. Darüber hinaus erkennen die Leute bereits, daß XML HTML überlegen ist. Hier folgen einige der häufigsten Beschwerden über HTML (aber viele davon sind nicht wirklich berechtigt, wie Sie meinen Kommentaren entnehmen können):

- HTML bietet keine Syntaxprüfung: Es ist nicht möglich, den HTML-Code zu überprüfen.

Ja und nein. Es gibt formale Definitionen für die Struktur von HTML-Dokumenten – wie Sie später noch erfahren werden, ist HTML eine SGML-Anwendung, und es gibt für jede HTML-Version eine Dokumenttypdefinition (DTD).

Die *Dokumenttypdefinition* (*DTD*) ist ein SGML- oder XML-Dokument, das die Elemente und Attribute beschreibt, die innerhalb aller Dokumente erlaubt sind, die konform mit dieser DTD sind. Sie werden in späteren Kapiteln alles über XML DTDs erfahren.

Außerdem gibt es bereits einige Werkzeuge (und eine oder zwei Web-Sites), die eine Syntaxprüfung von HTML-Dokumenten unterstützen. Damit tritt die Frage auf, warum nicht mehr Leute ihre HTML-Dokumente prüfen; die Antwort ist, daß die Prüfung etwas irreführend ist. Web-Browser werden so entwickelt, daß sie fast alles akzeptieren, was auch nur im entferntesten wie HTML aussieht (wodurch das Risiko besteht, daß die Anzeige nicht mehr so erscheint wie erwartet – aber das ist wieder eine andere Geschichte). Seltsamerweise ist das einzige Tag, das in einem HTML-Dokument zwingend erforderlich ist, das `TITLE`-Tag; und noch verwunderlicher ist, daß dies gerade eines der am wenigsten gebräuchlichsten Tags ist.

- HTML hat keine Struktur.

Das stimmt nicht wirklich. HTML hat ordnungsgemäße Überschriftstags (`H1` bis `H6`), und Sie können Informationsblöcke innerhalb von `DIV`-Tags verschachteln. Browsern ist es egal, in welcher Reihenfolge Sie diese Überschriften verwenden, und häufig wählt man sie einfach nur ihrer Schriftgröße nach aus. Das Problem liegt also darin, wie der HTML-Code genutzt wird.

- HTML ist nicht inhaltsbezogen.

Ja und nein. Das Durchsuchen des Web ist kompliziert, weil Ihnen HTML keine Möglichkeit bietet, den Informationsinhalt – die Semantik – Ihrer Dokumente zu beschreiben. In XML können Sie beliebige Tags verwenden (beispielsweise `<NAME>` statt `<H3>`), aber die Verwendung von Attributen innerhalb des Tags (wie etwa `<H3 CLASS="name">`) kann ebensoviel semantische Information bereitstellen wie benutzerdefinierte

Tags. Ohne standardisierte Tag-Namen wird der Nutzen benutzerdefinierter Tags eher zweifelhaft sein. Um das Ganze noch schlimmer zu machen, kann derselbe Tag-Name innerhalb eines Kontexts etwas ganz anderes bedeuten als in einem anderen Kontext. Darüber hinaus gibt es Komplikationen mit Fremdsprachen – wenn Sie ein <inkoo-prijs> sehen, wird Ihnen das nicht sehr viel weiterhelfen, es sei denn, Sie wissen, daß dies das niederländische Wort für »Einkaufspreis« ist.

- HTML ist nicht international.

Das stimmt fast. Es gab einige Anläufe, HTML zu internationalisieren, und ihm insbesondere eine Möglichkeit bereitzustellen, die innerhalb eines Tags verwendete Sprache zu erkennen.

- HTML ist nicht für den Datenaustausch geeignet.

Fast richtig. Die Tags von HTML können nur wenig dazu beitragen, die in einem Dokument enthaltene Information zu identifizieren.

- HTML ist nicht objektorientiert.

Richtig. Die modernen Programmierer haben sich endlich an den schwierigen Umstieg auf die objektorientierten Techniken gewöhnt. Sie wollen diese Fähigkeiten nun anwenden und Dinge wie die Vererbung nutzen – womit HTML kaum dienen kann.

- HTML hat keinen robusten Link-Mechanismus.

Sehr richtig. Wenn Sie schon mehrere Stunden im Web verbracht haben, dann ist Ihnen dabei mindestens einmal eine nicht mehr funktionale Verknüpfung begegnet. Fehlerhafte Verknüpfungen sind der Fluch der Web-Manager auf der gesamten Welt, aber man kann nur wenig tun, um sie zu verhindern. Die Links in HTML sind größtenteils 1:1, wobei die Verknüpfung in den Quell-HTML-Dateien fest codiert ist. Ändert sich die Position einer der Zieldateien, muß ein Webmaster womöglich Dutzende oder sogar Hunderte anderer Seiten aktualisieren.

- HMML ist nicht wiederverwendbar.

Richtig. Abhängig davon, wie gut sie geschrieben sind, kann es extrem schwierig sein, HTML-Seiten und HTML-Codeabschnitte wiederzuverwenden, weil sie speziell auf ihren Platz im Web und die damit verbundenen Seiten zugeschnitten sind.

- HTML ist nicht erweiterbar.

Richtig, aber unfair. Man könnte genausogut sagen, ein Auto ist ein besseres Kraftfahrzeug als ein Fahrrad. Es war *nie* geplant, daß HTML erweiterbar sein sollte.

Wo liegt also das Problem bei HTML? Für die ganz normale Nutzung von Web-Seiten gibt es kein Problem. Aber wenn man an die Zukunft des elektronischen Kommerzes im Web denkt, gerät HTML sehr schnell an seine Grenzen.

Wo ist das Problem mit ...?

Wenn also HTML es nicht verarbeiten kann, was stimmt dann mit TeX, PDF oder RTF nicht?

TeX ist eine Sprache für den Computersatz, die in wissenschaftlichen Gemeinden immer noch stark verbreitet ist. Anfang der 80er Jahre gab es Online-Datenbanken, die die Daten in TeX-Format zurückgaben, so daß sie direkt in ein TeX-Dokument eingesetzt werden konnten. Adobe besitzt den *PDF*-Standard (Adobe Acrobat), der jedoch gut dokumentiert ist. Alle diese Formate leiden unter derselben Schwäche: Sie sind proprietär (gehören also einer kommerziellen Firma oder Organisation), sie sind nicht offen, und sie sind nicht standardisiert. Wenn man eines dieser Formate nutzt, riskiert man, plötzlich ohne jede Unterstützung dazustehen. Der Markt bildet zwar eine strenge stabilisierende Kraft (wie man bei RTF gesehen hat), aber wenn Sie zu viel in ein einziges Format investieren, über das Sie keine Kontrolle haben und in das Sie relativ wenig Einsicht besitzen, riskieren Sie Probleme für den Fall, daß sich dieses Format ändert.

SGML

Ich werde versuchen, nicht all mein Wissen über SGML aufzuschreiben. Es kann zwar sinnvoll sein, Informationen darüber zu besitzen, aber es gibt Situationen, wo man am besten gar nichts darüber weiß. Wenn Sie zu viel über SGML lernen, müssen Sie, wenn Sie zu XML wechseln, viel Zeit darauf verwenden, Dinge zu vergessen, die Sie gerade gelernt haben. XML unterscheidet sich so wesentlich von SGML, daß Sie XML-Experte werden können, ohne SGML überhaupt zu kennen.

Nachdem Sie das wissen, kann ich Ihnen sagen, daß XML ein Nachfolger von SGML ist, und wenn Sie ein bißchen über SGML Bescheid wissen, wird Ihnen das helfen, XML zu verstehen.

SGML (*Standard Generalized Markup Language*), von dem sich XML ableitet, entstand aus dem Bedarf, die Datenspeicherung unabhängig von Softwarepaketen oder Softwareherstellern zu machen. SGML ist eine *Metasprache*, eine Sprache für die Beschreibung von Markup-Sprachen. HTML ist eine solche Markup-Sprache und wird deshalb als SGML-Anwendung bezeichnet. Es gibt Dutzende, vielleicht sogar Hunderte von Markup-Sprachen, die unter Verwendung von SGML definiert wurden. In XML werden diese Anwendungen häufig als Markup-Sprachen bezeichnet, wie beispielsweise *HDML* (*Hand-Held Device Markup Language*) oder *QML* (*FAQ Markup Language*).

In SGML haben die meisten Markup-Sprachen keine formalen Namen; sie werden einfach nach dem Namen ihrer Dokumenttypdefinition bezeichnet (DocBook), nach ihrer Aufgabe (LinusDOC), ihrer Nutzung (TEI) oder sogar nach dem Standard, den sie implementieren (J2008-Automobilteile, Mil-M-38784-US Military).

Mit Hilfe einer SGML-Deklaration (XML hat auch eine) gibt die SGML-Anwendung an, welche Zeichen als Daten und welche als Markup interpretiert werden sollen. (Sie müssen dabei nicht die bekannten Zeichen < und > enthalten; in SGML könnten statt dessen auch einfach { und } benutzt werden.)

Unter Verwendung der in der SGML-Deklaration vorgegebenen Regeln und der Ergebnisse der Informationsanalyse (die letztlich zu etwas führt, was man ganz einfach als Informationsmodell betrachten kann) legt der SGML-Anwendungsentwickler die verschiedenen Dokumenttypen – wie etwa Berichte, Broschüren, technische Handbücher usw. – an und entwickelt für jeden davon eine DTD. Durch Verwendung der festgelegten Zeichen identifiziert die DTD Informationsobjekte (Elemente) und ihre Eigenschaften (Attribute).

Die DTD ist die Seele einer SGML-Anwendung; wie gut sie ist, bestimmt zu einem großen Teil den Erfolg oder das Scheitern der gesamten Aktivität. Unter Verwendung der in der DTD festgelegten Informationselemente wird die eigentliche Information dann mit Hilfe der dafür in der Anwendung festgelegten Tags gekennzeichnet. Wenn die Entwicklung der DTD sehr voreilig und unüberlegt passiert ist, muß sie möglicherweise immer wieder verbessert, angepaßt oder korrigiert werden. Bei jeder Änderung der DTD muß auch die Information, die damit gekennzeichnet wurde, geändert werden, weil sie möglicherweise nicht mehr korrekt ist. Sehr schnell kann die Menge der Daten, die dabei abgeändert werden müssen (heute auch als *Legacy-Daten* bezeichnet), schnell zu einem ernsthaften Problem werden – das teurer und zeitaufwendiger ist als das Problem, das die SGML ursprünglich lösen sollte.

Sie haben damit bereits ein Gefühl für die Dimension einer SGML-Anwendung. Es gibt viele Gründe, warum es so umfangreich ist: SGML wurde so entwickelt, daß es dauerhaft ist. Seine Entwickler hatten im Hinterkopf die Idee der Langlebigkeit und Dauerhaftigkeit, ebenso wie das Ziel, Daten vor zukünftigen Änderungen an der verwendeten Soft- und Hardware zu schützen.

SGML ist die Industrielösung: teuer und kompliziert, aber auch extrem leistungsfähig.

Warum nicht SGML?

Das SGML der Web-Initiative gab es lange bevor XML überhaupt in Betracht gezogen wurde. Irgendwie war es aber nie wirklich erfolgreich. Grundsätzlich ist SGML zu teuer und zu kompliziert für den allgemeinen Einsatz im Web. Das heißt nicht, daß es nicht benutzt werden *kann* – es *wird* nicht benutzt. Der Einsatz von SGML fordert zu viele Investitionen in Hinblick auf Zeit, Werkzeuge und Training.

Warum XML?

XML stellt die Funktionen von SGML bereit, die es braucht, und versucht, die aus HTML bereits bekannten Dinge zu nutzen.

Eine der wichtigsten Ähnlichkeiten zwischen XML und SGML ist, daß XML eine DTD verwendet. In Kapitel 17 erfahren Sie mehr über die aktuellen Entwicklungen, die diese wichtige Verbindung zu SGML kappen und die DTD durch etwas ersetzen wollen, was den Anforderungen der Datenverarbeitung von XML-Anwendungen besser entspricht.

Als sich die Entwickler von XML hinsetzten, um seine Spezifikationen festzulegen, hatten sie mehrere Entwurfsziele im Kopf (die im Empfehlungsdokument detailliert aufgeführt sind). Diese Ziele und der Grad, zu dem sie bereits erfüllt sind, sind der Grund dafür, warum man XML für besser als SGML hält:

- XML kann mit bereits existierenden Web-Protokollen (wie etwa HTTP und MIME) und Mechanismen (wie URLs) eingesetzt werden und stellt keine zusätzlichen Anforderungen. XML wurde in Hinblick auf das Web entwickelt – Funktionen von SGML, die zu schwierig für die Verwendung im Web waren, wurden hier einfach weggelassen, und Funktionen, die für die Verwendung im Web benötigt wurden, wurden hinzugefügt oder von bereits funktionierenden Anwendungen geerbt.
- XML unterstützt eine Vielzahl von Anwendungen. Es ist schwierig, allein mit HTML viele Anwendungen zu unterstützen; deshalb auch die rasche Entstehung der Scripting-Sprachen. HTML ist einfach zu spezifisch. XML übernimmt die generische Natur von SGML, ergänzt es aber um die Flexibilität, so daß es wirklich erweiterbar wird.
- Es ist einfach, Programme zur Verarbeitung von XML-Dokumenten zu schreiben. Eine der wichtigsten Stärken von HTML ist, daß es selbst für einen Nichtprogrammierer ganz einfach ist, ein paar Zeilen Scripting-Code zu entwerfen, mit dem man bereits eine grundlegende Verarbeitung durchführen kann (und es gibt heute bereits eine erstaunliche Anzahl an Scripting-Sprachen). HTML beinhaltet sogar einige eigene Funktionen, die Ihnen ermöglichen, eine grundlegende Verarbeitung durchzuführen (wie beispielsweise Formulare und CGI-Abfragezeichenketten). XML hat von dem Erfolg von HTML gelernt und versucht, so einfach wie möglich zu bleiben, indem viele der komplexeren Funktionen von SGML einfach verworfen wurden. Es gibt bereits XML-Verarbeitungsanwendungen in Java, SmallTalk, C, C++, JavaScript, Tcl, Perl und Python, um nur ein paar wenige zu nennen.
- Die Anzahl der optionalen Funktionen in XML wurde auf einem absoluten Minimum gehalten. SGML beinhaltet viele optionale Funktionen, die SGML-Software muß sie also alle unterstützen. Man könnte argumentieren, daß es nicht ein einzi-

ges Softwarepaket gibt, das alle SGML-Funktionen unterstützt (und man kann sich schwerlich eine Anwendung vorstellen, in der sie auch alle gebraucht werden). Diese hohe Leistungsfähigkeit führt sofort zur Komplexität, was gleichzeitig Größe, Kosten und Schwerfälligkeit bedeutet. Die Geschwindigkeit des Web ist bereits zu einem wichtigen Aspekt geworden; es ist schlimm genug, auf den Download eines Dokuments warten zu müssen, aber wenn man für seine Verarbeitung noch einmal Lichtjahre warten müßte, wäre XML vom Anfang an zum Scheitern verurteilt.

- XML-Dokumente sind selbst dem Laien relativ verständlich. Es wird zwar immer seltener und auch schwieriger, HTML-Dokumente manuell einzutippen, und XML-Dokumnete sollten auch gar nicht von Menschen erzeugt werden, aber dies bleibt ein wichtiger Aspekt. Die Maschinencodierung ist im Hinblick auf Langlebigkeit und Portierbarkeit begrenzt, und häufig ist das Ergebnis an das System gebunden, auf dem sie erzeugt wurde. Die Markup-Syntax von XML ist gut verständlich.

  Wenn Sie die Zeit haben, können Sie jedes XML-Dokument ausdrucken und seine Bedeutung erkennen – aber es geht noch viel weiter. Ein gültiges XML-Dokument

  - beschreibt sämtliche Regeln, denen die Markup-Syntax folgen soll.
  - listet alle externe Ressourcen (externe Entities) auf, die Teil des Dokuments sind.
  - deklariert alle internen Ressourcen (interne Entities), die innerhalb des Dokuments verwendet werden.
  - listet die Typen der Nicht-XML-Ressourcen (Notationen) auf und bezeichnet alle Hilfsanwendungen, die möglicherweise benötigt werden.
  - listet alle Nicht-XML-Ressourcen (Binärdateien) auf, die innerhalb des Dokuments verwendet werden, und bezeichnet alle Hilfsanwendungen, die möglicherweise benötigt werden.

- Der Entwurf von XML ist formal und knapp. Als Grundlage der XML-Spezifikation wurde das *Extended Backus-Naur Format* (*EBNF*) verwendet (eine Methode, die die meisten Programmierer kennen). Informationen, die in XML gekennzeichnet sind, können von Computerprogrammen einfach verarbeitet werden. Noch besser verwendet man dazu ein System, das den Computerprogrammierern vertraut ist, und da es fast völlig frei von Mehrdeutigkeiten ist, ist es relativ einfach für Programmierer, Programme zu entwickeln, die mit XML arbeiten.
- XML-Dokumente sind einfach anzulegen. HTML ist geradezu berühmt für seine einfache Handhabung, und XML nutzt diese Stärke aus. Es ist nämlich sogar einfacher, ein XML-Dokument anzulegen als ein HTML-Dokument. Schließlich müssen Sie überhaupt keine Markup-Tags lernen – Sie können Ihre eigenen erzeugen!

Wie XML SGML und HTML ergänzt

XML nimmt das beste aus SGML, kombiniert es mit einigen der besten Funktionen von HTML und fügt einige Funktionen der erfolgreichsten Anwendungen von beidem ein. XML übernimmt sein Hauptgerüst aus SGML und läßt alles weg, was nicht mehr unbedingt nötig ist. Alle Funktionen und Eigenschaften wurden genau untersucht, und wenn es keinen wirklich guten Grund gab, sie beizubehalten, wurden sie verworfen. XML wird häufig auch als *Untermenge* von SGML bezeichnet, aber technisch betrachtet, handelt es sich dabei um ein *Unterprofil* von SGML; während HTML SGML benutzt und eine Anwendung von SGML ist, ist XML eigentlich SGML – in kleinerem Umfang.

Von HTML erbt XML die Verwendung der Web-Adressen (URLs), die auf andere Objekte verweisen. Von *HyTime* (einer sehr komplexen Anwendung von SGML, offiziell als *ISO/IEC 10744 Hypermedia/Time-based Structuring Language* bezeichnet) und einer akademischen Anwendung von SGML, *TEI* (*Text Encoding Initiative*) erbt XML einige andere extrem leistungsfähige Adressierungsmechanismen, die Ihnen erlauben, auf Teile und Bereiche anderer Dokumente zu verweisen statt beispielsweise auf einfache Ziele.

XML fügt außerdem mehrere Funktionen ein, die es für die Verwendung im immer komplexer und differenzierter werdenden Web besser geeignet machen als SGML oder HTML.

- **Modularität** – Obwohl HTML scheinbar keine DTD hat, gibt es eine implizite DTD, die in den Web-Browsern codiert ist. SGML andererseits unterstützt eine unbegrenzte Anzahl von DTDs, aber es gibt für jeden Dokumenttyp nur eine. XML ermöglicht Ihnen, die DTD völlig wegzulassen oder unter Verwendung komplexer Auflösungsmechanismen mehrere Teile von XML-Instanzen oder separater DTDs zu einer zusammengesetzten Instanz zu kombinieren.
- **Erweiterbarkeit –** Die leistungsfähigen Linking-Mechanismen erlauben Ihnen, Links einzufügen, ohne daß das Link-Ziel physisch im Objekt vorhanden sein muß. Damit eröffnen sich ungeahnte Möglichkeiten für die Verknüpfung von Dingen, auf die man keinen Schreibzugriff hat, auf CD-ROMs, auf Bibliothekskataloge, die Ergebnisse von Datenbankabfragen oder sogar andere Medien, wie beispielsweise Soundtracks oder Ausschnitte aus Videofilmen. Darüber hinaus wird es möglich, die Links unabhängig von den Objekten, die sie verknüpfen, zu speichern (vielleicht sogar in einer Datenbank, so daß die Linklisten gemäß dem dynamischen Inhalt der Dokumentensammlung automatisch erzeugt werden können). Damit wird eine langfristige Link-Verwaltung möglich.
- **Verteilung –** Neben den Link-Mechanismen führt XML auch eine viel komplexere Methode zur Aufnahme von Link-Instanzen in die aktuelle Instanz ein. Damit öffnen sich die Türen zu einer neuen Welt der *zusammengesetzten Dokumente*

– Dokumente, die sich aus Teilen anderer Dokumente zusammensetzen, die automatisch (und transparent) zusammengesetzt werden, um genau das zu bilden, was zu dem jeweiligen Zeitpunkt angezeigt werden soll. Der Inhalt kann unmittelbar auf den Moment, das Medium und den Leser zugeschnitten werden und besitzt nur flüchtige Existenz: eine virtuelle Informationsrealität, die sich aus virtuellen Dokumenten zusammensetzt.

- **Internationalität –** Sowohl HTML als auch SGML basieren auf ASCII, wodurch die Verwendung von Zeichen aus anderen Sprachen relativ schwierig wird. XML basiert auf Unicode und bedingt, daß die gesamte XML-Software ebenso Unicode unterstützt. Unicode erlaubt es XML, nicht nur die Zeichen der westlichen Welt darzustellen, sondern auch asiatische Sprachen zu unterstützen. (In Kapitel 8 erfahren Sie mehr über Zeichensätze und die Zeichencodierung.)
- **Datenorientierung –** XML arbeitet mit Datenorientierung und nicht nach der Lesbarkeit für den Menschen. Die Möglichkeit, daß der Mensch alles lesen kann, ist eines der Entwurfsziele von XML, aber der E-Kommerz bedingt ein Datenformat, das auch von Maschinen gelesen werden kann. XML macht das möglich, indem eine Form von XML definiert wird, die einfacher von einer Maschine erzeugt werden kann, fügt jedoch durch die neuesten XML-Schema-Initiativen auch eine strengere Datensteuerung ein.

Ist XML nur für Programmierer vorgesehen?

Nachdem Sie bis hierhin gelesen haben, denken Sie womöglich, daß XML nur für Programmierer gedacht ist, und daß Sie am besten weiterhin HTML verwenden. Da haben Sie nicht ganz unrecht, bis auf einen wichtigen Aspekt: Wenn Programmierer mit XML mehr tun können, dann können sie das auch mit HTML, und irgendwann pflanzt sich das bis zu Ihnen fort, in Form von Anwendungssoftware, die Sie für Ihre XML-Daten einsetzen können. Um diese Werkzeuge jedoch wirklich nutzen zu können, müssen Sie Ihre Daten in XML aufbereiten. Noch ist die Unterstützung von XML in Web-Browsern unvollständig und unzuverlässig (Sie werden später lernen, wie XML-Code in Mozilla und dem Internet Explorer 5 dargestellt wird), aber die vollständige Unterstützung wird nicht lange auf sich warten lassen.

Sollen in der Zwischenzeit nur Programmierer XML nutzen? Sicher nicht! Eines der Probleme bei HTML ist, daß alle Tags optional sind, so daß Sie mit ihnen wirklich vertraut sein müssen, um eine sinnvolle Auswahl zu treffen. Und was noch schlimmer ist, Ihre Auswahl wird dadurch beeinflußt, wie der Code in einem bestimmten Browser aussieht. XML dagegen ist erweiterbar, und die Erweiterbarkeit funktioniert in beide Richtungen – es bedeutet auch, daß Sie weniger statt mehr benutzen können. Statt mehr als 40 HTML-Tags lernen zu müssen, können Sie Ihren Text so kennzeichnen,

wie es für Sie selbst viel einfacher ist, und dann ein Stylesheet verwenden, um die sichtbare Darstellung festzulegen. Listing 1.1 zeigt ein typisches XML-Dokument, das einen ganz typischen Eintrag für einen Verkaufskontakt kennzeichnet.

Listing 1.1: Beispiel für ein XML-Dokument

```
1: <?xml version="1.0"?>
2:   <contacts>
3:     <contact>
4:       <name>
5:         <first>John</first>
6:         <last>Belcher</last>
7:       </name>
8:       <address>
9:         <street>Pennington 13322</street>
10:        <city>Washington</city>
11:        <state>DC</state>
12:        <zip>66522</zip>
13:      </address>
14:      <tel>555 1276</tel>
15:      <fax>555 9983</fax>
16:      <mobile>887 8887 7777</mobile>
17:      <email>jb@southside.com</email>
18:    </contact>
19:  </contacts>
```

Wie Listing 1.1 zeigt, können Sie in Ihre Kennzeichnungen umfassende Informationen aufnehmen (semantischer Inhalt). Das Praktische bei XML ist, daß Sie es genau Ihren Bedürfnissen anpassen können. Wenn Sie weniger brauchen, dann nutzen Sie eben weniger, wie in Listing 1.2 gezeigt. (Wir müssen ganz einfach mit all den anderen Computerbüchern der Welt gleichziehen und mindestens einmal das »Hello World«-Beispiel bringen.)

Listing 1.2: »Hello World« in XML

```
1: <?xml version="1.0"?>
2:   <greeting>
3:     <salutation>Hello</salutation>
4:     <target>World!</target>
5:   </greeting>
```

XML ist bereits auf dem besten Weg, zur bevorzugten Sprache für die Bereitstellung einer Schnittstelle zwischen Datenbanken und dem Web zu werden, und es wird zu einer wichtigen Methode für den Datenaustausch zwischen Computern und Computeranwendungen. Auf der Ebene des »normalen« Web-Dokument-Authorings hat XML

jedoch noch viel zu bieten. Sie können bestimmen, welche und wie viele Tags Sie brauchen, und Sie geben den Tags die Namen, die für Sie oder Ihre Leser einen ausreichenden Informationsgehalt bieten. Statt Dokumente mit nichtssagenden Auszeichnungen wie `H1`, `H2`, `P`, `LI`, `UL` und `EM`-Tags zu produzieren, können Sie sagen, was Sie meinen, und dafür die Namen `KAPITEL`, `ABSCHNITT`, `ABSATZ`, `LISTEN.ELEMENT`, `NICHT-NUMERIERTE.LISTE` und `WICHTIG` verwenden. Das macht Ihre Dokumente nicht sinnvoller, aber andere Menschen können sie einfacher verstehen. Werkzeuge (wie beispielsweise Suchmaschinen) sind in der Lage, intelligentere Abfragen zum Inhalt und zur Struktur Ihrer Dokumente auszuführen und sinnvolle Schlußfolgerungen dazu zu erlauben, die weit über Ihre ursprünglichen Absichten hinausgehen.

Zusammenfassung

Dieser erste Tag hat eine abstrakte Beschreibung von XML als Markup-Sprache geboten. Sie sahen, warum man XML für das schnell wachsende Internet und seine kommerziellen Anwendungen braucht. Außerdem haben Sie einen sehr kurzen Überblick darüber erhalten, warum man XML als Lösung für die Veröffentlichung von Text und Daten über das Internet bevorzugt, statt SGML oder HTML zu verwenden.

So wie Medizinstudenten ihre Ausbildung mit dem Sezieren von Körpern beginnen, werden Sie morgen ein XML-Dokument zerlegen, um seine Anatomie und die dafür verwendeten Komponenten kennenzulernen.

F&A

F Ist XML ein Standard, auf den ich mich verlassen kann?

A *XML wird von einer Gruppe von Herstellern empfohlen, unter anderem von Microsoft und Sun, die man als das World Wide Web Consortium (W3C) bezeichnet. Das ist soviel oder sowenig ein Standard wie alles im Web. Das W3C hat sich selbst darauf geeinigt, XML in all seinen anderen Initiativen zu unterstützen. In den normalen Standardisierungsgremien wird der SGML-Standard aktualisiert, so daß XML auf die Unterstützung und Formalität von SGML vertrauen kann.*

F Muß ich SGML lernen, um XML verstehen zu können?

A *Nein. Es hilft vielleicht bei höchst technischen XML-Entwicklungen, wenn Sie bereits ein bißchen SGML beherrschen, aber für die meisten XML-Anwendungen brauchen Sie keine SGML-Kenntnisse.*

F Ich kenne SGML; wie schwierig ist es für mich, XML zu lernen?

A *Wenn Sie bereits Erfahrungen mit SGML haben, dauert es weniger als einen Tag, Ihr Wissen für XML anzupassen und alles weitere zu lernen. Sie brauchen jedoch die Disziplin, umzulernen, weil XML manche Dinge anders macht als SGML.*

F Ich kenne HTML; wie schwierig ist es für mich, XML zu lernen?

A *Das hängt davon ab, wie gut Sie HTML kennen und was Sie mit XML vorhaben. Falls Sie nur Web-Seiten erzeugen wollen, können Sie die Grundlagen möglicherweise in ein oder zwei Tagen erlernen.*

F Ersetzt XML SGML?

A *Nein. SGML wird in großen Anwendungen weiterhin eingesetzt, wo viele seiner Funktionen benötigt werden. XML wird einige der Aufgaben von SGML übernehmen, es aber nie ersetzen.*

F Ersetzt XML HTML?

A *Irgendwann sicher. HTML hat bisher eine wunderbare Arbeit geleistet, und es gibt Grund genug zur Annahme, daß das auch weiterhin der Fall sein wird. Irgendwann wird HTML jedoch als XML-Anwendung statt als SGML-Anwendung neu geschrieben – aber Sie werden den Unterschied nicht erkennen.*

F Ich habe sehr viel HTML-Code; sollte ich ihn in XML umwandeln? Und falls ja, wie?

A *Nein. Existierender HTML-Code kann sehr leicht in XML-Syntax dargestellt werden. Es ist auch möglich, HTML-Code in XML-Dokumente aufzunehmen und umgekehrt. Es ist jedoch nicht ganz so einfach, eine HTML-Authoring-Umgebung in eine XML-Umgebung umzuwandeln. Heute gibt es keine XML-DTDs für HTML. Bis es sie gibt, ist es einfacher, den HTML-Code mit HTML- (oder SGML-) Werkzeugen zu erzeugen, statt den fertigen Code zu konvertieren.*

Übung

Sie haben bereits gesehen, wie ein grundlegendes XML-Dokument aussieht. Legen Sie auf dieselbe Weise ein Dokument an, das Sie im Web darstellen möchten (etwas Persönliches, wie beispielsweise eine Homepage oder die Verwaltung von Spuren einer CD).

Die Anatomie eines XML-Dokuments

Woche 1

So wie Medizinstudenten beim Sezieren menschlicher Körper die Organe in ihrer Konsistenz und ihrem Zusammenspiel kennenlernen, bevor sie erfahren, wie man sie behandelt, beginnt unsere Erkundung von XML mit der Betrachtung eines kleinen XML-Dokuments, dessen einzelne Komponenten erklärt werden. Heute werden Sie

- ein kleines XML-Dokument und seine Komponenten kennenlernen
- den Unterschied zwischen Markup- und Zeichendaten erfahren
- logische und physische Strukturen betrachten

Markup

Bevor wir unter die Oberfläche von XML gehen, wollen wir einen kleinen Schritt zurückgehen und eines der grundlegendsten Konzepte betrachten – Markup. Gestern haben Sie von einigen Markup-Sprachen gehört und einige Details über die Eigenschaften einiger Implementierungen kennengelernt, etwa am Beispiel von TeX, aber was genau ist Markup?

Einfach gesagt, durch das *Markup* werden einer Information Zeichen hinzugefügt, die genutzt werden, um diese Information auf eine ganz bestimmte Weise zu verarbeiten. Am unteren Ende der Skala könnte das etwas so Einfaches wie das Einfügen von Kommas zwischen verschiedenen Datenabschnitten sein, die beim Importieren dieser Information in ein Datenbankprogramm als Feldtrennzeichen interpretiert werden. Es kann aber auch so komplex sein wie eine extrem umfangreiche Metasprache, wie etwa die TEI (Text Encoding Initiative) SGML DTD. Die TEI DTD macht es möglich, die Transkriptionen historischer Dokumentmanuskripte so zu kennzeichnen, daß die jeweilige Version, die Übersetzung, die Interpretation, Kommentare zum Inhalt und sogar eine ganze Bibliothek zusätzlicher Informationen kenntlich gemacht wird, die für die akademische Arbeit zu diesem Manuskript genutzt werden kann.

Was genau zu diesen Markierungen, dem Markup, gehört, und was nicht, muß die *Interpreter-Software* erkennen. Vergleichen Sie den WordPerfect-Code für einen einzigen Satz in Listing 2.1 mit demselben Satz im Microsoft RTF-Format, das Sie in Listing 2.2 sehen.

Listing 2.1: WordPerfect-Code

```
ŸWPC)_  _  °Ÿ_ 2 x_  _ _   B   _ (   J   _ ¨   r    Z   -_  _ ¦ x
Times New Roman  (TT) Courier New  (TT)  _ X___Ù_C Ù_\__  _ Pé ”6Q___
_ _ Ù___Ù_d ?_6_X_   __@… DèQ___
°?_ 2 C_  _ <   ™_  _ [   Ê_  ??    A_  _ _   A_
Ÿ__ ?ÙÈ?phoenix _   _ _Ÿ-_   Ù_C Ù_\__           _ Pé ”6Q___
__  _- Im einfachsten Fall ist Markup einfach,
```

```
Simon North Simon North    °?_ 2     __¥   u_
Default Paragraph Fo _Default Paragraph Font  _.   .
-_   X_P Ù_\__  _ Pé ”6Q___ __  _-”__  __ _””__ _
_ _”-_   Ù_C Ù_\__  _ Pé ”6Q___ __
_--_ 8_8_<_<_ _--__ _ _ _-”__
__ _”Im einfachsten Fall ist Markup einfach,
bestimmte Zeichen in einen Informationsabschnitt einzufügen.-_
Ù_d ?_6_X_   __@… DèQ_____  _-
```

Listing 2.2: Microsoft Word RTF-Code

```
{\rtf1\ansi\ansicpg1252\uc1 \deff0\deflang1033
\deflangfe1033{\fonttbl{\f0\froman\fcharset0\fprq2
{\*\panose 02020603050405020304}Times New Roman;}
{\f2\fmodern\fcharset0
\fprq1{\*\panose 02070309020205020404}Courier New;}}
{\stylesheet{
\widctlpar\adjustright \fs20\cgrid \snext0 Normal;}
{\*\cs10 \additive Default Paragraph Font;}}{\info{\title
simple}{\edmins1}{\nofpages1}{\nofwords0}{\nofchars0}{\nofcharsws0}
{\vern89}}
\widowctrl\ftnbj\aenddoc\formshade\viewkind4\viewscale100
\pgbrdrhead\pgbrdrfoot \fet0\sectd \linex0\endnhere\sectdefaultcl
{\*\pnseclvl9\pnlcrm\pnstart1\pnindent720\pnhang{\pntxtb (}
{\pntxta )}}
\pard\plain
\widctlpar\adjustright
\fs20\cgrid {Im einfachsten Fall ist Markup einfach, Zeichen}
{in einen Informationsabschnitt einzufügen.}
{\f2 \par }}
```

Sie könnten jetzt sagen, daß WordPerfect- und RTF-Code eigentlich keine Markups als solche sind, aber genau betrachtet sind sie es doch. Sie sind sicher nicht das, was sich die meisten Leute unter einem Markup vorstellen, und sie sind nicht so leicht lesbar wie das Markup, mit dem Sie es im restlichen Buch zu tun haben werden, aber sie sind ebensosehr Markups wie alle XML-Element-Tags, denen Sie noch begegnen werden. Diese Codes sind eine Form prozeduralen Markups, das genutzt wird, um die Verarbeitung durch eine bestimmte Anwendung zu steuern.

Offensichtlich kann man nicht erwarten, daß der WordPerfect-Code in Microsoft Word genutzt werden kann, und ebenso unsinnig wäre die Annahme, daß WordPerfect mit Microsoft-RTF-Code zurechtkommt (auch wenn beide die jeweils anderen Dokumente importieren können). Diese beiden Beispiele für Markup sind proprietär, und jede Portabilität zwischen den Applikationen sollte als Bonus und weniger als Forderung betrachtet werden.

SGML soll absolut unabhängig von jeder Anwendung sein. Als reines Markup ist es häufig unabhängig, und der SGML-Code, den Sie in einem SGML-Paket erstellen, ist direkt portabel zu jeder anderen SGML-Anwendung. (Sie sind möglicherweise nicht in der Lage, sehr viel damit zu tun, bis Sie nicht Code der lokalen Anwendung eingefügt haben, aber das ist eine andere Geschichte.) Das Leben ist eben nicht ganz so einfach. Innerhalb des Kontexts von SGML hat das Wort Anwendung eine eigene Bedeutung erhalten.

Eine SGML-Anwendung besteht aus einer SGML-Deklaration und einer SGML DTD. Die SGML-Deklaration richtet die grundlegenden Regeln ein, welche Zeichen als Markup-Zeichen erkannt werden sollen und welche nicht. Beispielsweise könnte die SGML-Deklaration festlegen, daß Elemente durch Sterne markiert werden und nicht mit den bekannten spitzen Klammern (*buch* anstelle von <buch>). Die DTD kann alle möglichen zusätzlichen Regeln festlegen, beispielsweise Minimalisierungsregeln, die es erlauben, Markups aus dem Kontext abzuleiten.

Wenn man noch einen Schritt weitergeht, könnte man die Markup-Minimalisierungsregeln und die in der DTD definierten Elementmodelle nutzen, um ein Dokument zu erzeugen, das nur normale englische Wörter enthält. Bei der Verarbeitung (Parsing) durch die SGML-Software würden der Anfang und das Ende der im Dokument enthaltenen Elemente implizit angenommen und so behandelt, als wären sie explizit gekennzeichnet. Vergleichen Sie Listing 2.3, das alle von SGML gebotenen Minimalisierungstechniken in einem einzigen Dokument nutzt (es ist unwahrscheinlich, daß diese Form auch in der Praxis Anwendung findet!), mit Listing 2.4, das denselben Code ohne jede Minimierung zeigt.

Listing 2.3: Tag-Minimierung

```
1:  <p>SGML verwendet Markup zur Kennzeichnung der
2:  <em/logischen/ Dokumentstruktur,
3:  und nicht sein <em/physisches/ Aussehen.
4:  Tags können minimiert werden, indem man
5:  <it/sie wegläßt/,
6:  <it/Abkürzungen nutzt/,
7:  <it/bestimmte Elemente/ und
6:  <it/Daten-Tags/.
7:  <p>und all das kann <b<em/gleichzeitig/</> be
8:  genutzt werden.</p>
```

Listing 2.4: Die nichtminimierte Version

```
1:  <section number="4"><p>SGML verwendet Markup, um die
2:  <em>logische</em> Dokumentstruktur zu kennzeichnen,
3:  und nicht ihr <em>physisches</em> Aussehen.
```

```
4:  Tags können minimiert werden, indem man</p>
5:  <list type="unordered"><li><it>sie wegläßt</it>,</li>
6:  <li><it>und Abkürzungen verwendet</it>,</li>
7:  <li><it>bestimmte Elemente</it> und</li>
6:  <li><it>Daten-Tags</it>.</li></list>
7:  <p>Und all das kann <bold><em>gleichzeitig</em></bold>
8:  verwendet werden.</p></section>
```

Offensichtlich brauchen Sie recht leistungsfähige Software, um diese komplexen Funktionen nutzen zu können, nicht zu reden von der Zeit und dem Aufwand, Code wie diesen schreiben zu lernen. Berücksichtigen Sie außerdem die Divergenzen zwischen SGML-Anwendungen, die dadurch entstehen, noch nicht einmal die verschiedenen Elemente, die man in den unterschiedlichen Anwendungen antrifft – aber es ist damit völlig unwahrscheinlich, daß SGML-Code von einer Anwendung in einer anderen Anwendung genutzt werden könnte. Es wäre ähnlich einer Konvertierung von WordPerfect in Microsoft RTF, außer daß Sie dabei völlig auf sich selbst gestellt wären.

Angesichts dieser Komplexität bietet HTML einen viel einfacheren, praktischeren Ansatz. Betrachten Sie Listing 2.5, wo Sie ein HTML sehen, das absichtlich mit mehreren »Fehlern« durchsetzt ist.

Listing 2.5: Fehlerhafter HTML-Code

```
1: Dies ist eine Pseudo-HTML-Datei. Sie verwendet ein Kleiner- (<) &amp;
2: und ein Größerzeichen (>), die aber nicht als &hf;
3: <em>really</em> Markup verwendet werden..
4: <p>
5: <P>Der Browser sucht nur nach den Tags, die er kennt, so
6: daß er bei einem ihm unbekannten Tag, wie etwa
7: &bsp;<point><font color="red">
8: dieses</font></point> einfach ignoriert.
```

Wenn Sie diesen Text eingeben und ihn in einem Web-Browser anzeigen, sieht die Ausgabe möglicherweise aus wie in Abbildung 2.1 dargestellt. (Es gibt aber keine Garantie, daß Ihr Web-Browser überhaupt eine vergleichbare Anzeige produziert!)

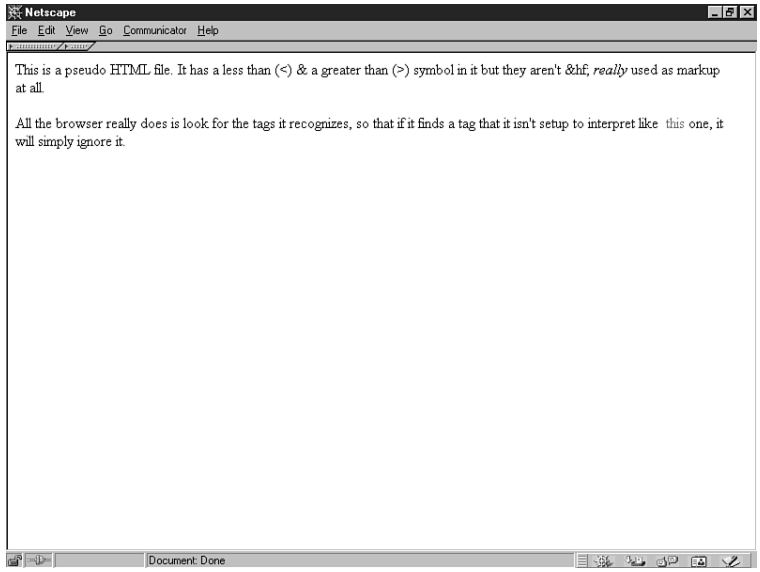


Abbildung 2.1: So zeigt Netscape den fehlerhaften Code aus Listing 2.5 an.

Wie Sie in Abbildung 2.1 sehen, versucht der Web-Browser, aus dem übergebenen Markup irgendeinen Sinn abzuleiten, indem er es mit den Tags vergleicht, die er erwartet. Falls es eine Übereinstimmung gibt, erhalten Sie die entsprechende Ausgabe. Falls es keine Übereinstimmung gibt, das Markup aber als Sonderzeichen oder als speziell zu behandelnder Textbereich (wie etwa das `<point>`-Tag) interpretiert werden soll, ignoriert der Browser es. Wenn das Markup vielleicht gar kein Markup war (die freistehenden Symbole < und >), behandelt der Browser sie als Zeichendaten und zeigt sie als ganz normale Zeichen an.

SGML und HTML stellen Extremfälle dar. Es gibt ein sehr vages Kontinuum zwischen inhaltsorientiertem und darstellungsorientiertem Markup. (Wir werden hier ein wenig vorgreifen, weil das bereits in die DTD-Entwicklung hineinreicht, die viel später im Buch erst erklärt wird.) Das Markup von SGML ist generisch. Hauptsächlich sollte es *inhaltsorientiert* sein – es sagt nichts darüber aus, wie etwas angezeigt werden soll, sondern kennzeichnet statt dessen die Natur oder die Aufgabe der Textabschnitte. Es *kann* für das eine oder auch für beides verwendet werden.

Es gibt viele Bezeichnungen für die verschiedenen Markup-Typen, und es besteht kaum eine Übereinkunft, wie diese Namen korrekt verwendet werden. Der technisch genaueste Begriff für inhaltsorientiertes Markup ist *semantisches Markup*, wobei die Namen der Elemente beschreiben, worum es sich bei diesen Elementen handelt, und nicht, was sie tun, wofür sie verwendet oder wie sie verarbeitet werden sollen. Geht man jedoch tiefer, können Formatanweisungen als eine Art Semantik betrachtet werden; deshalb bleibe ich bei dem möglicherweise weniger exakten, aber doch neutraleren Begriff *inhaltsorientiert*.

Das Markup von HTML ist eine Anwendung von SGML, die rein darstellungsbasiert ist und wirklich wenig mit der in den Dokumenten enthaltenen Information zu tun hat. Das ist aber eigentlich nicht der Fehler von HTML, sondern eher ein Ergebnis seiner lockeren Handhabung. Wäre beispielsweise eine strenge Hierarchie für die Anordnung der Überschriften vorgegeben, so daß `H2` immer nach `H1` kommen müßte, dann wären sie alle korrekt ineinander verschachtelt und HTML könnte mehr inhaltsorientiert sein. Ginge man so weit, Klassen und Typen für alle HTML-Elemente in eine Datei einzufügen, könnte man es in ein sehr ausführliches, inhaltsorientiertes Markup verwandeln. Die Welt ist jedoch nicht perfekt, und jeder mögliche semantische Wert, der durch die Organisation der Überschriftsebenen zu erzielen ist, würde durch den willkürlichen und beliebigen Gebrauch allein der Schriftgröße in einem Browser zunichte gemacht. Falls Sie Zweifel haben, daß HTML rein darstellungsorientiert ist, fragen Sie sich einfach, welche Art semantischer Information das <HR>-Element (Horizontal Rule) übermittelt.

Wo also läßt sich XML einordnen? Irgendwo in der Mitte. Sie wissen, daß XML eindeutig sein und keine Optionen beinhalten soll. Die Regeln, die XML für die Unterscheidung zwischen Markup und Inhalt verwendet, sind sehr einfach:

1. Der Anfang eines Markups wird durch das Kleiner-Zeichen (<) oder das Ampersand-Zeichen (&) gekennzeichnet.
2. Drei weitere Zeichen werden als Markup-Zeichen behandelt: das Größer-Zeichen (>), der Apostroph bzw. das einfache Anführungszeichen (') und das (doppelte) Anführungszeichen (»).
3. Wenn Sie eines dieser Sonderzeichen als normales Zeichen verwenden wollen, müssen Sie ein *Escape* dafür angeben, indem Sie die dafür vorgesehenen allgemeinen Entities verwenden. Sie finden die Ersatzzeichen in Tabelle 2.1.

Ein *Escape* für ein Zeichen bedeutet, daß dieses vor einem nachfolgend angewandten Software-Paket oder der Verarbeitung verborgen wird. Häufig wird dieser Begriff in der Computerwelt verwendet, um in Programmiersprachen ein Präfix für bestimmte Zeichen einzufügen, das verhindert, daß

sie als Sonderzeichen interpretiert werden. Ursprünglich wurde der ESC-Zeichenstring (Escape) den Befehlen für den Drucker vorangestellt, um bestimmte Dinge zu steuern, wie etwa die Schrift- oder Seitengröße, und die Befehlsstrings von druckbaren Zeichen zu unterscheiden.

4. Alles, was kein Markup ist, ist Inhalt (Zeichendaten).

Zeichen	**Ersatz**
&	`&`
'	`&apos;`
<	`>`
>	`<`
"	`&quot;`

Die Interpretationsregeln erinnern irgendwie mehr an HTML als an SGML, aber anders als bei HTML werden die Markup-Zeichen nicht als normale Zeichen behandelt, wenn sie nicht sinnvoll als Markup interpretiert werden können. Werden diese Regeln mit den anderen Regeln zur Struktur von XML-Dokumenten kombiniert, die Sie später noch kennenlernen, sind die Ergebnisse immer vorhersehbar – auch wenn sie nicht immer ganz so sind, wie Sie sich das vorgestellt haben.

Welche Art von Markup verwenden Sie in XML? Die Antwort ist einfach. XML ist *erweiterbar*, und es ist auch eine generische Markup-Sprache, sie kann also inhaltsorientiert, darstellungsorientiert oder beides sein. Es gibt jedoch einen wichtigen Unterschied dahingehend, wie XML und HTML die physische Darstellung realisieren: Das endgültige Erscheinungsbild des XML-Codes wird nicht durch den Web-Browser bestimmt, sondern durch den zugeordneten Stil.

Wie SGML (und anders als HTML) ist XML keine Markup-Sprache, sondern eine Sprache für die Definition von Markup-Sprachen. Sie können also jede XML-Markup-Sprache nach Bedarf erweitern, statt immer dieselben festste

henden Elemente verwenden zu müssen, wie es in HTML der Fall ist. XML ist wirklich *erweiterbar*: Es hat keinen vordefinierten Elementsatz, und Sie können die Elemente nach dem Bedarf der jeweiligen Anwendung benennen und verwenden.

Ein Beispiel für ein XML-Dokument

Listing 2.6 zeigt den XML-Code für eine Web-Homepage. Es handelt sich dabei um ein sehr einfaches Beispiel, aber es enthält alle wichtigen Dinge, die Sie in fast allen XML-Dokumenten finden.

Wie bei allen anderen Listings in diesem Buch werden die Zeilen der einfacheren Orientierung halber numeriert. Die Zahlen sind nicht Bestandteil des XML-Codes, und Sie sollten sie weglassen, wenn Sie Ihren Code selbst eingeben.

Listing 2.6: XML-Code für eine einfache WWW-Homepage

```
1:  <?xml version="1.0"?>
2:   <home.page>
3:     <head>
4:       <title>
5:         My Home Page
6:       </title>
7:       <banner source="topbanner.gif"/>
8:     </head>
9:     <body>
10:       <main.title>
11:         Welcome to My Home Page
12:      </main.title>
13:      <rule/>
14:      <text>
15:        <para>
16:          Sorry, this home page is still
17:          under construction. Please come
18:          back soon!
19:        </para>
20:      </text>
21:    </body>
22:    <footer source="foot.gif"/>
23:  </home.page>
```

Abbildung 2.2 zeigt die Ausgabe des Codes aus Listing 2.6 im Internet Explorer 5.

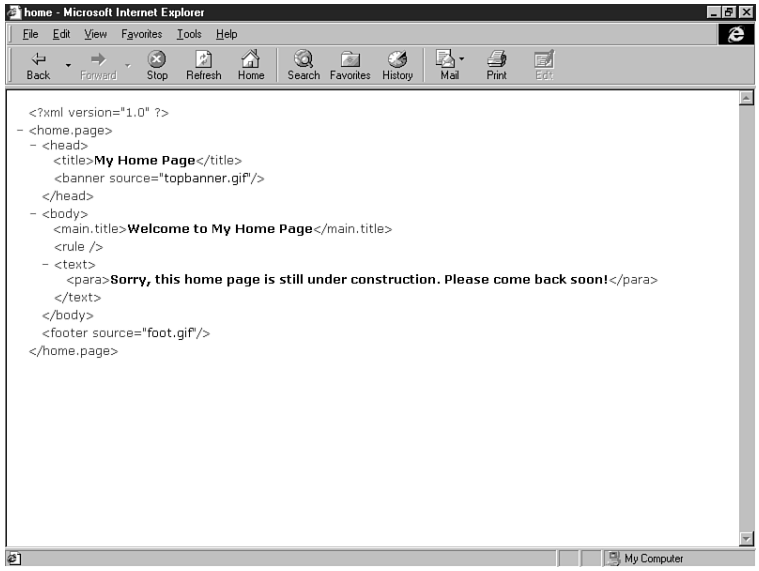


Abbildung 2.2: Der XML-Code für eine einfache Homepage (im Internet Explorer 5).

Die aktuellen Versionen der führenden Web-Browser können den in Listing 2.6 gezeigten XML-Code nicht anzeigen, ohne ihn völlig zu verändern. Die Anzeige in Abbildung 2.2 ist alles andere als optimal, weil IE5 entweder ein XSL- oder ein CSS-Stylesheet benötigt, um XML irgend etwas anderes als das hier gezeigte Standard-Rendering mitzugeben. Eine Erklärung dazu finden Sie in Kapitel 12.

Die XML-Deklaration (1. Zeile)

Die XML-Deklaration gibt an, daß das, was ihr folgt, XML-Code ist. Sie legt fest, welcher Version des XML-Standards der Code entspricht und ob das Dokument als Standalone-Dokument behandelt werden kann (kann es) oder ob man eine DTD braucht, um den Inhalt sinnvoll darstellen zu können.

Die XML-Deklaration ist letztlich eine *Verarbeitungsanweisung* (festgelegt durch das ? am Anfang und am Ende), aber hier ist es ausreichend, sie als Standarddeklaration zu behandeln.

Die XML-Deklaration ist nicht streng verpflichtend (die Tatsache, daß es sich bei dem Dokument um XML-Code handelt, kann auch durch den Web-Server so angekündigt werden, wie es für HTML-Dokumente häufig der Fall ist). Sie sollten sich jedoch angewöhnen, eine solche Deklaration bereitzustellen, weil Ihr Code dadurch besser portierbar wird.

Das Wurzelelement (Zeilen 2 bis 23)

Jedes XML-Dokument darf nur ein Wurzelelement haben, und alle anderen Elemente müssen vollständig von diesem Element eingeschlossen werden. Zeile 2 kennzeichnet den Anfang des <home.page>-Elements (das Start-Tag), Zeile 23 kennzeichnet das Ende des Elements (das Ende-Tag).

Beachten Sie, daß anders als in HTML, wo häufig ein <P>-Tag als Art Formatanweisung verwendet wird, um eine Leerzeile zwischen Textabschnitten einzufügen, in XML ein Element normalerweise aus drei Dingen besteht: einem Start-Tag, dem Inhalt (Text oder anderen Elementen) und einem Ende-Tag.

Ein XML-Element hat nicht immer einen Inhalt. Leere Elemente, wie beispielsweise das IMG-Element in HTML, die einfach durch ihr SRC-Attribut auf eine externe Grafikdatei verweisen, haben offensichtlich keinen Inhalt. Ein leeres Element könnte ein Ende-Tag haben, aber es kann auch eine spezielle Art Start-Tag haben, das erlaubt, daß ein explizites Ende-Tag weggelassen wird.

Beachten Sie außerdem, daß der Name, den Sie im Element-Start-Tag verwenden, genau mit dem im Ende-Tag verwendeten Namen übereinstimmen muß. Falls Sie eine lange Zeichenfolge verwenden wollen, um sprechende Namen darzustellen (beispielsweise DiesIstEinSprechenderName), müssen Sie sehr vorsichtig sein, daß die Groß-/Kleinschreibung in Start- und Ende-Tag exakt beibehalten wird.

XML berücksichtigt die Groß-/Kleinschreibung und erkennt den Unterschied zwischen Großbuchstaben (A-Z) und Kleinbuchstaben (a-z). In Anwendungen, wo die Groß-/Kleinschreibung nicht berücksichtigt wird, werden Zeichen, die gemischt geschrieben wurden, normalerweise konvertiert – also entweder in Großbuchstaben oder in Kleinbuchstaben. Der ASCII-Zeichensatz wandelt in der Regel in Großbuchstaben um. Unicode dagegen nimmt die Umwandlung in Kleinbuchstaben vor. XML muß dies berücksichtigen, auch, weil es möglicherweise in Sprachen eingesetzt wird, wo diese Konvertierung nicht sichergestellt ist. XML verwendet deshalb standardmäßig Kleinbuchstaben (und die XML-Deklaration muß in Kleinbuchstaben dargestellt werden). Es ist sinnvoll, das gesamte Markup in Kleinbuchstaben zu schreiben, auch wenn dadurch Ihr Code weniger gut lesbar wird.

Ein leeres Element (Zeile 13)

Leere Elemente sind ein Sonderfall in XML. In SGML und HTML ist es aus der DTD-Definition eines leeren Elements offensichtlich, daß es leer ist und keinen Kommentar enthält. XML fordert in Übereinstimmung mit den Entwurfszielen seiner Entwickler,

daß Sie deutlicher werden. Sie verwenden möglicherweise überhaupt keine DTD; deshalb könnte es schwierig zu entscheiden sein, ob ein Element leer ist oder leer sein sollte. Leere Elemente müssen als solche sehr deutlich gekennzeichnet werden. Dazu braucht man ein spezielles Endebegrenzungszeichen, />, wie im folgenden gezeigt:

```
<leeres_element/>
```

Um eine gewisse Abwärtskompatibilität zu SGML beizubehalten (bis der SGML-Standard dahingehend aktualisiert wurde, daß die Verwendung von Endekennzeichnern für leere Tags erlaubt ist) und die Umwandlung von existierendem SGML- und HTML-Code in XML ein bißchen einfacher zu machen (ein Prozeß, der als *Normalisierung* bezeichnet wird, wobei allen Elementen Ende-Tags hinzugefügt werden, und der von zahlreichen SGML-Werkzeugen unterstützt wird), können Sie ein Ende-Tag statt des speziellen Endekennzeichners für leere Elemente verwenden. Die Elementdeklaration

```
<graphic source="file.fig"/>
```

kann also ausgetauscht werden durch

```
<graphic source="file.gif"></graphic>
```

Attribute (Zeilen 7 und 22)

Element-Tags können ein oder mehr optionale oder zwingend erforderliche Attribute beinhalten, die weitere Informationen über die Elemente und ihre Begrenzung bereitstellen.

Attribute können nur im Start-Tag eines Elements angegeben werden. Die Syntax für die Angabe eines Attributs lautet:

```
<element.typ.name attribut.name="attribut.wert">
```

Betrachtet man Elemente als Substantive, sind die Attribute die Adjektive. Wir könnten also sagen:

```
<gemüse geschmack="scharf">
```

oder auch

```
<problem größe="riesig" ursache="unbekannt" lösung="davon.laufen"
```

Ein Attribut kann nur im Start-Tag eines Elements angegeben werden.

Im direkten Gegensatz zu SGML und HTML, wo Mehrfachdeklarationen zu fatalen Fehlern führen, behandelt XML Mehrfachdeklarationen von Attributen auf ähnliche Weise. Erscheint ein Element einmal mit einem bestimmten Attributsatz und dann noch einmal mit einem anderen Attributsatz, werden die beiden Attributsätze kombiniert. Die erste Deklaration eines bestimmten Attributs ist aber die einzig gültige, und

alle anderen Deklarationen werden ignoriert. Der XML-Prozessor warnt Sie möglicherweise, daß es mehrere Deklarationen gibt, aber das ist nicht zwingend erforderlich, und die Verarbeitung kann ganz normal weitergeführt werden.

Ein XML-Prozessor ist ein Softwarepaket, eine Bibliothek oder ein Modul, mit denen XML-Dokumente gelesen werden. Der XML-Prozessor macht es einer XML-Anwendung möglich, beispielsweise mit einer Formatier-Engine oder einem Viewer, auf die Struktur und den Inhalt eines XML-Dokuments zuzugreifen.

Logische Struktur

In der vorherigen Beschreibung des HTML-Markups haben Sie die konzeptuellen Unterschiede zwischen den Markups bei XML (und SGML) und HTML kennengelernt. HTML verwendet seine Tags wie Stilschalter. Das Start-Tag schaltet eine Funktion an, beispielsweise das Unterstreichen, und ein Ende-Tag schaltet sie wieder aus. XML verwendet seine Start- und Ende-Tags als Container. Zusammen bilden das Start-Tag, der Inhalt und das Ende-Tag ein einzelnes Element. Elemente sind die Bausteine, aus denen sich ein XML-Dokument zusammensetzt. Jedes XML-Dokument darf nur ein Wurzelelement haben, und alle anderen Elemente müssen vollständig in dieses Element verschachtelt sein. Das bedeutet, wenn ein Element andere Elemente enthält, müssen diese anderen Elemente vollständig von diesem Element eingeschlossen sein.

Betrachten wir, was das für das einfache Beispiel aus Listing 2.6 bedeutet. Wenn Sie die Struktur der Elemente in diesem XML-Dokument skizzieren, erhalten Sie die in Abbildung 2.3 gezeigte Baumstruktur.

Wie Sie in Abbildung 2.3 sehen, hat das Dokument eine baumähnliche Struktur, wobei sich das Wurzelelement (<home.page>) ganz oben im Baum befindet (oder ganz unten, abhängig von Ihrer Perspektive). Alle Elemente innerhalb dieses Elements sind sauber ineinander enthalten. Ein XML-Dokument darf nur ein einziges Root-Element haben, und es darf keine Elemente geben, die teilweise oder ganz außerhalb, vor oder hinter diesem Element liegen.

Um das Ganze noch einfacher zu machen, betrachten wir die Beziehungen zwischen Elementen und zu Elementen in Hinblick auf andere Elemente. Man könnte sagen, ein Element ist das *Eltern*-Element der darin enthaltenen Elemente, ihnen also übergeordnet. Die Elemente innerhalb eines Elements werden als *Kind*-Elemente bezeichnet; sie sind untergeordnet. Elemente, die dasselbe Eltern-Element haben, werden auch als *Geschwister* bezeichnet – als gleichrangig.

In dem einfachen Beispiel in Abbildung 2.3 ist <home.page> das Eltern-Element aller anderen Elemente, <text> ist das Eltern-Element von <para>, <title> ist ein Kind von <head>, und <title> und <banner> sind Geschwister. Wenn man den Elementbaum durchläuft, muß jedes Kind-Element vollständig in seinem Eltern-Element enthalten sein. Geschwister-Elemente dürfen sich nicht überlappen.

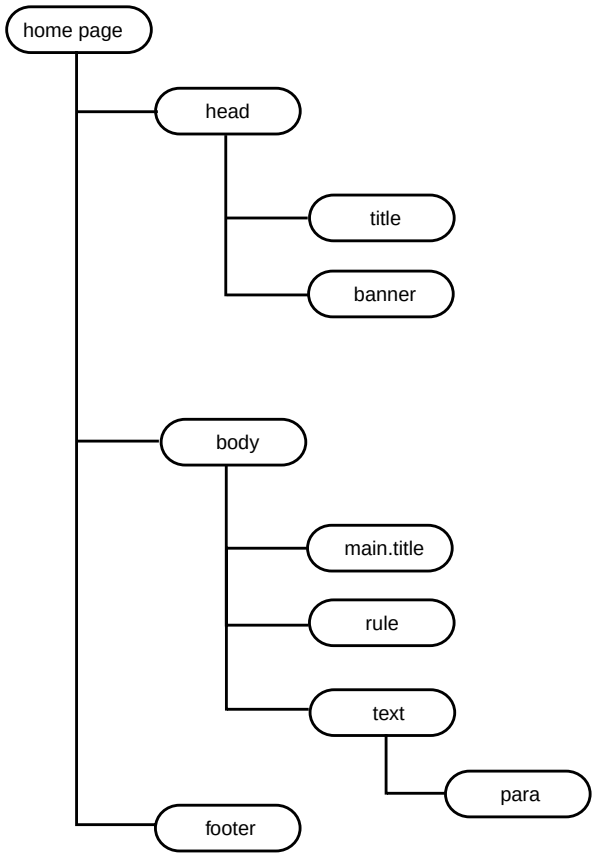


Abbildung 2.3:
Die logische Struktur der Elemente.

Die Anordnung der Elemente in einem XML-Dokument wird auch als logische Struktur bezeichnet. Wie Sie als nächstes sehen werden, hat ein XML-Dokument auch eine physische Struktur. Um well-formed (wohlgeformt) zu sein, müssen die logische und die physische Struktur eines XML-Dokuments synchron sein; sie müssen vollständig und korrekt ineinander verschachtelt sein.

Physische Struktur

Eines der Schlüsselkonzepte von XML ist das Entity. Um XML wirklich zu verstehen, müssen Sie verstehen, was Entities sind. Es gibt unterschiedliche Arten von Entities, und die Entities sind viel wichtiger als die Elemente, wenn es darum geht, festzulegen, wie der XML-Prozessor den XML-Code verarbeiten soll. Sie erfahren später mehr über die Entities; hier ist es ausreichend, sich ein Entity als physische Speichereinheit vorzustellen. Es handelt sich um ein Objekt, aber man kann sich die meisten Entities in der Regel auch als einzelne Computerdateien vorstellen.

Ein Entity ist im wesentlichen eine Informationseinheit, aber der offizielle Begriff in der XML-Sprachdefinition sagt, daß ein Entity ein Speicherobjekt ist. Dieses Objekt könnte ein Element sein, aber auch ein XML-ENTITY-Objekt (normalerweise eine nicht geparste, externe Datei).

Das wichtigste Entity, mit dem Sie die ganze Zeit arbeiten, obwohl Sie es vermutlich nie bemerken, ist das Dokument-Entity. Wie Sie gesehen haben, ist dieses Dokument (oder Wurzel-)Entity logisch in Elemente untergliedert. (Es gibt noch weitere logische Komponenten, die später beschrieben werden; hier wollen wir uns auf die Elemente konzentrieren.)

Entities können auf andere Entities verweisen und sie veranlassen, daß sie in das XML-Dokument aufgenommen werden. Und Sie kennen sogar bereits einige Entities. Die in Tabelle 2.1 aufgelisteten Entities, die Sie verwenden, um in normalem Text ein Escape für Markup-Zeichen einzufügen, sind letztlich interne Entities; mehr darüber später. Hier wollen wir den grundlegenden Verweis auf eine Grafikdatei betrachten, wie er in HTML-Webseiten völlig üblich ist. In Listing 2.6 (Zeile 7) haben wir folgendes gesehen:

```
<banner source="topbanner.gif"/>
```

Das `source`-Attribut des `banner`-Elements verweist auf ein externes Entity (das nicht im aktuellen Dokument enthalten ist), eine externe Grafikdatei. Wäre dies HTML-Code, würde die Grafik an dieser Stelle im Dokument in Ihrem Web-Browser erscheinen. In XML-Terminologie wird diese Grafikdatei als *nichtanalysiertes Entity* bezeichnet; der XML-Prozessor ignoriert den Inhalt dieses Entity und übergibt es der Anwendung.

XML ist ein wenig strenger als HTML, was das Einbinden externer Grafikdateien betrifft. Wie Sie später noch lernen werden, erlaubt Ihnen XML, die *Notation* oder das Format anzugeben, in der die Grafik vorliegt, aber es kann auch mehr als nur eine einfache Grafik einzubinden. Und hier kommen die Probleme!

XML kann Entities einbinden, die XML-Code, Text oder HTML-Code – fast alles – enthalten, und zwar abhängig davon, wie das betreffende Entity gekennzeichnet wird

– wäre es auch XML-Code, könnte es vom XML-Prozessor verarbeitet (geparst) werden, als wäre dieser XML-Code im ursprünglichen Dokument (Wurzel-Entity) und nicht in einer externen Datei enthalten. Um das Ganze noch komplizierter zu machen, könnte ein XML-Entity auch auf ein weiteres Entity verweisen usw. Neben den praktischen Problemen, die dadurch entstehen könnten (stellen Sie sich nur vor, Sie wollen ein kleines Dokument öffnen und erhalten mehrere tausend verknüpfte Seiten!), tauchen hier sehr spezielle Probleme auf, wenn die eingebundenen Entities ebenfalls Markup enthalten.

So wie die Anordnung der Elemente zu einer logischen Struktur führt, ergibt die Anordnung der Entities eine physische Struktur. Angenommen, ein eingebundenes Entity enthält auch Elemente. Das scheint auf den ersten Blick kein Problem zu sein, aber es wird zu einem Problem, wenn das eingebundene Entity Elemente mit denselben Elementtypnamen enthält wie die Elemente in den Wurzel- (oder anderen) Entities und wenn es für die Anwendung wichtig ist, sie zu unterscheiden. Dieses Problem wird durch einen Mechanismus gelöst, der als *Namensraum* bezeichnet und in Kapitel 7 genauer beschrieben wird.

Neben den Konflikten zwischen den Elementen können Sie sich vorstellen, Sie haben ein Element in dem Wurzel-Entity (Ihrem XML-Dokument) geöffnet und auf ein externes Entity verwiesen. Auch kein Problem; das ist eine übliche Vorgehensweise. Aber angenommen, dieses externe Entity enthält ein Ende-Tag für ein Element, das Sie gerade geöffnet haben. Plötzlich wäre Ihre gesamte logische Struktur zerstört.

Um das Auftreten dieser Probleme zu begrenzen, müssen die logische und die physische Struktur von XML-Entities synchron sein; logische Entities dürfen nicht über die Grenzen des physischen Entities hinausgehen, und die physischen Entities müssen vollständig in die logischen Entities eingeschlossen (verschachtelt) sein. Manchmal ist es kompliziert, das zu überprüfen, aber wenn es *nicht* der Fall ist, können zahlreiche Probleme daraus entstehen. Diese Anforderung ist einzigartig für XML, und für Autoren, die von SGML oder HTML kommen, ist es vermutlich das, was am schwierigsten zu verstehen ist und wo die meisten Fehler begründet liegen.

Zusammenfassung

Dieses Kapitel hat ein XML-Dokument als ein aus verschiedenen Komponenten zusammengesetztes Objekt betrachtet. Sie haben erfahren, was Markup ist und wie XML zwischen Markup und Zeichendaten unterscheidet. Außerdem haben Sie erfahren, daß XML-Dokumente eine logische und eine physische Struktur haben, und Sie haben die Verbindung zwischen diesen beiden und den Elementen und den Entities von XML gesehen.

F&A

F Warum berücksichtigt XML die Groß-/Kleinschreibung, während das in SGML und HTML nicht der Fall ist?

A *XML soll für Anwendungen eingesetzt werden, die möglicherweise die Groß-/Kleinschreibung nicht berücksichtigen und wo die Vorgehensweise für die Zeichenumwandlung (in Groß- oder in Kleinbuchstaben) nicht vorhergesagt werden kann. Statt gefährliche Annahmen zu treffen, schlägt XML den sichersten Weg ein und berücksichtigt die Groß-/Kleinschreibung.*

F Warum sind die Web-Browser bei der Interpretation von Markup so locker?

A *Das war eine bewußte Entscheidung, die eine möglichst einfache Handhabung des Web garantieren. Statt zahlreiche Regeln zum HTML-Code zu erzwingen, traf man die Entscheidung, alles zu akzeptieren und alles, was nicht interpretierbar ist, zu ignorieren.*

F Warum erlaubt XML keine Tag-Minimierung?

A *XML-Prozessoren sollen einfach zu erzeugen (und damit billig und schnell) sein. Das bedeutet, man soll nicht von ihnen erwarten, eine komplexe Verarbeitung vorzunehmen, wie etwa das Speichern des aktuellen Kontexts und dann eine Weitersuche, wo das aktuelle Elemente endet. Das Tagging muß deshalb zu 100 Prozent explizit sein, so daß der durch die Tag-Minimierung entstehende Verarbeitungsmehraufwand vermieden wird.*

F Welche Aufgabe haben die vordefinierten Entities?

A *Offiziell bieten sie Ihnen eine einfache Methode, ein Escape für die XML-Begrenzer vorzunehmen. Inoffiziell bedeuten sie, daß es keine Entschuldigung dafür gibt, zu versuchen, die Begrenzungszeichen als normale Zeichen zu verwenden. Im allgemeinen machen Sie es einem XML-Prozessor einfacher, weil die Begrenzungszeichen immer als Begrenzungszeichen und nie als normale Zeichen behandelt werden (obwohl es einige Ausnahmen gibt).*

F Warum müssen Elemente korrekt verschachtelt werden?

A *Auch dadurch wird versucht, Verarbeitungsmehraufwand zu vermeiden. Wenn sich der XML-Prozessor merken muß, welche Elemente geöffnet und noch nicht geschlossen wurden, muß er diese Information irgendwo ablegen. Das bedeutet, es muß Hauptspeicher (oder Plattenspeicher) verwendet werden, was wiederum mehr Verarbeitung, mehr Komplexität und weniger Geschwindigkeit bedeutet. Das ist eines von vielen Beispielen in XML, wo ein bißchen (erzwungene) Disziplin bei der Codierung sehr viel Programmieraufwand ersparen kann.*

Übungen

1. Tragen Sie das Markup für die folgende E-Mail-Nachricht ein, um ihren Informationsinhalt zu kennzeichnen:

```
From: Simon North <north@synopsys.com>
            To: Nick <sintac@xs4all.nl>
            Subject: Hi
            Hi Nick, this is just a quick message
            to say I got the material. Thanks.
```

2. Tragen Sie jetzt das Markup für dieselbe Nachricht ein, um ihr Erscheinungsbild festzulegen:

```
From: Simon North <north@synopsys.com>
            To: Nick <sintac@xs4all.nl>
            Subject: Hi
            Hi Nick, this is just a quick message
            to say I got the material. Thanks.
```

3. Vergleichen Sie die beiden bearbeiteten Nachrichten. Nennen Sie zwei Gründe, warum die eine Art Markup sinnvoller sein könnte als die andere.

XML-Markup

Woche 1

In Kapitel 2 haben Sie die wichtigsten Funktionen des XML-Markups für Elemente und Entities kennengelernt. Dieses Kapitel baut darauf auf, und Sie lernen die folgenden Dinge kennen:

- Attribute
- Entity-Verweise und ihre Verwendung als Shortcuts
- Kommentare im Code
- CDATA-Abschnitte und ihre Verwendung
- Verarbeitungsanweisungen (Processing Instructions)

Markup-Begrenzungszeichen

Gestern haben Sie die Markup-Zeichen von XML ganz allgemein kennengelernt. Jetzt wollen wir ein bißchen technischer werden und die genauen Details der Markup-Deklarationen von XML betrachten.

Tabelle 3.1 zeigt die Bestandteile der Element-Tags von XML. Wenn die Details noch technischer werden, ist es wichtig, daß Sie mit diesen Dingen vertraut sind. (Sie müssen sie aber nicht unbedingt im Gedächtnis festschreiben!)

Symbol	Beschreibung
<	Öffnendes Begrenzungszeichen für das Start-Tag
</	Öffnendes Begrenzungszeichen für das Ende-Tag
foo	Elementname
>	Schließendes Begrenzungszeichen für ein Tag
/>	Schließendes Begrenzungszeichen für ein leeres Tag

Tabelle 3.1: Die Teile eines Element-Tags

Sie sollten sich merken, daß HTML einfach auf dem Erkennen vorprogrammierter Tags basiert, während bei XML durch diese Komponenten eine Verarbeitung der Element-Tags ausgelöst wird. Das Verhalten des XML-Prozessors und was er als nächstes erwartet, wird direkt durch die hier gezeigten Symbole gesteuert.

Element-Markup

XML hat mit Element-Markup zu tun. Das scheint eine offensichtliche Feststellung zu sein, aber sie soll hier wiederholt werden, weil sie auf den großen Unterschied zwischen XML als Markup-Sprache und einer beliebigen Tag-Sprache hinweist. Wie Sie bereits gesehen haben, tendiert HTML oft dazu, eine Tag-Sprache statt eine Markup-Sprache zu sein. Das ist eine direkte Konsequenz der Tatsache, daß die Web-Browser so bewußt locker mit der Akzeptanz fehlerhaften Markups umgehen.

Statt die XML-Tags als Kennzeichner zu betrachten, die anzeigen, wann ein Stil geändert werden soll oder eine neue Zeile beginnt, setzt sich das Element-Markup von XML aus drei Teilen zusammen: einem Start-Tag, dem Inhalt und dem Ende-Tag. Das sehen Sie in Tabelle 3.2. Das Start-Tag und das Ende-Tag können als eine Hülle verstanden werden, und wenn Sie sich ein Element vorstellen, dann sollten Sie ein mentales Bild eines Textabschnitts mit beiden Tags vor Augen haben.

Symbol	Name	Beschreibung
`<foo>`	Start-Tag	Am Anfang eines Elements, das öffnende Tag
`text`	Inhalt	In der Mitte eines Elements, sein Inhalt
`</foo>`	Ende-Tag	Am Ende eines Elements, das schließende Tag

Tabelle 3.2: Bestandteile eines Elements

Beachten Sie, daß der Elementname im Start-Tag genau dem Namen im Ende-Tag entsprechen muß. Beispielsweise wäre folgendes falsch:

```
<simple.element>Dieses Element wird nicht mehr geschlossen!</simple.Element>
```

In den ersten XML-Versionen vor der vollständigen Spezifikation wurde die Groß-/Kleinschreibung nicht berücksichtigt. Es gibt immer noch XML-Softwarepakete, die die Groß-/Kleinschreibung nicht beachten und die keinen Fehler monieren, wenn Sie Groß- und Kleinbuchstaben kombinieren. Der Konformität mit den XML-Anforderungen halber sollten Sie sorgfältig darauf achten, daß Sie Groß- und Kleinbuchstaben konsistent verwenden.

Attribut-Markup

Wie Sie gestern erfahren haben, werden Attribute genutzt, um Informationen an die in einem Element enthaltenen Informationen anzufügen. Die allgemeine Form der Verwendung eines Attributs sieht wie folgt aus:

```
<element.name eigenschaft="wert">
```

oder

```
<element.name eigenschaft='wert'>
```

Die technische Beschreibung des Markups in dieser Attributspezifikation finden Sie in Tabelle 3.3.

Symbol	Beschreibung
<	Öffnendes Begrenzungszeichen für das Start-Tag
element.name	Elementname
eigenschaft	Attributname
=	Wertindikator
"	Begrenzungszeichen für die Zeichenkette
'	Alternatives Begrenzungszeichen für die Zeichenkette
wert	Attributwert
<	Schließendes Begrenzungszeichen für das Start-Tag

Tabelle 3.3: Bereitstellung eines Attributs

Beachten Sie, daß ein Attributwert in Anführungszeichen eingeschlossen werden muß. Sie verwenden dazu entweder einfache Anführungszeichen (`<lie size='big'>`) oder doppelte Anführungszeichen (`<lie size ="massive">`), aber Sie können die beiden nicht innerhalb einer einzigen Spezifikation kombinieren.

Falls Sie ohne eine DTD arbeiten (keiner der in diesem Kapitel gezeigten XML-Codes bedingt, daß Sie dem XML-Dokument eine DTD zuordnen), können Sie das Attribut und seinen Wert einfach dann angeben, wenn Sie das Element zum ersten Mal verwenden, wie in Listing 3.1 gezeigt. Wenn Sie mehrfach Attribute für dasselbe Element angeben (wie in den Zeilen 3 und 4 von Listing 3.1), werden diese Spezifikationen einfach.

Listing 3.1: Attributspezifikationen

```
1: <?xml version="1.0"?>
2: <home.page>
3:  <para number="first">Dies ist der erste Absatz.</para>
4:  <para number='second' color="red">Dies ist
5:   der zweite Absatz.</para>
6:  </home.page>
```

Wenn der XML-Prozessor auf die Zeile 3 trifft, erkennt er, daß ein `para`-Element ein Zahlenattribut hat. (Sie wissen, daß dies alles in Abwesenheit einer DTD passiert, die explizit deklarieren würde, welche Attribute ein `para`-Element hat.) Wenn er dann auf Zeile 4 trifft, erkennt er, daß ein `para`-Element auch ein `color`-Attribut hat.

Es gibt ein Attribut, das für XML selbst reserviert ist: `xml:lang`. Dieses Attribut gibt an, in welcher natürlichen Sprache das Element geschrieben wurde. Der Wert des Attributs ist einer der Sprachencodes, die in ISO 639 festgelegt sind; einige der gebräuchlichsten sind in Tabelle 3.4 aufgelistet.

Code	**Sprache**
`ar`	Arabisch
`ch`	Chinesisch
`de`	Deutsch
`en`	Englisch
`es`	Spanisch
`fr`	Französisch
`gr`	Griechisch
`it`	Italienisch
`ja`	Japanisch
`nl`	Niederländisch
`pt`	Portugiesisch
`ru`	Russisch

Tabelle 3.4: Gebräuchliche ISO-639-Sprachencodes

Wenn es mehrere Versionen einer Sprache gibt, wie beispielsweise britisches und amerikanisches Englisch, kann dem Sprachencode ein Trennstrich (-) und einer der in ISO 3166 festgelegten Ländercodes folgen. Einige der gebräuchlichsten Ländercodes sind in Tabelle 3.5 aufgeführt. Wenn Sie bereits Erfahrung mit dem Internet haben, werden Sie einige dieser Codes aus E-Mail-Adressen und URLs bereits kennen. Ein Element, das in amerikanischem Englisch geschrieben wurde, könnte wie folgt gekennzeichnet werden (beachten Sie die Groß-/Kleinschreibung; der Sprachcode wird in Kleinbuchstaben, der Ländercode in Großbuchstaben angegeben):

```
<para xml:lang="en-US">Mein Land sieht so aus.</para>
```

Code	Land
AT	Österreich
BE	Belgien
CA	Kanada
CN	China
DE	Deutschland
DK	Dänemark
EN	England
ES	Spanien
FR	Frankreich
GR	Griechenland
IT	Italien
JA	Japan
NL	Niederlande
PT	Portugal
RU	Rußland
US	Vereinigte Staaten von Amerika

Tabelle 3.5: Gebräuchliche ISO 3166-Ländercodes

Die in den Tabellen 3.4 und 3.5 gezeigten Codes sind keineswegs vollständig. Es gibt noch ein weiteres Codierungsschema von der IANA (*Internet Assigned Numbers Authority*), das in RFC 1766 definiert ist. Und falls nötig, können Sie auch Ihren eigenen Sprachcode definieren. Benutzerdefinierten Codes muß der String `x-` vorausgehen, so daß Sie ein Element auch als »Computer-Chinesisch« ankündigen können, etwa wie folgt:

```
<para xml:lang="x-cc">Können Sie das entziffern?</para>
```

Namensregeln

Bisher haben Sie das Markup für Elemente und Attribute kennengelernt, und in allen Beschreibungen wird gesagt, daß diese Markup-Objekte Namen tragen. XML hat bestimmte Namensregeln für seine Markup-Objekte.

Die Namensregeln von XML sehen wie folgt aus:

- Ein Name besteht aus mindestens einem Zeichen: a bis z oder A bis Z.
- Falls der Name aus mehr als einem Zeichen besteht, kann er mit einem Unterstrich (_) oder einem Doppelpunkt (:) beginnen.

- Dem ersten Buchstaben (oder Unterstrich) können ein oder mehr Buchstaben, Ziffern, Trennstriche, Unterstriche, Punkte, kombinierende Zeichen, Erweiterungszeichen oder ignorierbare Zeichen folgen. (Diese letzten drei Zeichenklassen wurden aus dem Unicode-Zeichensatz übernommen und beinhalten einige spezielle Unicode-Zeichensymbole und Akzente. Eine vollständige Liste finden Sie online in der XML-Empfehlung unter `http://www.w3.org/XML/REC-xml`.)

Das W3C (*World Wide Web Consortium*) aktualisiert seine Web-Site in regelmäßigen Abständen, und die URLs für Empfehlungen, Hinweise und Arbeitsskizzen ändern sich relativ häufig. Wenn Sie ihre Web-Site besuchen, finden Sie einen Zeiger auf den URL-Minder-Service. Dieser kostenlose Dienst ist eines der vielen Wunder im Web. Durch Registrierung einer Webseite – einer beliebigen Webseite – erhalten Sie automatisch eine E-Mail, wenn sich irgend etwas auf dieser Seite ändert. Das ist eine ausgezeichnete Möglichkeit, sich über Weiterentwicklungen zu informieren.

Beachten Sie, daß in Elementnamen keine Leerzeichen und Tabulatoren erlaubt sind (`<eins zwei>` würde als zwei separate Namen interpretiert), und die einzigen erlaubten Interpunktionszeichen sind der Gedankenstrich (-) und der Punkt (.).

Es gibt keine Regel, die besagt, Ihre Namensgebung müsse sinnvoll sein. So lange Sie die Namensregeln nicht verletzen, können Sie den XML-Objekten beliebige Namen zuweisen, die so lang und so sinnlos sein dürfen, wie Sie das für richtig halten. Einer der wichtigsten Vorteile bei der Verwendung von XML ist, daß der Code selbstbeschreibend ist. Wenn Sie Elemente wie `<dingsda>`, `<irgendwas>` oder `<huh>` verwenden, dann machen Sie den ganzen Effekt zunichte. Versuchen Sie Namen zu verwenden, die wenigstens ansatzweise die Art oder die Aufgabe des Objekts treffen. Vergessen Sie nicht, daß eines der Ziele von XML die Lesbarkeit durch den Benutzer ist. Lesbar sein ist das eine, aber es ist durchaus sinnvoll, wenn die Namen auch eine Bedeutung kenntlich machen.

Kommentare

Keine Sprache, egal ob eine Programmiersprache oder eine Markup-Sprache, könnte überleben, würde sie nicht erlauben, dem Code Kommentare hinzuzufügen. Aus der Wartungsperspektive ist es sehr wichtig, dauerhafte Aufzeichnungen zu besitzen, die sagen, warum bestimmte Dinge auf ganz bestimmte Weise gemacht wurden. Die beste Methode, Ihren Code zu dokumentieren, ist die Aufnahme von Erklärungen in den Code mit Hilfe von Kommentaren.

In Übereinstimmung mit dem Entwurfsziel, XML einfach zu halten, sind auch die Kommentarfunktionen einfach. Kommentare haben die folgende Form:

```
<!-- Dies ist ein Kommentar -->
```

Das Start-Tag (<!--) und das Ende-Tag (-->) müssen genau wie hier gezeigt verwendet werden. Das Einfügen von Leerzeichen oder anderen Zeichen in diese Zeichenketten kann dazu führen, daß Tags oder alles andere innerhalb des Kommentars vom XML-Prozessor versehentlich als Markup interpretiert wird.

Vorausgesetzt, Sie verwenden das Start-Tag und das Ende-Tag für den Kommentar korrekt, wird alles innerhalb des Kommentartexts vom XML-Prozessor vollständig ignoriert. Der folgende Kommentar ist also ganz sicher:

```
<!-- Dies sind die Deklarationen für den <title> und den <body> -->
```

Es gibt nur eine Einschränkung, was Sie in Ihrem Kommentartext schreiben dürfen: Die Zeichenkette -- ist nicht erlaubt. Damit bleibt XML abwärtskompatibel zu SGML. (Die Zeichenkette --> beendet den Kommentar offensichtlich.)

Kommentare können an beliebiger Stelle in einem XML-Dokument angelegt werden – außerhalb anderer Markups. Das folgende ist also erlaubt:

```
<para>Das ist einfach <!-- sagen sie jedenfalls alle --> zu erledigen.</para>
```

während das folgende nicht geht:

```
<para <!-- unverschämte Lüge --> >Das ist ganz einfach.</para>
```

Zeichenverweise

Anders als SGML (und damit auch anders als HTML), das weitgehend auf ASCII basiert, wurde XML von Anfang an darauf hinentwickelt, auch andere Sprachen als Englisch zu unterstützen. XML unterstützt deshalb Zeichen mit Akzenten oder aus Fremdsprachen besser als SGML oder HTML.

In HTML können Sie immer den Code für das gewünschte Fremdsprachenzeichen eingeben (`è` wäre `&egrave;`, `í` wäre `&iacute;`, und `û` wäre `&ucirc;`). Wie Sie später in diesem Kapitel noch sehen werden, sind diese Codes letztlich Verweise auf Entities. Die Abkürzungen `egrave`, `iacute` und `ucirc` stammen aus dem Zeichensatz ISO 8859/1 (das ist der SGML-Zeichensatz), der sich von der Version ISO/IEC 646 des ASCII-Alphabets (den ersten 128 Zeichen) ableitet. ISO 8859/1 ist außerdem die Grundlage für die Schriften in Microsoft Windows.

Mit Hilfe dieser Verweise auf Zeichen-Entities können Sie zwar den meisten europäischen und skandinavischen Sprachen gerecht werden, aber Sie hätten plötzlich ein Problem, wenn Sie asiatische Sprachen oder Sprachen aus Mittelost darstellen wollten, wie etwa Japanisch, Hindi oder Arabisch. XML basiert jedoch auf Unicode und den noch umfangreicheren ISO/IEC 10646-Standards (die sogar die Verwendung chinesischer Zeichen erlauben). Sie brauchen sich jetzt nicht zu detailliert mit diesen Zeichensätzen auseinandersetzen (oder überhaupt nicht, wenn Sie vorhaben, im Web nur in westlichen Sprachen zu veröffentlichen), aber wir werden später noch einmal darauf zurückkommen.

Das Wichtigste, was Sie über diese exotischen Zeichen wissen sollten, ist, daß Sie sie auch eingeben können, wenn sie von Ihrer Tastatur nicht unterstützt werden. Dazu geben Sie einen Zeichenverweis ein.

Ein Zeichenverweis besteht aus der Zeichenkette `&#`, gefolgt von der Nummer des Zeichens im ISO/IEC 10646-Alphabet und abgeschlossen durch ein Semikolon (;). Die Nummer des Zeichens kann dezimal dargestellt werden, dann geben Sie sie einfach unverändert ein, oder hexadezimal, dann schreiben Sie den Buchstaben `x` vor die Zahl, beispielsweise `x12ABC`. Der Zeichenverweis für das Copyright-Symbol (©) beispielsweise – das in HTML als `&copy;` angesprochen wird –, ist `&#169;` (dezimal) oder `&#xA9;` (hexadezimal).

Vordefinierte Entities

Zeichenverweise erlauben Ihnen auch, Zeichen einzugeben, die auf Ihrer Tastatur nicht zur Verfügung stehen. Unter anderem geht es dabei um die *vordefinierten Entities*. Dabei handelt es sich um Zeichen, die Sie normal eingeben, was aber nicht empfehlenswert ist, weil sie sonst möglicherweise als Markup-Zeichen fehlinterpretiert werden. Um Ihr Gedächtnis aufzufrischen, sehen Sie in Tabelle 3.6 die vordefinierten Entities.

Zeichen	**Ersatz**
&	& oder &#38
'	' oder '
>	> oder >
<	< oder <
"	" oder "

Tabelle 3.6: Vordefinierte Entities

Sie können ein benanntes Entity eingeben, um ein Zeichen darzustellen, wie etwa `&apos;`, oder einen Zeichenverweis eintragen, wie etwa `&#39;`. Die Zeichenverweise für das Ampersand- (`&`) und das Kleiner-Zeichen (`<`) sind jedoch Sonderfälle; deshalb muß ein doppeltes Escape dafür erfolgen. Der Grund dafür wird im nächsten Abschnitt erklärt.

Entity-Verweise

Wie Sie aus der gestrigen Beschreibung des Aufbaus eines XML-Dokuments wissen, sind Entities normalerweise externe Objekte, wie etwa Grafikdateien, die in das Dokument eingebunden werden sollen. Um auf diese externen Entities zu verweisen, brauchen Sie eine DTD für Ihr XML-Dokument. Sie werden mehr über diese Entities erfahren, wenn Sie die DTDs kennenlernen, aber es gibt noch einen Entity-Typ, den Sie bereits benutzen, das sogenannte *interne Entity*. Damit können Sie sich eine Menge unnötiger Schreibarbeit ersparen.

Interne Entities sehen ganz ähnlich wie Zeichenverweise aus, mit einem entscheidenden Unterschied – Sie müssen ein internes Entity deklarieren, bevor Sie es nutzen können.

Entity-Deklarationen

Die Deklaration eines internen Entities hat die folgende Form:

```
<!ENTITY name "ersatztext">
```

Immer wenn die Zeichenkette `&name;` dann in Ihrem XML-Code erscheint, ersetzt der XML-Prozessor sie automatisch durch den Ersatztext (der beliebig lang sein darf). Sinnvoll eingesetzt, können Ihnen die Entity-Verweise eine Menge Schreibarbeit ersparen.

Die Vorteile von Entities

Sie können sich einen Entity-Verweis ähnlich einem Makro vorstellen. Aber wie auch immer Sie ihn bezeichnen, er kann Ihnen viel Zeit sparen, wenn es einen Textabschnitt gibt, den Sie mehrere Male brauchen, oder wenn Sie irgendeine Art Schablonentext verwenden wollen.

Betrachten Sie das Beispiel in Listing 3.2. Dort wird ein Entity-Verweis für einen Copyright-Hinweis verwendet.

Listing 3.2: Verwendung eines internen Entity

```
1:  <?xml version="1.0"?>
2:  <home.page>
3:    <head><title>Title Page</title></head>
4:    <body> <h1>The Title Page</h1>
5:      <para>(c) 1998, &rights;</para>
6:     </body>
7:  </home.page>
```

Mit der folgenden Deklaration für das Entity `rights`:

```
<!ENTITY rights "Alle Rechte vorbehalten. Kein Teil dieses Buchs, auch nicht der
Entwurf, das Cover-Design und die Icons, dürfen ohne vorherige Zustimmung des
Verlags in irgendeiner Form über irgendein Medium (elektronisch, durch Kopien,
Aufzeichnungen oder andere Methoden) reproduziert oder weitergegeben werden."
```

Damit ergibt sich durch Ersetzung in Zeile 5 von Listing 3.2 das folgende Ergebnis:


```
<para>(c) 1998, Alle Rechte vorbehalten. Kein Teil dieses Buchs, auch nicht der
Entwurf, das Cover-Design und die Icons, dürfen ohne vorherige Zustimmung des
Verlags in irgendeiner Form über irgendein Medium (elektronisch, durch Kopien,
Aufzeichnungen oder andere Methoden) reproduziert oder weitergegeben werden.</
para>"
```

Bei dieser Verwendung des Entity-Verweises müssen Sie den Text nur ein einziges Mal eingeben, nämlich in der Entity-Deklaration, und brauchen dann bei einer Bearbeitung nicht mehr jedes Vorkommen der Zeichenkette im Text zu suchen und zu ändern. Auf diese Weise vereinfachen die Entity-Verweise die Aufgabe, XML-Dokumente zu erzeugen und zu verwalten. In Kapitel 8 erfahren Sie, wie Sie diese Funktion so erweitern können, daß externe Entities als Ausgangspunkt für Textfunktionen genutzt werden, und wie Sie diese Text-Entities in einem allgemeinen Dokument deklarieren, auf das beliebig viele andere Dokumente Zugriff haben.

Gefahren bei der Verwendung von Entities

Sie haben jetzt gesehen, wie praktisch interne Entity-Verweise als Abkürzung für die Eingabe von Textabschnitten und die Arbeit mit variablem Inhalt sein können. Offensichtlich können Ihnen Entity-Verweise bei der richtigen Handhabung und den entsprechenden Vorbereitungen viel Zeit und Mühe ersparen.

Natürlich stellt man bei so einer praktischen Funktion sofort eine Frage: »Könnte ich sie auch nutzen, um Markups damit einzufügen?« Das ist eine gute und naheliegende Idee. Können Sie Markups innerhalb des Ersatztexts angeben? Ja, das können Sie …, aber es unterliegt einigen Einschränkungen, und Sie müssen sorgfältig darüber nachdenken, um unliebsame Überraschungen zu vermeiden.

Als erstes müssen Sie daran denken, daß XML den Inhalt des Entity-Ersatztexts verarbeitet, wenn es den Entity-Verweis expandiert. Das bedeutet, Sie dürfen im Ersatztext keine Escapes für Markup-Zeichen verwenden; Sie müssen für diese Zeichen ein doppeltes Escape angeben. Betrachten Sie ein einfaches Beispiel:

```
<!ENTITY gefaehrlich "Black &#38; White">
```

Wenn der XML-Prozessor den Entity-Verweis `&gefaehrlich;` im XML-Dokument sieht, expandiert er sofort das vordefinierte Entity, bevor er den Ersatztext einfügt. Dieser XML-Code scheint ganz harmlos zu sein:

```
<text>Dieser Text ist nicht &gefaehrlich;.</text>
```

Aber sehen wir uns schrittweise an, was passiert:

1. Der XML-Prozessor sieht den Verweis `&gefaehrlich;` und sucht nach dem Ersatztext.
2. Er findet `Black &#38; White` und macht daraus `Black & White`.
3. Der XML-Prozessor fügt den Ersatztext ein, und der resultierende XML-Code sieht wie folgt aus:

   ```
   <text>Dieser Text ist nicht Black & White.</text>
   ```

4. Der XML-Prozessor versucht dann, das Ampersand zu verarbeiten, und gibt einen Fehler zurück, weil `&` nicht als Entity deklariert wurde.

Die Fallstricke und wie man sie vermeidet

Sie haben einige der Probleme kennengelernt, die entstehen, wenn der Inhalt von Entity-Verweisen dereferenziert wird. Im schlimmsten Fall können sie Ihren gesamten XML-Code zerstören. Es gibt natürlich eine Möglichkeit, diese Probleme zu vermeiden – durch ein doppeltes Escape für jedes Markup im Ersatztext, beispielsweise wie folgt:

```
<!ENTITY sicher "Harry &#38;#38; Fred &amp;amp; Joe">
```

Wenn der XML-Prozessor den Entity-Verweis `&sicher;` in diesem XML-Dokument sieht:

```
<text>Der Job konnte &sicher; übertragen werden.</text>
```

ergibt die Erweiterung korrekten Code. Betrachten wir, was passiert, wenn der XML-Prozessor den Entity-Verweis dereferenziert:

1. Der XML-Prozessor sieht den Entity-Verweis `&sicher;` und sucht nach dem Ersatztext.
2. Der XML-Prozessor findet »Harry & Fred & Joe« und dereferenziert es zu Harry & Fred & Joe.
3. Der XML-Prozessor fügt den Ersatztext ein, und es entsteht der XML-Code

   ```
   <text>Der Job konnte Harry &amp;#38; Fred &amp;amp; Joe übertragen werden.</text>
   ```

4. Der XML-Prozessor parst den resultierenden Code, sieht die Entity-Verweise und dereferenziert sie wie folgt:

```
<text>Der Job konnte Harry & Fred & Joe übertragen werden.</text>
```

Wie Sie aus diesem Beispiel erkennen, können Sie das Escape für das Markup mit Hilfe des Entity-Verweises (im Beispiel `&#38;`) oder als Zeichenverweis (`&amp;`) auf das vordefinierte Entity darstellen.

Synchrone Strukturen

Neben diesen Problemen gibt es noch eine sehr wichtige Einschränkung bei der Verwendung von Markup in Entities. In Kapitel 2 haben Sie erfahren, daß die logische und die physische Struktur in einem XML-Dokument synchron sein müssen.

Damals konnten Sie mit dieser Einschränkung noch nicht sehr viel anfangen, weil man sich kaum vorstellen kann, daß die beiden Strukturen *nicht* synchron sind. Deshalb zeigen wir Ihnen hier ein Beispiel, bei dem die beiden Strukturen asynchron werden: Die logische Struktur setzt sich aus den Elementen im XML-Dokument und im Ersatztext zusammen. Die physische Struktur setzt sich aus dem Dokument-Entity (dem Wurzel-Entity des XML-Dokuments mit dem Entity-Verweis) und dem internen Entity (dem Ersatztext) zusammen. Die beiden Objekt sind diskrete physische Entities, was XML betrifft, obwohl sie wie in diesem Fall tatsächlich innerhalb einer einzigen Datei abgelegt sind. Damit die beiden Strukturen synchron bleiben, muß jedes Element des Ersatztextes dort beginnen und auch dort enden (mit anderen Worten: innerhalb des Entities).

Das folgende wäre also erlaubt:

```
<!ENTITY sicher "&amp;#38;#60;emph&amp;#62;Harry&amp;#38;#60;/emph&amp;#62; und Joe">
<text>Der Job konnte &sicher; übertragen werden.</text>.
```

weil der dereferenzierte Entity-Verweis folgendes ergäbe:

```
<text>Der Job konnte <emph>Harry</emph> und Joe übertragen werden.</text>
```

Das folgende dagegen könnte zu vielen Problemen führen:

```
<!ENTITY unsicher ""&#38#60;emph&#62;Harry und Joe">
<text>Der Job konnte &unsicher;</emph> übertragen werden.</text>
```

Auch wenn der Entity-Verweis dereferenziert würde, wäre das resultierende Markup noch irgendwie gültig:

```
<text>Der Job wurde <emph>Harry und Joe</emph>übertragen.</text>
```

Obwohl wir immer noch über *interne* Entities sprechen, die völlig unserer Kontrolle unterliegen, sind die Einschränkungen relativ logisch. Derselbe Dereferenzierungsmechanismus wird für externe Entities angewendet, und wenn man das Entwurfsziel der einfachen Nutzung im Web von XML berücksichtigt, haben wir absolut keine Kontrolle darüber, was in externen Entities enthalten ist. Die XML-Entwickler hätten einen Unterschied zwischen internen und externen Entities einführen können, aber das würde gegen zwei der wichtigsten Entwurfsziele von XML verstoßen – Einfachheit und Klarheit.

Wo Entities deklariert werden

Sie haben jetzt erfahren, wie ein interner Entity-Verweis aussieht, und Sie haben einige der Vorteile und Nachteile bei der Verwendung von Entity-Verweisen gesehen. Bevor wir weitergehen, müssen Sie noch lernen, wo die Entity-Deklarationen erfolgen.

Entity-Verweise sind normalerweise nur in der DTD des XML-Dokuments erlaubt. Die Deklarationen von Elementstrukturen und Entities sind der einzige Grund dafür, warum überhaupt eine DTD verwendet wird. Später erhalten Sie detailliertere Informationen über DTDs; alles, was Sie hier wissen müssen, finden Sie in Listing 3.3 erklärt.

Listing 3.3: Deklaration eines internen Entity

```
1:  <?xml version="1.0"?>
2:  <!DOCTYPE home.page [
3:    <!ENTITY shortcut "Dies ist der Ersatz.">
4:  ]>
5:  <home.page>
```

Zeile 1 von Listing 3.3 enthält die jetzt schon bekannte XML-Deklaration. Zeile 2 ist eine Dokumenttypdeklaration. Diese Zeile wird später für die Zuordnung zwischen dem XML-Dokument und der DTD, die seine Struktur beschreibt, herangezogen.

Die Dokumenttypdeklaration ist die XML-Anweisung, die deklariert, welche Art XML-Dokument folgt, und die die DTD (Dokumenttypdefinition) angibt, die die Beschreibung der für diesen Dokumenttyp erlaubten Struktur enthält. (Diese beiden Begriffe werden leicht verwechselt.)

Die Dokumenttypdeklaration hat die folgende Form:

```
<!DOCTYPE name externer.zeiger [ interne.untermenge ]>
```

Dabei zeigt `externer.zeiger` auf eine separate Datei, die die externe Untermenge der DTD enthält. Machen Sie sich hier noch keine Gedanken darüber; der Trick dabei ist, daß Sie den Zeiger auch weglassen und sich auf die interne Untermenge der DTD konzentrieren können. In diesem Fall brauchen Sie die folgende Deklaration:

```
<!DOCTYPE name [ interne.untermenge ]>
```

In dieser internen Untermenge können Sie beliebige viele Elemente, Attribute und Entities deklarieren, ohne eine externe DTD zu benötigen.

Wie Sie später noch erfahren werden, gibt es zahlreiche weitere Tricks, die mit der internen DTD-Untermenge möglich sind. Alles, was Sie in der internen Untermenge ablegen, hat Priorität vor allen Inhalten externer Untermengen. Beispielsweise können Sie eine Standardmenge globaler Werte für eine ganze Folge von XML-Dokumenten deklarieren und dann die globalen Werte in jedem einzelnen XML-Dokument ganz individuell überschreiben, aber das ist eine andere Geschichte.

Bevor wir das Thema der DTDs verlassen, gibt es noch eines, was Sie sich angewöhnen sollten, auch wenn es hier scheinbar noch keinen Sinn ergibt. Sie verwenden zwar noch keine externe DTD, aber wenn Sie es tun, muß der Name, den Sie dem Dokumenttyp geben, derselbe wie der Name des Wurzel-Elements im XML-Dokument sein. Das sehen Sie in Listing 3.3, wo der Dokumenttypname (`home.page` in Zeile 2) derselbe wie der (erste) Wurzel-Elementname (Zeile 5) ist. Das ist nicht erforderlich, wenn es keine externe DTD gibt, aber Sie sollten sich diesen Stil dennoch angewöhnen.

CDATA-Abschnitte

Sie haben erfahren, wie man ein Escape für Markup-Zeichen anlegt, und zwar indem man die vordefinierten Entities und Zeichenverweise nutzt. Das Ersetzen jedes Markup-Zeichens in einem Textabschnitt könnte ein langwieriger und mühseliger Prozeß sein. Darüber hinaus könnte es Situationen geben, wo Sie möchten, daß diese Zeichen so bleiben, wie sie sind (etwa wenn Sie den XML-Code für die weitere Verarbeitung durch eine andere Applikation weitersenden).

In diesem Fall verwenden Sie einen CDATA-Abschnitt (Character Data), etwa wie folgt:

```
<![CDATA[Dies ist der Text < 5 Zeilen >, den der &!%# XML-Prozessor nicht
verändern soll!]]>
```

Nichts, absolut nichts, was zwischen dem öffnenden Tag (`<![CDATA[`) und dem schließenden Tag (`]]>`) steht, wird als Markup erkannt. Sie müssen kein Escape für Markup-Zeichen in einen CDATA-Abschnitt einfügen. (Sie können das nicht einmal, weil auch das Escape nicht erkannt würde.) Das einzige, was erkannt wird, ist das Tag für das Abschnittsende (`/]]>`); diese Zeichenkette darf also offensichtlich nicht in einem CDATA-Abschnitt erscheinen. Als logische Konsequenz erkennen Sie, daß CDATA-Abschnitte nicht ineinander verschachtelt werden dürfen.

Die Verwendung von Markup-Zeichen in einem solchen CDATA-Abschnitt in einem XML-Dokument, das um das Markup herum aufgebaut ist, geht einem eigentlich gegen den Strich. Ein XML-Prozessor soll Sie daran hindern, dieses ungeschriebene Gesetz zu brechen, und er verzeiht keine Fehler. Die öffnende und die schließende Zeichenkette eines CDATA-Abschnitts müssen genau so geschrieben werden wie hier gezeigt. Die kleinste Abweichung, ein Leerzeichen oder ein Tabulator irgendwo innerhalb der Zeichenkette, wird sofort bestraft. Der Inhalt des CDATA-Abschnitts wird dann entweder als Markup betrachtet, oder Ihr restliches Dokument (bis zum nächsten korrekt abgeschlossenen CDATA-Abschnitt) wird als Teil des CDATA-Abschnitts betrachtet, und das gesamte Markup wird ignoriert. Seien Sie also gewarnt!

CDATA-Abschnitte sind eine der empfohlenen Methoden, Applikationscode (JavaScript, VBasic-Code, Perl-Code usw.) in Ihren XML-Code einzubetten. Sie könnten den eingebetteten Code auch in Kommentare einschließen, aber es ist nicht garantiert, daß der XML-Prozessor einer Applikation auch den Kommentartext übergibt. Es besteht also das Risiko, daß der Kommentarinhalt entfernt wird, bevor die Applikation den Code sieht.

Es ist zwar durchaus erlaubt, Ihren eigenen Elementtyp zu deklarieren, der den eingebetteten Code aufnimmt (wie das `<script>`-Element in HTML 4), damit brechen Sie aber implizit den Geist des generischen Markups. Es wäre auch nicht hilfreicher, wenn Ihr eingebetteter Code Zeichen enthielte, die als Markup interpretiert werden könnten, weil der Inhalt dieser Elemente vom XML-Prozessor auf die normale Weise interpretiert würde.

Die andere, vielleicht sogar beste, Methode, Code einzubetten, ist die Verwendung von Verarbeitungsanweisungen.

Verarbeitungsanweisungen (Processing Instructions)

Sie haben sicher schon Verarbeitungsanweisungen gesehen – ohne sie als solche zu erkennen. Die XML-Deklaration am Anfang jedes XML-Dokuments (die sich dort zumindest befinden *sollte*) ist eine Verarbeitungsanweisung:

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

XML-Markup soll generisch sein, und üblicherweise ist es das auch. Es gibt jedoch immer Situationen, in denen Sie Anweisungen für bestimmte Applikationen eingeben müssen. Eine solche Applikation könnte etwa ein Skript-Interpreter sein, deshalb sind Verarbeitungsanweisungen ähnlich wie CDATA-Abschnitte gut dafür geeignet, Code einzubetten. Während CDATA-Abschnitte einzig eine Methode darstellen, die Interpretation von Zeichen als Markup zu vermeiden, sind Verarbeitungsanweisungen direkt auf Ihre Applikation zugeschnitten, was noch viel besser ist. Beispielsweise könnten Sie auf diese Weise zwei oder mehr Mengen eingebetteten Skriptcodes haben, die jeweils für unterschiedliche Prozessoren oder Interpreter vorgesehen sind, und sie separat kennzeichnen, wie in Listing 3.4 gezeigt.

Listing 3.4: Eingebetteter Code in Verarbeitungsanweisungen

```
1:  <para>Dieser Text enthält zwei
2:  Verarbeitungsanweisungen,
3:    <?javascript Kann ich hier schreiben, was ich will?>
4:    <?perl Und hier auch?>
5:  eine für jeden Interpreter.</para>
```

Es gibt keine Einschränkungen bezüglich des Inhalts von Verarbeitungsanweisungen (der XML-Prozessor betrachtet den Inhalt nicht einmal als Bestandteil der Zeichendaten des Dokuments), aber der dafür gewählte Name muß den Namensregeln von XML gehorchen.

Zusammenfassung

In diesem Kapitel haben Sie Details über die Markup-Sprache XML kennengelernt. Außerdem haben Sie erfahren, wie man interne Entities deklariert und nutzt und welche Vorteile und Gefahren sie bergen. Sie haben die Zeichenverweise kennengelernt, mit denen Zeichen eingegeben werden können, die sich nicht auf Ihrer Tastatur befinden, und Sie haben gesehen, wie Sie mit Hilfe von Zeichenverweisen und vordefinierten Entities in Ihren Zeichendaten die Zeichen nutzen können, die normalerweise für das Markup reserviert sind.

Schließlich haben Sie erfahren, wie man Kommentare und CDATA-Abschnitte nutzt, um Text zu verbergen, der durch den XML-Prozessor als Markup interpretiert werden könnte, und wie man das Ganze noch erweitert, indem man mit Hilfe von Verarbeitungsanweisungen Code weitergeben kann, der für die Verarbeitung durch andere Applikationen vorgesehen ist.

F&A

F Welche der folgenden Elementnamen sind gültig, welche nicht?

a) `<para 1>`

b) `<para,1>`

c) `<para.1>`

d) `<Pa3A1>`

e) `<para!>`

A *Nur c und d sind erlaubt; a enthält ein Leerzeichen, b enthält ein Komma, und e enthält ein Ausrufezeichen.*

F Was stimmt im folgenden Codeabschnitt nicht?

```
<para size="12pt">'twas brillig and
       the slithey toves <!-- I've no idea
        what these are --> did gyre and gymble
        in the wabe.</para>
```

A *Kommentare dürfen nicht innerhalb von Elementen angelegt werden. Sie müssen sich außerhalb von Markup befinden.*

F Wo werden Entities deklariert?

A *Sie können Entities innerhalb der internen Untermenge oder der externen Untermenge der DTD deklarieren. Wenn Sie eine externe DTD verwenden, müssen Sie eine vollständige DTD anlegen. Falls Sie nur die Entities und sonst nichts brauchen, können Sie mit der internen DTD-Untermenge auskommen. Entity-Verweise in XML-Dokumenten, die externe DTD-Untermengen haben, werden nur ersetzt, wenn das Dokument analysiert wird.*

F Warum brauche ich eine XML-Deklaration? Es ist doch offensichtlich, daß es sich um XML-Code handelt.

A *Genau gesagt, braucht man gar keine XML-Deklaration. XML wurde auch als MIME-Typ anerkannt, d.h. wenn Sie den korrekten MIME-Header (*`xml/text` *oder* `xml/application`*) einfügen, kann ein Web-Server die darauffolgenden Daten explizit als XML-Dokument erkennen, unabhängig davon, was in dem Dokument steht. (MIME, Multipurpose Internet Mail Extensions, ist ein Internet-Standard für die Übertragung von Daten beliebigen Typs über E-Mail. Er definiert, wie Nachrichten formatiert und aufgebaut werden, kann den Typ und die Art des Nachrichteninhalts anzeigen und Informationen des internationalen Zeichensatzes beibehalten. MIME-Typen werden von Web-Servern verwendet, um die Daten in einer Antwort auf eine Suchabfrage zu kennzeichnen.)*

Die XML-Deklaration ist aus praktischen Gründen nicht zwingend; SGML- und HTML-Code kann häufig ganz einfach in perfekten XML-Code umgewandelt werden (falls er es noch nicht ist). Wäre die XML-Deklaration zwingend, wäre das nicht möglich.

F Kann ich Entities in Attributwerten sowie im Inhalt verwenden? Damit könnte ich Elemente parametrisieren.

A *Ja und nein. Sie können Entity-Verweise als Attributwerte angeben, aber ein Entity kann kein Attributwert sein. Es gibt strenge Regeln darüber, wo Entities verwendet werden können, und wann sie erkannt werden. Manchmal werden sie nur erkannt, wenn das XML-Dokument ausgewertet wird. Weitere Informationen finden Sie in der XML-Empfehlung unter* `http://www.w3.org/XML/`

F Kann ich binäre Daten in einen CDATA-Abschnitt schreiben?

A *Technisch gesehen gibt es keinen Hinderungsgrund, obwohl es sich eigentlich um einen Zeichendatenabschnitt handelt. Weil der XML-Prozessor den Inhalt eines CDATA-Abschnitts nicht als Teil der Zeichendaten des Dokuments betrachtet, weiß er nicht, was Sie dort geschrieben haben, und es ist ihm auch egal. Sie müßten sonst auch mit größeren Dateien und allen möglichen Übertragungsproblemen leben. Letztlich wäre es unsinnig, die Portabilität Ihrer XML-Dokumente zu gefährden, wenn es eine XML-Funktion gibt, die Sie für diese Zwecke nutzen können. Entities, die Sie in Kapitel 8 genauer kennenler-*

nen, erlauben Ihnen, ein Format und eine Hilfsanwendung für die Verarbeitung einer Binärdatei (möglicherweise eine Anzeige) und ihre Zuordnung zu einem XML-Dokument per Verweis zu deklarieren.

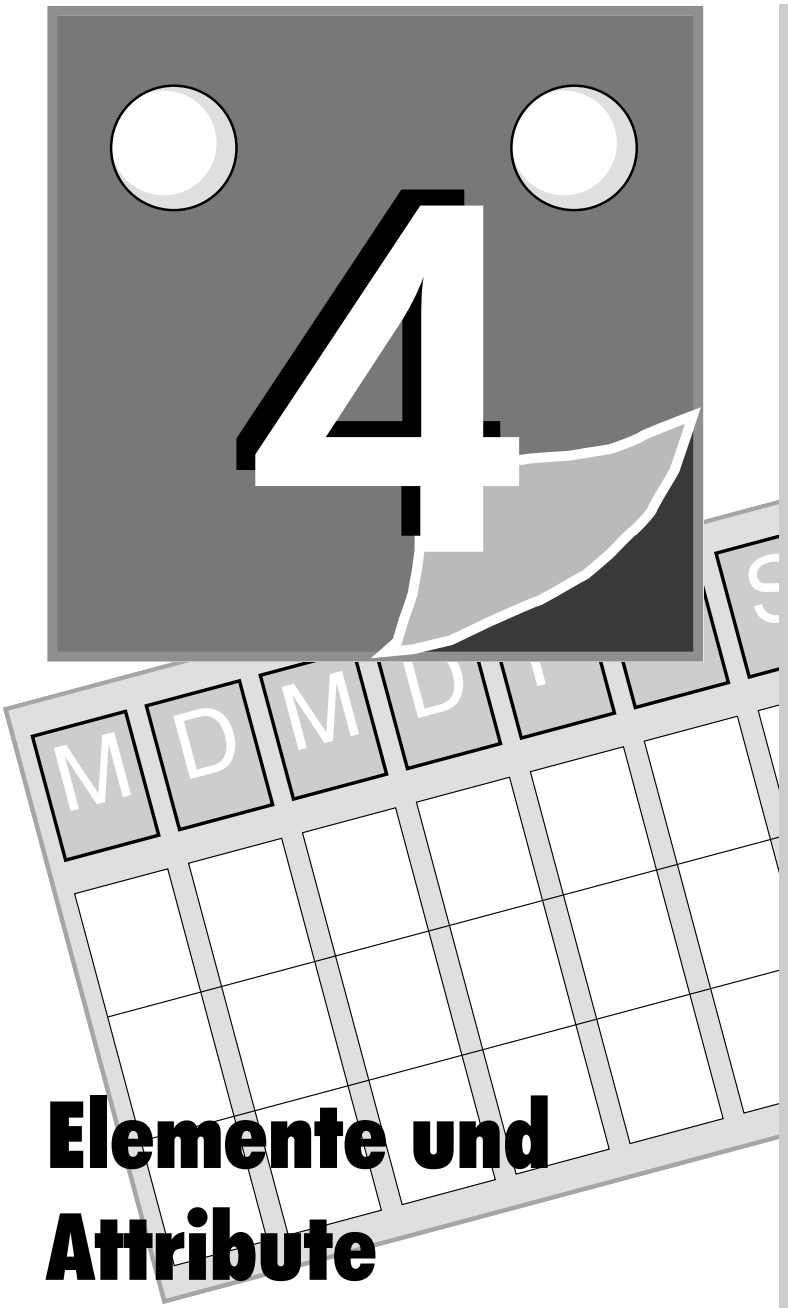
Übungen

1. Der folgende Codeausschnitt enthält zwei Fehler. Erkennen Sie sie?

   ```
   <![CDATA [Dies ist das verborgene &Markup!] ]>
   ```

   Überprüfen Sie Ihre Antworten, indem Sie den Code in einem der XML-Parser ausführen, was in Kapitel 5 noch weiter erklärt wird.

2. Gestern haben Sie das Markup für eine E-Mail-Nachricht angelegt. Ändern Sie das Markup unter Verwendung geeigneter Entities so, daß der XML-Code zu einer Grundlage für beliebige E-Mail-Nachrichten wird.



Elemente und Attribute

Woche 1

In Kapitel 3 haben Sie die Grundlagen der XML-Spezifikationen und ihre Verwendung zur Kennzeichnung von Elementen und Attributen in einem XML-Dokument kennengelernt. Anhand der internen Untermenge haben Sie dann, ohne wirklich eine externe DTD erzeugen zu müssen, erfahren, wie man interne Entity-Deklarationen anlegt, die im XML-Dokument als eine Art Makro verwendet werden können.

Heute werden Sie dieses Wissen über Markups erweitern und die folgenden Dinge kennenlernen:

- Element- und Attributdeklarationen in die interne DTD-Untermenge einfügen
- Elementinhaltsmodelle
- Grundlagen der Dokumentmodellierung
- Die Prinzipien wohlgeformter XML-Elemente

Mit diesem Wissen gerüstet, können Sie die ersten Ausflüge in die Welt der Informationsmodellierung unternehmen. Dort können Sie unter Verwendung der Elementinhaltsmodelle, die Sie in XML deklarieren, den Inhalt von XML-Dokumenten steuern und überprüfen, die mit den von Ihnen angelegten DTDs konform sind.

Am Ende des heutigen Kapitels sollten Sie in der Lage sein, schon relativ komplexe XML-Dokumente anzulegen und sicherzustellen, daß diese wohlgeformt sind. Danach können Sie weiter zu Tag 5 gehen, wo Sie prüfen, ob Sie alles Gelernte richtig angewendet und korrekte XML-Dokumente erzeugt haben.

Markup-Deklarationen

Bevor wir die Deklaration von Elementen und Attributen detailliert betrachten, wollen wir noch einmal kurz darauf eingehen, wo diese Deklarationen im XML-Dokument erfolgen. Listing 4.1 zeigt das Grundgerüst für die Deklaration einer internen DTD-Untermenge in einem XML-Dokument.

Listing 4.1: Die Deklaration einer internen DTD-Untermenge

```
1:  <?xml version="1.0"?>
2:  <!DOCTYPE page [
3:    <!--hier befindet sich die interne DTD-Untermenge. -->
4:  ]>
```

```
5:  <page>
6:    <!-- dies ist der Inhalt des (einzigen) Elements -->
7:  </page>
```

Wie in Listing 4.1 gezeigt, beginnt das XML-Dokument mit der XML-Deklaration (Zeile 1). In dieser Phase arbeiten Sie noch ohne externe DTD; die hier gezeigte Deklaration ist deshalb ausreichend.

Zeile 2 ist der Anfang der `DOCTYPE`-Deklaration, die in Zeile 4 endet. Alle Markup-Deklarationen für das XML-Dokument werden also zwischen den eckigen Klammern ([]) eingegeben.

Obwohl die vollständige Syntax etwas komplexer ist als hier beschrieben (Sie erhalten detailliertere Informationen über die Syntax, wenn Sie diese brauchen), hat sie, wenn sie nur mit einer internen DTD-Untermenge benutzt wird, die folgende Form:

```
<!DOCTYPE dokument.typ.name [ interne.untermenge ]>
```

Dabei ist der Dokumenttypname derselbe wie der Name des Wurzelelements (`<page>` in Listing 4.1).

In dieser Phase ist es nicht erforderlich, daß der Dokumenttypname gleich dem Wurzelelementnamen ist, aber später wird es zwingend erforderlich, sobald Sie Ihr XML-Dokument auswerten. Sie sollten sich deshalb angewöhnen, dieses Namensschema immer zu befolgen, weil Sie sich damit unnötige Überarbeitungen ersparen.

Elementdeklarationen

Die erste Deklarationsart, die Sie innerhalb einer DTD benutzen, egal ob interne oder externe Untermenge, ist die Elementdeklaration. Sie hat die folgende Form:

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

Der Name ist ein Standard-XML-Name, der mit den Namensregeln übereinstimmt, die Sie gestern kennengelernt haben.

Der Teil, der den Inhalt der Elementdeklaration bildet, beschreibt entweder einen spezifischen Inhalt in Form der Schlüsselwörter `EMPTY` oder `ANY` oder aus einem Inhaltsmodell, das sich aus einer Abfolge und Wiederholung von Elementen innerhalb (untergeordnete Elemente) dieses Elements aufbaut.

Bevor wir auf diese speziellen Elementdeklarationen eingehen, wollen wir ein ganz einfaches Beispiel einer E-Mail-Nachricht betrachten, wie in Listing 4.2 gezeigt.

Listing 4.2: Eine sehr einfache E-Mail-Nachricht in XML

```
1:  <message>
2:    <header>
3:      <date>14 May 1998</date>
4:      <From>Me</From>
5:      <To>You</To>
6:      <Subject>Test Message</Subject>
7:     </header>
8:     <Body> ...
9:     </Body>
10:   <Sig>Irgendwas Cleveres
11:   </Sig>
12: </message>
```

Sie wissen noch nicht genug, um den Inhalt der Elemente bis auf die Ebene des eigentlichen Texts zu deklarieren. Aber mit Ihren jetzigen Kenntnissen sollten Sie in der Lage sein, Inhaltsmodelle zu schaffen, wie in Listing 4.3 gezeigt.

Listing 4.3: Teil einer DTD für die E-Mail-Nachricht

```
1: <!ELEMENT message (header, body, sig) >
2: <!ELEMENT Header (date, from, to, subject) >
```

Leere Elemente

Wie Sie in Kapitel 3 erfahren haben, haben leere Elemente keinen Inhalt (sie dürfen keinen Inhalt haben) und werden wie folgt ausgezeichnet:

```
<leeres.element/>
```

oder

```
<leeres.element></leeres.element>
```

Ein leeres Element wird einfach wie folgt deklariert:

```
<!ELEMENT leeres.element EMPTY>
```

Sie könnten leere Elemente für Dinge wie externe Grafiken verwenden (wo Sie, wie Sie noch erfahren werden, den Namen der externen Datei deklarieren, indem Sie ihn als Attributwert verwenden).

Unbeschränkte Elemente

Das Gegenteil eines leeren Elements ist ein unbeschränktes Element, das beliebig viele Elemente enthalten kann, die an anderer Stelle in der DTD des XML-Dokuments deklariert sind (in der internen oder in der externen DTD-Untermenge). Sie verwenden jetzt noch keine externe DTD-Untermenge; deshalb gibt es offensichtlich keine Möglichkeit, wie der XML-Interpreter etwas über Elemente erfahren könnte, die darin deklariert sind.

Der Inhalt eines unbeschränkten Elements wird wie folgt deklariert:

```
<!ELEMENT beliebiges.element ANY>
```

Offensichtlich können Sie den Inhalt nicht in einer bestimmten Reihenfolge deklarieren.

Elementinhaltsmodelle

Leere Elemente und unbeschränkte Elemente sind Spezialfälle, die so einfach sind, daß sie mehr oder weniger für sich selbst sprechen; entweder erlauben sie alles, oder sie erlauben nichts.

Sehr viel interessanter sind für Sie die Elemente, die weitere Elemente enthalten. Mit Hilfe der Elementdeklarationen von XML können Sie genau angeben, welche anderen Elemente innerhalb eines Elements erlaubt sind, wie oft sie dort erscheinen dürfen und in welcher Reihenfolge. Dazu spezifizieren Sie ein Elementinhaltsmodell.

Wie Sie später erfahren werden, reicht es nicht aus, daß Sie spezifisch sein wollen – Sie müssen spezifisch sein. Eines der Dinge, mit denen ein XML-Interpreter (oder ein SMGL-Parser) nicht zurechtkommt, ist Mehrdeutigkeit. Ihre Inhaltsmodelle müssen so gestaltet sein, daß sie nur auf genau eine Weise interpretiert werden können. Das bedingt ein bißchen Erfahrung und kann Ihnen viel Kopfzerbrechen bereiten, insbesondere am Anfang. Sie werden im Laufe dieses Buchs noch mehr über mehrdeutige Inhaltsmodelle erfahren.

Ein Elementinhaltsmodell besteht aus einer einfachen, aber sehr spezifischen Beschreibung der Elemente, die im aktuellen Element auftreten dürfen, der Reihenfolge, in der sie dort erscheinen dürfen oder müssen, und wie oft sie erscheinen dürfen oder müssen. Darüber hinaus können Sie die Elemente in Gruppen anordnen und spezielle Container-Objekte anlegen. Sie können die Elemente in einer ganzen Klasse von XML-Dokumenten (dem Dokumenttyp) anlegen, um die Bedeutung und die Beziehungen zwischen den Informationsabschnitten zu reflektieren.

Die Reihenfolge der Elemente

Die einfachste Form eines Elementinhaltsmodells besteht aus einer Liste der möglichen Elemente, eingeschlossen in Klammern und durch Kommas voneinander abgetrennt:

```
<!ELEMENT zählen (erstes, zweites, drittes, viertes)>
```

Dieses Beispiel sagt, daß ein `zählen`-Element aus einem `erstes`-Element, gefolgt von einem `zweites`-Element, gefolgt von einem `drittes`-Element und abgeschlossen durch ein `viertes`-Element bestehen muß.

In diesem Beispiel müssen alle vier Elemente in einem `zählen`-Element vorhanden sein, und jedes davon darf nur ein einziges Mal vorkommen. (Sie können angeben, wie oft ein Element erscheinen darf, indem Sie eine Anzeige für das Vorkommen einführen, was Sie später in diesem Kapitel noch lernen werden.)

Optionale Elemente

Optionale Elemente in einem Elementinhaltsmodell werden durch einen vertikalen Balken (¦) zwischen den Alternativen gekennzeichnet:

```
<!ELEMENT auswahl (dieses.hier ¦ jenes.dort)>
```

In diesem Beispiel besteht ein `auswahl`-Element entweder aus einem `dieses.hier`-Element oder aus einem `jenes.dort`-Element. Wenn Sie diese Elemente in einem XML-Dokument einsetzen, können Sie es wie folgt schreiben:


```
<auswahl><dieses.hier>Ich wähle dieses hier</dieses.hier></auswahl> und
dann <auswahl><jenes.dort>Ich wähle jenes dort</jenes.dort></auswahl>
```

Beachten Sie auch hier, daß das ausgewählte Element ohne einen Kennzeichner für das Vorkommen nur einmal auftreten darf.

Beachten Sie außerdem, daß nur ein Element ausgewählt werden kann, unabhängig davon, wie lang die Liste der Alternativen ist:

```
<!ELEMENT auswahl (dieses.hier ¦ jenes.dort ¦ das.ganz.andere ¦ noch.ein.anderes ¦
nicht.dieses.schreckliche)>
```

Kombinierte Abfolgen und Auswahlen

Sie kombinieren Inhaltsabfolgen und -auswahlen, indem Sie den Elementinhalt in Modellgruppen organisieren, beispielsweise so:

```
<!ELEMENT ganz.viele.auswahlen (vielleicht ¦ könnte.sein), (dieses.hier ¦
jenes.dort)>
```

Hier kann das Element `ganz.viele.auswahlen` aus einem `vielleicht`-Element oder einem `könnte.sein`-Element bestehen, gefolgt von einem `dieses.hier`-Element und dann einem `jenes.dort`-Element.

Mehrdeutige Inhaltsmodelle

Sie können Abfolgen und Auswahlen in Elementinhaltsmodellen kombinieren, aber gehen Sie dabei sehr sorgfältig vor. Es ist zwar nicht zwingend erforderlich in XML (oder zumindest nicht, wenn Sie Ihre Dokumente nicht auswerten), aber es können sich Kompatibilitätsprobleme ergeben, wenn Ihre Inhaltsmodelle unterschiedlich interpretiert werden können.

Betrachten Sie die folgende Möglichkeit:


```
<!ELEMENT verwechslung ((dieses.hier, jenes.dort) ¦ (dieses.hier,
das.ganz.andere))>
```

Wenn der XML-Interpreter das XML-Dokument auswertet (dabei wird der Inhalt daraufhin überprüft, ob die Elemente in einer erlaubten Reihenfolge vorliegen), ist er nicht in der Lage zu entscheiden, was erlaubt ist, und was nicht. Wenn er ein `dieses.hier`-Element gesehen hat, kann er nicht feststellen, welches Element als nächstes kommen soll.

Natürlich könnte der XML-Interpreter weiterlesen und prüfen, ob das, was auftaucht, erlaubt ist, aber XML-Interpreter sind nicht darauf ausgelegt, weiterzulesen. Sie sollen einfach und schnell sein. Wenn der Prozessor vorauslesen könnte, müßte er das, was er dort findet, im Speicher ablegen, den nächsten Teil lesen, ihn im Speicher ablegen, die beiden Speicherinhalte vergleichen und dann eine Entscheidung treffen. Und all das erfordert zusätzliche Verarbeitungszeit.

Durch sorgfältige Überlegung können Sie mehrdeutige Inhaltsmodelle durch eine einfache Überarbeitung vermeiden:

```
<!ELEMENT nicht verwechselbar ( dieses.hier, (jenes.dort ¦ das.ganz.andere ) ) >
```

Im allgemeinen sollten Sie aber bei jeder Kombination dieser beiden Operatoren darauf achten, keine Mehrdeutigkeiten zu produzieren.

Betrachten wir noch ein Beispiel:

```
<!ELEMENT noch.eine.verwechslung ( dieses.hier, jenes.dort, das.ganz.andere ¦
dieses.nicht ) >
```

Das könnte Sie (und den XML-Interpreter) ganz einfach zu dem Glauben verleiten, daß das `dieses.nicht`-Element eine Alternative zu allen anderen Elementen ist. Oder ist es einfach nur eine Alternative zum `das.ganz.andere`-Element?

Auch hier kann eine Überarbeitung die Mehrdeutigkeit aufheben und Ihre Absicht ein bißchen deutlicher machen:

```
<!ELEMENT erklärt ( dieses.hier, jenes.dort, (das.ganz.andere ¦ dieses.nicht ) ) >
```

Sie haben vielleicht bemerkt, daß eines meiner mehrdeutigen Elementinhaltsmodelle nicht ganz mehrdeutig war; der XML-Interpreter wäre in der Lage gewesen, eine sinnvolle Anwendung abzuleiten. Sie erinnern sich vielleicht, daß eines der Entwurfsziele von XML war, für Menschen möglichst leichtverständlich zu sein, ohne daß sie dazu die Hilfe von Software brauchen. Es ist eine ausgezeichnete Idee, dasselbe Prinzip auf XML-DTDs zu erweitern. Falls es Ihnen möglich ist, organisieren Sie Ihre Inhaltsmodelle in Gruppen, so daß leichter verständlich ist, was sie bedeuten. Schließlich können Sie nicht vorhersehen, wann Ihnen die DTD wieder begegnen wird, so daß Sie zu diesem Zeitpunkt möglicherweise bereits alle diese wunderbaren Begründungen vergessen haben, die den Entwurf Ihres dann kaum noch nachvollziehbaren Elementinhaltsmodells erklären.

Angaben zum Vorkommen der Elemente

Durch eine Angabe für das Elementvorkommen können Sie festlegen, wie oft ein Element oder eine Elementgruppe innerhalb eines Elements auftreten dürfen. Es gibt drei solcher Angaben für das Vorkommen (ohne diese Angabe darf das Element oder die Elementgruppe nur einmal auftreten):

- Das Fragezeichen ? zeigt an, daß das Element oder die Elementgruppe einmal oder keinmal auftreten darf. Betrachten Sie folgendes Inhaltsmodell:

  ```
  <!ELEMENT testen (eins, zwe?, drei)>
  ```

- Damit könnten Sie in Ihrem Dokument schreiben:

  ```
  <testen><eins>klopf</eins><zwei>klopf</zwei><drei>klopf</drei></testen>
  ```

- oder

  ```
  <testen><eins>klopf</eins><drei>klopf</drei></testen>
  ```

- Der Stern * gibt an, daß ein Element oder eine Gruppe keinmal oder nullmal oder öfter vorkommen darf. Betrachten Sie das folgende Inhaltsmodell:

  ```
  <!ELEMENT wunderbar (mmm, mmmm*)>
  ```

- Damit könnten Sie folgendes in Ihrem XML-Dokument schreiben:

```
<wunderbar><mmm>Ich kann nicht klagen.</mmm></wunderbar>
```

- oder

```
<wunderbar><mmm>Ich kann nicht klagen.</mmm><mmmm>Mehr, </mmmm><mmmm>mehr,
<mmmm>mehr, </mmmm><mmmm>mehr, <mmmm> mehr.</mmmm></testen>
```

- Das Pluszeichen + zeigt an, daß ein Element oder eine Elementgruppe mindestens einmal und beliebig oft vorkommen soll. Betrachten Sie das folgende Inhaltsmodell:

```
<!ELEMENT lustig (ha, haha+)>
```

- Damit könnten Sie in Ihrem XML-Dokument folgendes schreiben:

```
<lustig><ha>Wer?</ha><haha>ist er</haha></lustig>
```

- oder

```
<lustig><ha>Ich lachte </ha><haha>bis </haha><haha>ich </haha><haha>dachte, </
haha><haha>Ich würde <haha>sterben!</haha></lustig>
```

Wie Sie sehen, bieten Ihnen die Angaben zum Vorkommen der Elemente ein wenig Kontrolle darüber, wie oft ein Element oder eine Elementgruppe auftritt – überhaupt nicht, einmal oder beliebig oft. Dieser »Alles oder Nichts«-Ansatz ist etwas zu locker für viele mögliche XML-Applikationen.

Eine der Ankündigungen von XML war, daß es schließlich möglich sein sollte, damit ernsthaft Datenbanken zu modellieren. (Das war in SGML ein praktisch unerreichbarer Traum.) Die Syntax von XML-Elementinhaltsmodellen ist jedoch nicht umfassend und präzise genug, um die genaue Anzahl eines Elementvorkommens anzugeben, außer nur 0, 1 oder unendlich.

Ein weiteres Beispiel, was die Programmierer gerne hätten, ist eine Art bedingten Inhaltsmodells nach dem Konzept: »Falls Element A und danach B, dann C oder D; andernfalls E oder F«.

Komplexe Inhaltsmodelle wie diese Beispiele können einfach nicht in der Syntax von XML-DTDs ausgedrückt werden. Um dieses Problem zu lösen und die Anforderungen eines viel breiteren Benutzerkreises zu erfüllen, hat eine Suche nach alternativen Methoden zur Beschreibung und Deklaration von XML-Inhaltsmodellen (DCD, XML-Data und RDF) begonnen. Sie lernen einige der wichtigsten dieser Schemata in Kapitel 17 kennen.

Zeicheninhalt

Es gibt noch eine Art Zeicheninhalt, den ich absichtlich bis zum Schluß aufgehoben habe, weil er einen Sonderfall darstellt.

Wenn Text – und nur Text – in einem Element erlaubt ist, wird das im Inhaltsmodell durch das Schlüsselwort `PCDATA` (*Parseable Character Data*) gekennzeichnet. Um zu verhindern, daß Sie dieses Schlüsselwort mit einem normalen Elementnamen verwechseln (und Ihnen unmöglich zu machen, es als Namen zu verwenden), geht dem Schlüsselwort ein Pfundzeichen (#) voraus, das Zeichen für die reservierten Namen (RNI, *Reserved Name Character*).

Betrachten Sie die folgenden Elementdeklarationen:

```
<!ELEMENT para (title, text)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT text (#PCDATA)>
```

Damit könnten Sie in Ihrem XML-Dokument folgendes schreiben:

```
<para><title>Mein Leben</title><text>Mein Leben war in der
letzten Zeit sehr ruhig.</text></para>
```

Ein Zeichendatenelement, das geparst werden kann, darf keine weiteren Markups enthalten, und das Ende der Start-Tags für die `title`- und `text`-Elemente ist deshalb dort, wo das Markup endet und der »normale« Text beginnt.

Vergessen Sie nicht, daß die Inhaltsmodelle von XML nur die Struktur eines XML-Dokuments betreffen und nicht versuchen, seinen Inhalt zu steuern. Ein Element, das überhaupt keinen Dateninhalt hat, kann dennoch mit einem `#PCDATA`-Inhaltsmodell übereinstimmen.

Elemente mit gemischtem Inhalt

Elemente, die Text (Zeichenelemente, die geparst werden können), Elemente oder beides enthalten, können manchmal ein Problem darstellen. Sie werden als gemischte Inhaltsmodelle bezeichnet und bedürfen einer Sonderbehandlung.

Achten Sie darauf, daß es für einen XML-Interpreter schwierig ist, zwischen unbeabsichtigtem `PCDATA` (Leerzeichen, Tabulatoren, Zeilenenden usw.) und Elementinhalt zu unterscheiden. Ein zusätzliches Leerzeichen zwischen einem Ende-Tag und dem nächsten Start-Tag kann zum totalen Chaos führen.

Für die Deklaration des gemischten Inhalts verwenden Sie die Inhaltsmodellgrammatik, die Sie bereits kennen, aber auf eine ganz bestimmte Weise. Das Inhaltsmodell muß die Form einer einzelnen Alternativenmenge annehmen, beginnend mit `#PCDATA`

und gefolgt von den Elementtypen, die im gemischten Inhalt auftreten können und die je einmal deklariert sind. Außer wenn #PCDATA die einzige Option ist (wie Sie bereits gesehen haben), muß der schließenden Klammer der Bezeichner * folgen.

```
<!ELEMENT auswahl (#PCDATA ¦ ene ¦ mene ¦ runkelrübe)*>
```

Attributdeklarationen

In XML können Sie jeweils nur ein Element deklarieren. Elemente können jedoch zahlreiche Attribute haben, die alle gleichzeitig in einer Attributlistendeklaration deklariert werden können.

Eine Attributlistendeklaration hat die folgende Form:

```
<!ATTLIST element.name attribut.definitionen>
```

Es ist üblich, die Attributlistendeklaration für ein Element in der Nähe seiner Deklaration unterzubringen, es ist aber nicht zwingend erforderlich. Die DTD wird jedoch dadurch einfacher zu verstehen und zu warten.

Eine Attributlistendeklaration besteht aus einer oder mehreren Attributdeklarationen. (Der Lesbarkeit halber werden sie häufig in separaten Zeilen abgelegt, aber das ist nicht unbedingt nötig.) Sie bewirkt für ein Element folgendes:

- Sie deklariert die Namen der erlaubten Attribute.
- Sie gibt den Typ der verschiedenen Attribute an.
- Sie kann einen Vorgabewert für jedes Attribut bereitstellen.

Jede Attributdeklaration besteht aus einer Kombination aus einfachem Attributnamen und Attributtyp in der folgenden Form:

```
attribut.name attribut.typ
```

Attributtypen

Es gibt drei Attributtypen:

- Ein String-Attribut, dessen Wert aus beliebig vielen Zeichendaten besteht.
- Ein Token-Attribut, dessen Wert aus einem oder mehreren *Token* besteht, die für XML eine Bedeutung haben.
- Ein Aufzählungsattribut, dessen Wert aus einer Liste deklarierter möglicher Werte ausgewählt wird.

String-Attribute

Die Werte von String-Typen sind einfach nur Zeichenketten. Jedes Attribut, das in einem XML-Dokument verwendet wird und das keine DTD hat (weder eine interne noch eine externe DTD-Untermenge), wird automatisch als String-Attribut behandelt.

Hier ein Beispiel für die Deklaration eines String-Attributs:

```
<!ATTLIST buch eigentümer CDATA>
```

Das könnte wie folgt benutzt werden:

```
<buch eigentümer="Stadtbibliothek Deggendorf">
```

Sie können auch ein internes Entity (in diesem Fall hat es den generischeren Namen *allgemeines Entity*) als Wert eines String-Attributs verwenden:

```
<buch eigentümer="Stadtbibliothek; &my.local">
```

Token-Attribute

Token-Attribute werden abhängig von ihrem Wert bzw. ihren Werten klassifiziert:

- ID – Dieses Attribut dient als Bezeichner für das Element. Innerhalb eines Elements können keine Elemente denselben ID-Attributwert haben. Ein ID-Wert muß den Standardnamensregeln von XML entsprechen. Ein ID-Attributtyp kann auf jedes Attribut angewendet werden, aber in der Regel beschränkt man sich dabei auf Attribute, die auch als ID bezeichnet werden, so daß man sie leichter findet.

  Hier ein Beispiel für eine ID-Typdeklaration:

  ```
  <!ATTLIST buch
  id ID>
  ```

  Das könnten Sie wie folgt benutzen:

  ```
  <buch id="A51">
  ```

- IDREF – Dieses Attribut ist ein Zeiger auf eine ID (ein ID-Verweis). Der Wert muß mit dem Wert eines ID-Attributs übereinstimmen, das an anderer Stelle im selben Dokument deklariert ist.

- `IDREFS` - Der Wert dieses Attributs besteht aus einem oder mehreren `IDREF`-Typwerten, durch Leerzeichen voneinander getrennt.

  Hier ein Beispiel für eine `IDREFS`-Typdeklaration:

  ```
  <!ATTLIST buch
        autoren IDREFS>
  ```

Das könnten Sie wie folgt verwenden:

```
<buch autoren="A51 A62 B87">
```

- ENTITY – Dieses Attribut ist ein Zeiger auf ein externes Entity, das in der DTD deklariert wurde, entweder in der internen oder in der externen DTD-Untermenge. Der Wert des Attributs ist der Name des Entity, das aus Namenszeichen bestehen muß. Das XML-Dokument kann kein Standalone-Dokument mehr sein, wenn Sie externe Entities verwenden. Sie haben noch nicht die erforderlichen Grundlagen für externe Entities, aber wir werden in Kapitel 8 noch einmal darauf zurückkommen.
- ENTITIES - Der Wert dieses Attributs besteht aus einem oder mehreren ENTITY-Typwerten, durch Leerzeichen voneinander abgetrennt.

  Attribute vom Typ ENTITY oder ENTITIES werden normalerweise verwendet, um auf Dinge wie Grafikdateien oder nicht geparste Daten zu verweisen:

```
<!ELEMENT grafik EMPTY >
<!ATTLIST grafik    tafelbild   ENTITIY >
```

- NMTOKEN – Der Wert dieses Attributs ist ein Name-Token-String, der aus einer beliebigen Kombination von Namenszeichen besteht.
- NMTOKENS – Der Wert dieses Attributs besteht aus einem oder mehreren NMTOKEN-Typwerten, durch Leerzeichen voneinander getrennt.

Aufzählungsattribute

Aufzählungsattribute haben Werte, die einfach Listen möglicher Werte darstellen. Jeder dieser Werte muß ein gültiges Namens-Token (NMTOKEN) darstellen. Hier ein Beispiel:

```
<!ATTLIST malen
      COLOR (RED ¦ YELLOW ¦ GREEN) "RED">
```

Geht einer Liste möglicher Werte das Schlüsselwort NOTATION voraus (das Sie in Kapitel 8 kennenlernen), müssen die als mögliche Werte aufgelisteten Notationen bereits deklariert worden sein:

```
<!ATTLIST bild
      typ NOTATION (GIF ¦ JPEG ¦PNG) "GIF">
```

Beim Vergleich eines Attributwerts mit den in der Attributdefinition als erlaubt aufgelisteten Werten berücksichtigt der XML-Interpreter die Groß-/Kleinschreibung nicht, außer für die Typen CDATA, IDREF oder IDREFS.

Genauer gesagt gibt es in XML kein optionales Attribut. Es gibt jedoch einen nur wenig bekannten Trick, die Aufzählungstypen so zu benutzen, daß das Verhalten eines optionalen Attributs nachgebildet werden kann. Betrachten Sie die folgende Attributlistendeklaration:

```
<!ATTLIST clever.element
clever.att SORT (A ¦B ¦ C ) "" >
```

Wenn Sie jetzt dieses Attribut nicht explizit deklarieren, sondern es einfach weglassen, wie hier gezeigt:

```
<clever.element> Dieses Element hat keinen deklarierten
Attributwert.</clever.element>
```

wird dem Element der Vorgabewert zugewiesen, der in diesem Fall leer ist.

Standardwerte für Attribute

Sie können am Ende der Attributspezifikation ein Schlüsselwort angeben, das dem XML-Interpreter mitteilt, was zu tun ist, wenn Sie in einem Start-Tag ein Attribut weglassen (oder vergessen).

Es gibt drei solcher Schlüsselwörter:

- `#REQUIRED` bedeutet, daß das Attribut zwingend erforderlich ist und angegeben werden muß. Falls es fehlt, wird das Dokument ungültig.

  Hier ein Beispiel für eine zwingend erforderliche Deklaration:

```
<!ATTLIST buch
         autor ID #REQUIRED>
```

Normalerweise werden Attribute vom Typ `ID` als zwingend erforderlich gekennzeichnet. Sie werden später erfahren, daß sie als zwingend erforderlich angegeben werden müssen, wenn das Dokument ausgewertet werden soll.

- `#IMPLIED` bedeutet, daß der XML-Interpreter der Applikation mitteilen muß, daß kein Wert angegeben wurde. Es bleibt der Applikation überlassen, das weitere Vorgehen festzulegen.

  Hier ein Beispiel für eine implizite Deklaration:

```
<!ATTLIST abschnitt
       nummer #IMPLIED>
```

Implizite Attributwerte werden häufig für Dinge wie Abschnitts- oder Listenelementnumerierungen verwendet, wo die Applikation den Wert selbst durch einfaches Zählen berechnen kann. Sie könnten diese impliziten Wert auch für Attributwerte verwenden, die ein Element von seinen Eltern erben soll.

- Wenn dem Vorgabewert das Schlüsselwort #FIXED vorausgeht, muß jeder Wert, der angegeben wird, mit dem Vorgabewert übereinstimmen, sonst ist das Dokument ungültig.

Hier einige Beispiele für Attributdeklarationen mit Standardwerten:

```
<!ATTLIST begriffsdef
         id ID #REQUIRED
         name CDATA #IMPLIED>
<!ATTLIST list
         typ (roman ¦ arabic ¦ Roman ¦ Arabic) "roman">
<!ATTLIST form
         method CDATA #FIXED "POST">
```

Wohlgeformte XML-Dokumente

Elemente, Attribute und Entities sind die drei wichtigsten Bausteine in XML-Dokumenten. Schon alleine mit Elementen können Sie richtige XML-Dokumente anlegen. Mit Hilfe aller drei Objektarten können Sie relativ komplexe XML-Dokumente erstellen und die Anforderungen von 90 % aller Anwendungen erfüllen, in denen Sie XML einsetzen.

Damit solche XML-Dokumente korrekt genutzt werden können – mit anderen Worten, damit ein XML-Interpreter sie verarbeiten kann –, müssen sie wohlgeformt sein. Gemäß dem XML-Standard ist ein Datenobjekt erst dann ein XML-Dokument, wenn es wohlgeformt ist. Sie haben bereits die meisten Regeln kennengelernt, die befolgt werden müssen, damit ein XML-Dokument wohlgeformt ist, aber hier wollen wir noch einmal einen Überblick zeigen. Ein Dokument, das Sie unter Verwendung von Elementen, Attributen und Entities erzeugen, ist wohlgeformt, wenn folgendes gilt:

- Es enthält ein oder mehrere Elemente.
- Es hat ein Element (das Dokument- oder Wurzelelement), das alle anderen Elemente beinhaltet.
- Seine Elemente (falls es mehrere Elemente enthält) sind korrekt ineinander verschachtelt (kein Element beginnt innerhalb eines Elements und endet in einem anderen).
- Die Namen in den zusammengehörigen Start- und Ende-Tags stimmen genau überein.
- Die Namen von Attributen erscheinen nicht mehrmals im selben Elemtent-Start-Tag.

- Die Werte seiner Attribute sind in einfache oder doppelte Anführungszeichen eingeschlossen.
- Die Werte seiner Attribute verweisen nicht auf externe Entities, weder direkt noch indirekt.
- Der Ersatztext für Entities, auf den in einem Attributwert verwiesen wird, enthält kein Kleinerzeichen, < (er darf den String `<` enthalten).
- Seine Entities werden deklariert, bevor sie benutzt werden.
- Keiner seiner Entity-Verweise enthält den Namen eines nicht geparsten Entity.
- Seine logische und physische Struktur sind korrekt verschachtelt.

Es gibt viele Möglichkeiten, zu prüfen, ob ein XML-Dokument wohlgeformt ist, und viele Werkzeuge helfen Ihnen dabei. Beispielsweise gibt es als Public Domain Perl-Skripts und einige praktische Werkzeuge (einige davon werden Sie in Kapitel 5 kennenlernen). Oder Sie könnten versuchen, Ihren XML-Code in Mozilla zu laden, der Entwicklungsversion von Netscape 5 (mehr darüber in den Kapiteln 13 und 18).

Zusammenfassung

Heute haben Sie wichtige Details zur Elementdeklaration sowie zu den Elementinhaltsmodellen kennengelernt. Außerdem haben Sie erfahren, wie man Elementattribute deklariert und verwendet. In Kombination mit dem, was Sie bereits über Verarbeitungsanweisungen, die interne DTD-Untermenge und interne Entities gelernt haben, besitzen Sie jetzt genügend Wissen, um XML bereits in ersten Applikationen einzusetzen.

Wenn Sie keine externen Objekte einbinden (Grafiken oder andere Verweise) oder Ihre XML-Dokumente nicht auswerten wollen, könnten Sie hier einfach aufhören zu lesen! Es gibt jedoch noch vieles zu erfahren. In Kapitel 5 erfahren Sie, wie man mit Hilfe eines XML-Parsers prüft, ob Ihr XML-Dokument wohlgeformt ist, und dann können wir den nächsten, den technischen Schritt ins Auge fassen: die Auswertung von XML-Dokumenten.

F&A

F Warum ist die Elementgruppierung eine sinnvolle Methode, Elementinhaltsmodelle zu deklarieren?

A *Sie kann eine wichtige Hilfe darstellen, um mehrdeutige Inhaltsmodelle zu vermeiden. Bei sinnvollem Einsatz kann sie helfen, DTDs einfacher und verständlicher zu machen.*

F Warum bedingen Inhaltsmodelle mit gemischtem Inhalt besondere Aufmerksamkeit?

A *Der XML-Interpreter muß unbedingt zwischen einem beabsichtigten Leerzeichen, das er als Zeichendaten behandelt, und versehentlichen Leerzeichen unterscheiden können. Die Reihenfolge, in der das Inhaltsmodell deklariert wird, und die Verwendung von Vorkommensangaben sind deshalb sehr spezifisch und müssen genau befolgt werden.*

F Gibt es eine Beschränkung für die Länge eines Element- oder eines Attributnamens?

A *Nein. Solange Sie die Beschränkungen befolgen, welche Zeichen erlaubt sind, können Namen beliebig lang sein. (In SGML müssen Sie spezielle Vorkehrungen treffen, damit lange Namen möglich sind.)*

F Welche Besonderheit haben ID-Attributwerte?

A *Ein* `ID`*-Attributwert muß in einem XML-Dokument eindeutig sein.*

F Welcher Unterschied besteht zwischen einem Namen und einem Namens-Token?

A *Namens-Token sind weniger eingeschränkt als Namen. Namen müssen mit einem Buchstaben, einem Unterstrich oder einem Doppelpunkt beginnen; Namens-Token dürfen mit jedem gültigen Namenszeichen beginnen.*

F Wie erkennen Sie, ob ein XML-Dokument wohlgeformt ist?

A *Sie können einen XML-Interpreter benutzen, aber auch eines der vielen kostenlosen Utilities, die in Perl, JavaScript, Java und C geschrieben sind, die ein XML-Dokument parsen und Fehler bei der Wohlgeformtheit erkennen können.*

Übungen

1. Ein heißes Diskussionsthema in SGML-Kreisen (und bald auch in XML-Kreisen) ist, ob ein Element genutzt werden soll, um Information zu beschreiben, oder ob die Information statt dessen in einem Attribut abgelegt werden soll. Nennen Sie je zwei Begründungen für jede der beiden Möglichkeiten.
2. Der folgende XML-Codeabschnitt ist einem einfachen Teilekatalog entnommen. Fügen Sie eine interne DTD-Untermenge hinzu.

```
<?xml version="1.0"?>
          <parts>
            <part>
              <name>Widget</name>
              <catalog.number>11037</catalog.number>
              <price.code>4</price.code>
              <quantity>d</quantity>
            </part>
            <part>
              <name>Bolt</name>
              <catalog.number>11497</catalog.number>
              <thread>w</thread>
              <price.code>1</price.code>
              <quantity>100</quantity>
            </part>
            <part>
              <name>Screw</name>
              <catalog.number>10020</catalog.number>
              <type>countersunk</type>
              <price.code>7</price.code>
              <quantity>c</quantity>
            </part>
          </parts>
```

3. Die folgenden Inhaltsmodelle sind mehrdeutig. Schreiben Sie sie so, daß das nicht mehr der Fall ist.

a. `(section?), section)`

b. `((eins, zwei) ¦ (eins, drei))`

Prüfung wohlgeformter Elemente

Woche 1

In den ersten Kapiteln dieser Woche haben Sie gelernt, wohlgeformtes XML zu schreiben. Aber wie können Sie sicherstellen, daß Ihre XML-Dateien wirklich wohlgeformt sind? Es gibt Möglichkeiten! Heute gibt es bereits Software, deren Hauptaufgabe die Syntaxprüfung Ihrer Dateien gemäß der Regeln für die Wohlgeformtheit nach der XML-Empfehlung ist. Diese Programme werden auch als nichtauswertende Parser bezeichnet.

Auswertende Parser werden in Kapitel 9 vorgestellt, nachdem Sie ausreichend viel über die Dokumenttypdefinitionen wissen.

Neben der Prüfung Ihrer XML-Dateien bieten XML-Prozessoren auch anderen Applikationen Zugriff auf die Struktur und den Inhalt Ihrer XML-Dateien.

In diesem Kapitel lernen Sie die folgenden Dinge:

- Wo Sie diese XML-Parser finden
- Wie Sie den expat-Parser verwenden
- Wie Sie den DXP-Parser von DataChannel verwenden
- Wie Sie Ihre XML-Dateien mit Hilfe von RUWF über das Web prüfen
- Wie Sie Ihre XML-Dateien mit Hilfe anderer Auswertungsdienste über das Web prüfen

Wo finden Sie Informationen über die verfügbaren Parser?

Listen verfügbarer Parser finden Sie unter den folgenden URLs:

- Robin Cover bietet die vollständigste Referenzseite zu SGML und XML. Wenn Sie mehr über XML oder SGML wissen wollen, sehen Sie einfach hier nach:

  `http://www.oasis-open.org/cover/`.

- Lars Marius Garshol verwaltet `http://www.stud.ifi.uio.no/~larsga/linker/XMLtools.html#C_Parsers`. Dort finden Sie einen sinnvollen Überblick über kostenlose XML-Werkzeuge mit vielen erklärenden Kommentaren und allen notwendigen Verweisen.

- James Tauber bietet unter `http://www.xmlsoftware.com/parsers/` den klarsten Überblick. Er listet die Parser nach der verwendeten Programmiersprache auf, nach den angebotenen Schnittstellen, ob sie auswertend sind oder nicht usw.

Prüfen der XML-Dateien mit Hilfe von expat

In diesem Abschnitt erfahren Sie, wie Sie mit dem XML-Parser-Programm expat Ihre XML-Dateien prüfen. Das expat-Programm wurde von James Clark geschrieben, dem technischen Leiter der W3C-XML-Arbeitsgruppe. Er ist auch Autor weiterer praktischer kostenloser Software für die SGML/XML-Gemeinde. Der Code liegt in plattformunabhängigem C vor.

Mozilla 5, der nächste Release des Browsers von Netscape, verwendet den expat-Parser. Weitere Informationen finden Sie unter `http://www.mozilla.org/rdf/doc/xml.html`.

Für Perl gibt es ein Erweiterungsmodul, XML::Parser, die Perl-Schnittstelle zu expat. Weitere Informationen finden Sie unter `http://www.netheaven.com/~coopercc/xmlparser/intro.html`.

Installation von expat

expat steht unter `ftp://ftp.jclark.com/pub/xml/expat.zip` zum Herunterladen zur Verfügung. Diese ZIP-Datei ist 138 Kbyte groß (Version 19981122). Entzippen Sie die Datei auf Ihrer Platte, dann können Sie damit arbeiten.

Verwendung von expat

Uns interessiert besonders die xmlwf-Anwendung. Die Programmdatei befindet sich im Unterverzeichnis `bin` der expat-Distribution. Die Anwendung nimmt als Argumente eine oder mehrere Dateien entgegen, die auf ihre Wohlgeformtheit geprüft werden.

Die typische Verwendung sieht wie folgt aus:

```
xmlwf datei
```

Beispiel:

```
xmlwf c:\xmldateien\parse1.xml
```

Fehlerprüfung einer Datei

Jetzt wollen wir eine Datei schrittweise durchlaufen und dabei prüfen, welche Fehler expat erkennt. Dazu verwenden wir die in Listing 5.1 gezeigte XML-Datei.

Listing 5.1: wfq.xml – Überprüfung auf Syntaxfehler

```
1: <?xml version="1.0">
2: <?protext objid="I5678" ?>
3: <helptopic>
4: <title keyword="printing,network;printing,shared printer">How
➥to use a shared network printer?</title>
5: <procedure>
6: <step><action>In <icon>Network Neighborhood</icon>, locate and
➥double-click the computer where the printer you want to use is
➥located. </action>
7: <tip targetgroup="beginners">To see which computers have
➥shared printers attached, click the <menu>View</menu> menu,
8: click <menu>Details</menu>, & look for printer names or
➥descriptions in the Comment column of the Network
➥Neighborhood window.</tip>
9: </step>
10: <step>
11: <action>&doubleclick; the printer icon in the window that
➥appears.</action>
12: </step>
13: <step>
14: <action>
15: To set up the printer, <xref linkend="id45">follow the
➥instructions</xref> on the screen.
16: </step>
17: </procedure>
18: <rule form=double>
19: <tip>
20: <P>After you have set up a network printer, you can use it as
➥if it were attached to your computer. For related topics, look up
➥&quot;printing&quot; in the Help Index.
21: </p>
22: </tip>
23: </helptopic>
```

Betrachten wir, was expat ergibt.


```
xmlwf wfq.xml
```

Die erste Fehlermeldung lautet:


```
wfq.xml:1:19: syntax error
```

Diese Meldung beinhaltet vier Informationen:

- Den Namen der Datei, in der der Fehler erkannt wurde
- Die Zeilennummer mit dem Fehler
- Die Position des Fehlers in dieser Zeile
- Eine Fehlerbeschreibung

In dieser Datei wurde in Zeile 1, Zeichen 19 ein Syntaxfehler erkannt, das ist vor dem Größer-Zeichen, >. Die XML-Deklaration muß mit ?> enden, was hier nicht der Fall war. Wir müssen es korrigieren:

```
1: <?xml version="1.0"?>
```

Führen Sie expat noch einmal aus, dann erhalten Sie eine weitere Fehlermeldung:


```
wfq.xml:8:29: not well-formed
```

Das Problem scheint durch das verwendete Ampersand-Zeichen verursacht zu werden. Die XML-Spezifikation besagt, daß »das Ampersand-Zeichen (&) in seiner literalen Form nur als Markup-Begrenzungszeichen auftreten darf .... Falls es an anderer Stelle benötigt wird, muß ein Escape dafür ausgeführt werden.«

Dieses Escape kann mit Hilfe des vordefinierten Entity `&` erfolgen:

```
8: click <menu>Details</menu>, &amp; look for printer names
➥or descriptions in the Comment column of the Network
➥Neighborhood window.</tip>
```

Die nächste Fehlermeldung lautet:


```
wfq.xml:11:18: undefined entity
```

Sie haben einen Entity-Verweis auf das Entity `doubleclick` verwendet, das noch nicht deklariert wurde.

Jetzt wollen wir betrachten, was die Spezifikation dazu sagt: »Ein Textobjekt ist ein wohlgeformtes XML-Dokument, ... wenn für jeden im Dokument auftretenden Entity-Verweis das entsprechende Entity in der Dokumenttypdeklaration deklariert wurde oder der Entity-Name einer der folgenden ist: `amp`, `lt`, `gt`, `apos`, `quot`.«

Sie müssen also eine Entity-Deklaration in Ihr Dokument einfügen, d.h. in die Dokumenttypdeklaration, die Sie noch nicht haben.

Wir beginnen mit dem Einfügen einer Dokumenttypdeklaration:

```
3: <!DOCTYPE helptopic []>
```

Jetzt fügen wir die Deklaration des Entity `doubleclick` ein:

```
3: <!DOCTYPE helptopic [
4: <!ENTITY doubleclick "Double-click">
5: ]>
```

Jetzt sieht die XML-Datei aus wie in Listing 5.2 gezeigt.

***Listing 5.2:** `wfq.xml` – ein in der internen DTD-Untermenge deklariertes Entity*

```
1: <?xml version="1.0" ?>
2: <?protext objid="I5678" ?>
3: <!DOCTYPE helptopic [
4: <!ENTITY doubleclick "Double-click">
5: ]>
6: <helptopic>
7: <title keyword="printing,network;printing,shared printer">How
➥to use a shared network printer?</title>
8: <procedure>
9: <step><action>In <icon>Network Neighborhood</icon>, locate and
➥double-click the computer where the printer you want to use is
➥located. </action>
10: <tip targetgroup="beginners">To see which computers have
➥shared printers attached, click the <menu>View</menu> menu,
11: click <menu>Details</menu>, &amp; look for printer names or
➥descriptions in the Comment column of the Network
➥Neighborhood window.</tip>
12: </step>
13: <step>
14: <action>&doubleclick; the printer icon in the window that
➥appears.</action>
15: </step>
16: <step>
17: <action>
18: To set up the printer, <xref linkend="id45">follow the
➥instructions</xref> on the screen.
19: </step>
20: </procedure>
21: <rule form=double>
22: <tip>
23: <P>After you have set up a network printer, you can use it as
➥if it were attached to your computer. For related topics, look up
➥&quot;printing&quot; in the Help Index.
24: </p>
25: </tip>
26: </helptopic>
```

Jetzt parsen wir die Datei noch einmal:


```
wfq.xml:19:2: mismatched tag
```

Sie treffen auf ein Ende-Tag für das `step`-Element. Betrachten Sie genauer, was innerhalb des `step`-Elements passiert: Ein Aktions-Element wurde erkannt, aber es wird am Ende von `step` nicht abgeschlossen.

Dazu sehen wir uns noch einmal die XML-Spezifikation an: »Für alle anderen Elemente (bis auf das Wurzelelement) ist das Ende-Tag im selben Element wie das Start-Tag enthalten. Einfach ausgedrückt, die Elemente, die durch Start- und Ende-Tags begrenzt sind, werden ineinander verschachtelt.«

In Zeile 19 muß also stehen:

```
19: </action></step>
```

Nach dieser Korrektur trifft der Parser auf den nächsten Fehler:


```
wfq.xml:21:11: not well-formed
```

Das Problem liegt in der folgenden Zeile:

```
21: <rule form="double">
```

Betrachten wir das Attribut `form`. Innerhalb der Anführungszeichen muß ein Attributwert stehen:

```
21: <rule form="double">
```

Die nächste Meldung von expat lautet:


```
wfq.xml:24:2: mismatched tag
```

In Zeile 24 befindet sich ein Ende-Tag für das Element `p`, aber es wurde kein Element `p` geöffnet. `P` ist geöffnet. Start- und Ende-Tag müssen gleich geschrieben werden. Ändern Sie das Start-Tag in `p`:

```
23: <p>After you have set up a network printer, you can use it as
➥if it were attached to your computer. For related topics, look up
➥&amp;quot;printing&amp;quot; in the Help Index.
24: </p>
```

Als nächste Parser-Meldung erhalten wir:


```
wfq.xml:26:2: mismatched tag
```

Das ist ein komplizierter Fehler. Sehen Sie in Zeile 26 einen Fehler? Nicht sofort. Hier wird das Wurzelelement geschlossen. Sie wissen, daß alle anderen Elemente innerhalb dieses Wurzelelements verschachtelt werden müssen. Ist das der Fall? Wurde jedes geöffnete Element geschlossen?

Nein. Das Element `rule` ist noch geöffnet. Betrachten wir jedoch dieses Element etwas genauer. Enthält das Element `rule` irgendwelchen Inhalt? Nein, es scheint sich um ein leeres Element zu handeln. Und gemäß der Spezifikation müssen leere Elemente eines der folgenden Formate haben:

- `<tag></tag>`
- `<tag/>`

Man braucht also folgendes:

```
21: <rule form="double"/>
```

Nach diesen Korrekturen sollte die Datei aussehen wie in Listing 5.3 gezeigt.

***Listing 5.3:** `wf.xml` – die korrigierte XML-Datei*

```
1: <?xml version="1.0" ?>
2: <?protext objid="I5678" ?>
3: <!DOCTYPE helptopic [
4: <!ENTITY doubleclick "Double-click">
5: ]>
6: <helptopic>
7: <title keyword="printing,network;printing,shared printer">How
➥to use a shared network printer?</title>
8: <procedure>
9: <step><action>In <icon>Network Neighborhood</icon>, locate and
➥double-click the computer where the printer you want to use is
➥located. </action>
10: <tip targetgroup="beginners">To see which computers have
➥shared printers attached, click the <menu>View</menu> menu,
11: click <menu>Details</menu>, &amp; look for printer names or
➥descriptions in the Comment column of the Network
➥Neighborhood window.</tip>
12: </step>
13: <step>
14: <action>&doubleclick; the printer icon in the window that
➥appears.</action>
15: </step>
16: <step>
17: <action>
18: To set up the printer, <xref linkend="id45">follow the
➥instructions</xref> on the screen.
19: </action></step>
20: </procedure>
21: <rule form="double"/>
22: <tip>
23: <p>After you have set up a network printer, you can use it as
➥if it were attached to your computer. For related topics, look up
➥&quot;printing&quot; in the Help Index.
24: </p>
25: </tip>
26: </helptopic>
```

Jetzt parsen wir noch einmal. Es wird keine Fehlermeldung mehr angezeigt. Die Datei scheint fehlerfrei zu sein.

Überprüfen der XML-Dateien mit DXP

DXP steht für DataChannel XML Parser und basiert auf NXP (Norbert Mikula's XML Parser), einem der ersten XML-Parser. Er wurde in Java geschrieben und ist mittlerweile auch aus Geschwindigkeitsgründen in anderen Sprachen verfügbar.

Java ist die gebräuchlichste Sprache für die XML-Entwicklung. Das ist auch leicht zu verstehen. XML ist ein vereinfachtes und für das Web optimiertes SGML. Eines der Entwurfsziele von SGML war die Unabhängigkeit von Hardware, Betriebssystem, Software usw. Und genau dafür sorgt Java: Es ermöglicht Ihnen, Ihre Programme auf verschiedenen Maschinen unter verschiedenen Betriebssystemen und sogar in Web-Browsern auszuführen. Ein perfektes Paar.

Deshalb sind viele XML-Parser in Java programmiert

Sie können diesen Parser nicht nur zur Prüfung der Wohlgeformtheit einsetzen, sondern auch zur Auswertung, was in Kapitel 9 noch genauer besprochen wird.

Installation von DXP

Sie können DXP von `http://www.DataChannel.com/xml.resources/dxp.shtml#download` herunterladen. Die Datei, die Sie dabei herunterladen, ist `DXP.zip` mit 547 Kbyte. Neben den benötigten Java-Klassen finden Sie dort auch eine umfassende Dokumentation und Beispiele.

So installieren Sie DXP:

1. Entzippen Sie die Datei `DXP.zip`.
2. Stellen Sie sicher, daß die Java Virtual Machine Version 1.x läuft. Sie können den Java Development Kit oder JDK von SUN benutzen (`http://java.sun.com/products/jdk/1.1/`), die Java Runtime Environment oder JRE von Sun (`http://java.sun.com/products/jdk/1.1/jre/index.html`) oder den SDK für Java von Microsoft (`http://www.microsoft.com/java/`).
3. Stellen Sie sicher, daß Ihre virtuelle Java-Maschine die DXP-Klassen findet, indem Sie eine der folgenden Maßnahmen treffen:

   Fügen Sie Ihrer Umgebungsvariablen `classpath` das Paketverzeichnis des DXP-Parsers hinzu, `c:\DataChannel\dxp\classes`, oder

   setzen Sie den Parameter `-cp` für JRE in der Befehlszeile auf den Pfad: `jre -cp c:\DataChannel\dxp\classes`.

Verwendung von DXP

Geben Sie in die Befehlszeile (DOS-Eingabeaufforderung) das folgende ein:

```
jre -cp .;c:\DataChannel\dxp\classes dxpcl -s c:\xmlex\wfq.xml
```

Dabei gilt:

- `-cp` setzt den Klassenpfad (wo die verwendeten Klassen gefunden werden). In diesem Fall werden zwei Pfade angegeben: der erste, (.), der sich auf das aktuelle Arbeitsverzeichnis bezieht, und der zweite (`c:\DataChannel\dxp\classes`), durch ein Semikolon (;) voneinander getrennt.

Auf Unix-Maschinen werden Pfade durch einen Doppelpunkt voneinander getrennt.

- `dxpcl` ist der Name des Java-Programms (Klasse).
- `-s` steht für Silent-Modus.

Hier sind wir hauptsächlich an den Fehlern in dieser Datei interessiert. Bei der Aktivierung des Silent-Modus werden als Ausgabe nur die Fehlermeldungen erzeugt.

- `c:\xmlex\wfq.xml` ist die Datei, die geprüft werden soll.

Schrittweises Prüfen einer Datei

Jetzt wollen wir dieselbe Datei wieder schrittweise durchlaufen und die von DXP erkannten Fehler betrachten.

Die Prüfung der Datei `wfq.xml` ergibt den folgenden Fehler:

```
FATAL ERROR: encountered ">". Was expecting one of: <S> , "?"
Location: file:/c:/xmlex/wfq.xml:1:20
```

Für jede Fehlermeldung erhalten Sie:

- die Art des Fehlers und eine Erklärung sowie
- in einer zweiten Zeile die Position des Fehlers (in der Datei `wfq.xml` in Zeile 1, Zeichen 20).

Die XML-Spezifikation unterscheidet zwischen Fehlern und fatalen Fehlern. Verletzungen der Wohlgeformtheitsregeln sind fatale Fehler. Ein XML-Prozessor muß solche Fehler erkennen und darf danach die normale Verarbeitung nicht fortsetzen.

Am Ende finden Sie einen Überblick mit den Fehlernummern, fatalen Fehlern und Warnungen.

Sie wissen bereits, wo in dieser XML-Datei die Probleme liegen. Der erste Fehler war das fehlende Fragezeichen (?) am Ende der XML-Deklaration. Das können wir ändern und führen den Parser noch einmal aus:

```
FATAL ERROR: after the "&" there is no "entity-name" and ";"
➥following in the entity reference
Location: file:/c:/xmlex/wfq.xml:7:146

FATAL ERROR: reference to undefined entity "doubleclick"
Location: file:/c:/xmlex/wfq.xml:10:21

FATAL ERROR: name of the start-tag "action" doesn't match the
➥name of the end-tag "step"

Location: file:/c:/xmlex/wfq.xml:15:7

Found errors/warnings: 3 fatal error(s), 0 error(s) and 0 warning(s)
```

Jetzt machen wir das Ganze schneller. Es gibt noch drei Fehler, die wir schon kennen:

- Das Ampersand, &, für das kein Escape ausgeführt wurde
- Das nicht definierte Entity `doubleclick`
- Das fehlende Ende-Tag für das Element `action`

Korrigieren Sie und führen Sie den Parser noch einmal aus:

```
FATAL ERROR: encountered "double". Was expecting one of: "\"" , "\'"
Location: file:/c:/xmlex/wfq.xml:20:12

Found errors/warnings: 1 fatal error(s), 0 error(s) and 0 warning(s)
```

Attributwerte müssen in Anführungszeichen stehen. Nach dieser Korrektur erhalten Sie folgendes:

```
FATAL ERROR: name of the start-tag "P" doesn't match the name of
➥the end-tag "p"
Location: file:/c:/xmlex/wfq.xml:23:4

Found errors/warnings: 1 fatal error(s), 0 error(s) and 0 warning(s)
```

Nachdem Sie diesen Fehler korrigiert haben, parsen Sie noch einmal:

```
FATAL ERROR: name of the start-tag "rule" doesn't match the name
➥of the end-tag "helptopic"
Location: file:/c:/xmlex/wfq.xml:25:12

Found errors/warnings: 1 fatal error(s), 0 error(s) and 0 warning(s)
```

Machen Sie das `rule`-Element leer. Nach dieser Korrektur haben Sie eine korrekte Datei. Der Parser erzeugt keine weiteren Fehlermeldungen.

Prüfen der Dateien über das Web mit RUWF

RUWF, der XML-Syntaxprüfer, ist ein Dienst, der auf der Web-Site von XML.COM bereitgestellt wird, einer Partnerschaft zwischen Seybold Publications und O'Reilly, mit der technischen Unterstützung von Tim Bray und einer Präsentation von Xavier McLipps. Sie erreichen diesen Dienst unter `http://www.xml.com/xml/pub/tools/ruwf/check.html`.

Dieser Dienst wurde unter Verwendung von Lark eingerichtet, dem Parser von Tim Bray.

Die Verwendung von RUWF

1. Plazieren Sie Ihre XML-Datei auf einem Web-Server, so daß sie durch eine HTTP-Adresse angesprochen werden kann (beispielsweise `http://www.protext.be/wfq.xml`).
2. Gehen Sie zu `http://www.xml.com/xml/pub/tools/ruwf/check.html`.
3. Schreiben Sie Ihre URL in das Eingabefeld der RUWF-Seite, und übergeben Sie die Seite.

Wenn Sie `wfq.xml` dieser URL zuordnen, wird die in Abbildung 5.1 gezeigte HTML-Seite zurückgegeben.

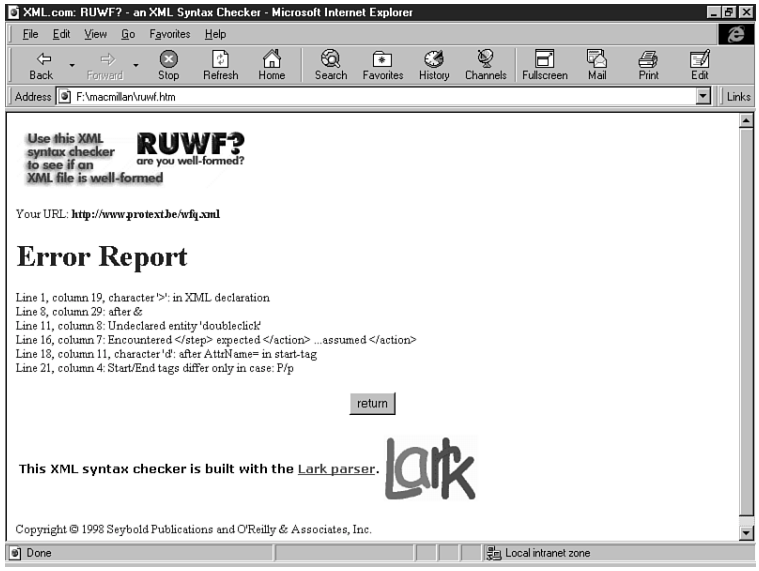


Abbildung 5.1: Das Ergebnis der Syntaxprüfung durch RUWF.

Prüfung Ihrer Dateien über das Web mit Hilfe anderer Online-Auswertungsdienste

Hier finden Sie noch einige andere Online-Auswertungsdienste:

- XML-Wohlgeformtheitsprüfer von Richard Tobin unter `http://www.cogsci.ed.ac.uk/~richard/xml-check.html`.
- XML Syntax Checker von Frontier unter `http://www.scripting.com/frontier5/xml/code/syntaxChecker.html`. Sie können den eingebauten Parser von Frontier 5.1.3 verwenden oder blox, der auf expat basiert.
- Beim Koala XML Validation Service unter `http://koala.inria.fr:8080/XML/`.
- Techno 2000 Project XML Validation Service unter `http:// xml.t2000.co.kr/xmlval/`.
- WebTech's Validation Service unter `http://valsvc.webtechs.com/`.

Der XML well-formdness Checker

Unter `http://www.cogsci.ed.ac.uk/~richard/xml-check.html` finden Sie die in Abbildung 5.2 gezeigte Seite.

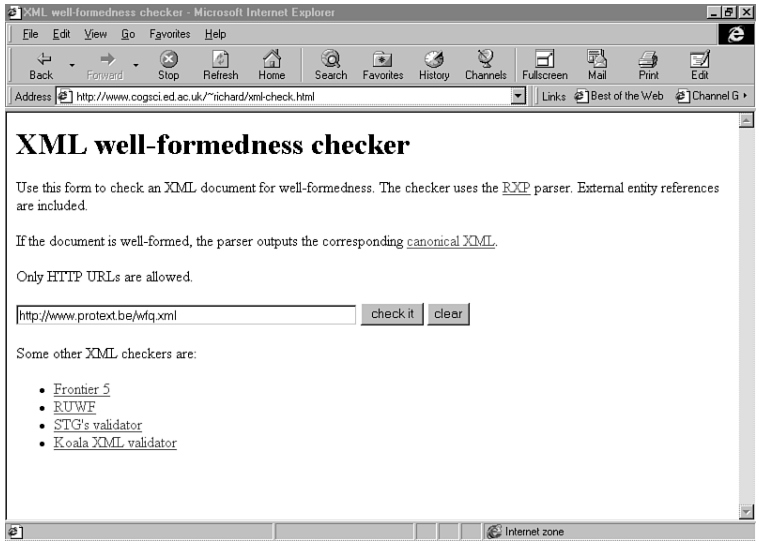


Abbildung 5.2: Die Übergabeseite des XML well-formedness Checker.

Nachdem Sie die URL der zu prüfenden Seite übergeben haben, erhalten Sie die in Abbildung 5.3 gezeigte Ergebnisseite.

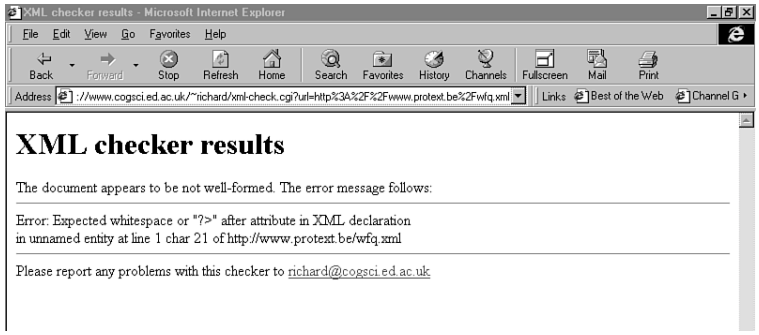


Abbildung 5.3: Die Ergebnisseite des XML well-formedness Checker.

Der XML Syntax Checker von Frontier

Abbildung 5.4 zeigt die Übergabeseite, die Sie unter `http://www.scripting.com/frontier5/xml/code/syntaxChecker.html` finden.

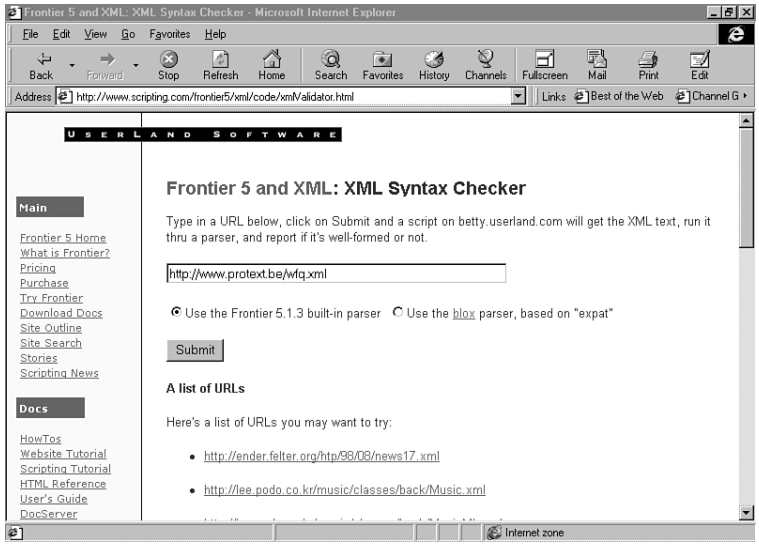


Abbildung 5.4: Die Übergabeseite des XML Syntax Checker von Frontier.

Abhängig von dem ausgewählten Parser erhalten Sie das in Abbildung 5.5 oder das in Abbildung 5.6 gezeigte Ergebnis.

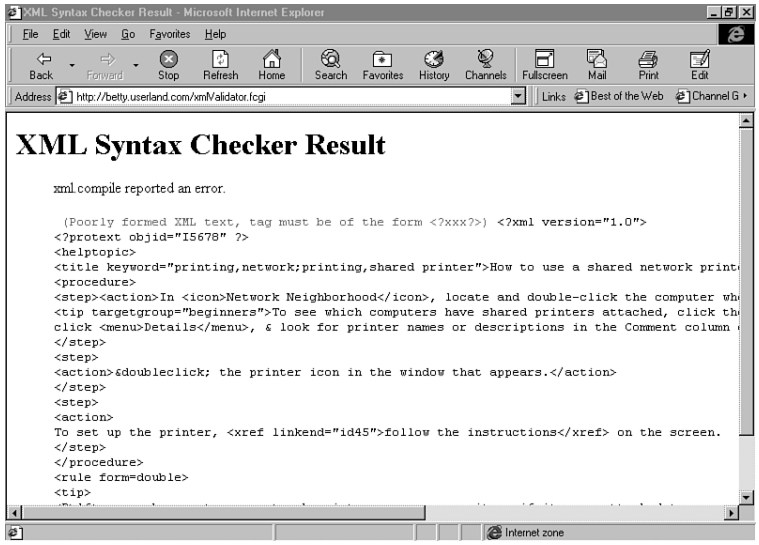


Abbildung 5.5: Die Ergebnisse des eingebauten Parsers von Frontier.

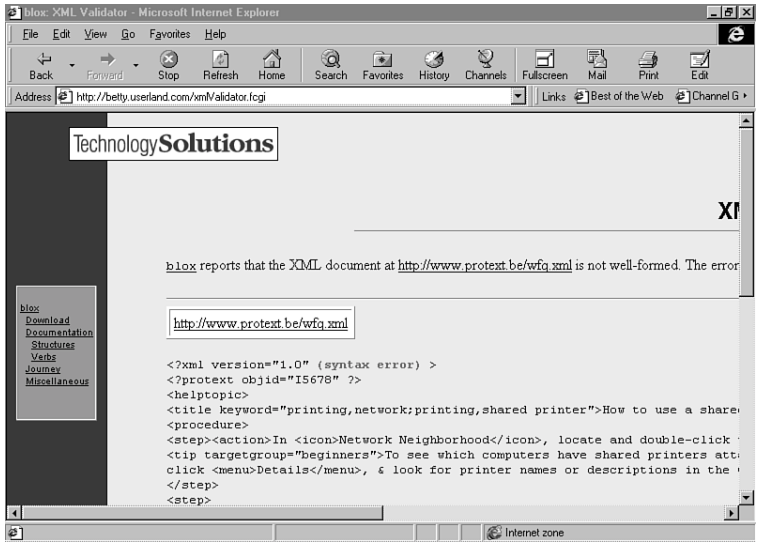


Abbildung 5.6: Die Ergebnisse von Blox, wo expat benutzt wird.

Zusammenfassung

In einer XML-Umgebung müssen Sie sicherstellen, daß Ihre Dokumente gemäß der XML-Empfehlungen wohlgeformt sind. Nichtauswertende Parser helfen Ihnen, diese Forderung zu erfüllen.

Viele dieser Werkzeuge stehen kostenlos im Web zur Verfügung. Wir haben unsere Prüfungen mit den folgenden ausgeführt:

- expat, einem Parser von James Clark
- DXP, einem Parser von DataChannel
- RUWF mit Lark von Tim Bray
- Andere Online-Auswertungsdienste

Was Sie tun sollten

Prüfen Sie Ihre XML-Dateien unbedingt auf ihre Wohlgeformtheit.

Das ist sehr wichtig, denn wenn eine XML-Datei die vorgegebenen Regeln zur Wohlgeformtheit verletzt, enthält sie fatale Fehler, und die normale Verarbeitung muß unterbrochen werden.

F&A

F Warum sollte ich mich darum kümmern, daß meine XML-Dateien wohlgeformt sind?

A *XML-Prozessoren dienen als Hilfsanwendungen für andere Anwendungen und bieten diesen Zugriff auf den Inhalt und die Struktur von XML-Dateien. Wenn eine XML-Datei eine Verletzung der für die Wohlgeformtheit spezifizierten Regeln enthält, enthält sie fatale Fehler. Beim Vorliegen fataler Fehler muß die normale Verarbeitung unterbrochen werden.*

Hier besteht ein echter Unterschied zu HTML, wo die Browser versuchen, auch aus dem Unsinn, der ihnen manchmal übergeben wird, noch etwas Sinnvolles abzuleiten. Aus diesem Grund werden etwa 75 Prozent des Codes im Internet Explorer und Netscape Navigator nur dafür bereitgestellt, fehlerhaftes HTML zu verarbeiten.

F Ich komme mit den zurückgegebenen Fehlermeldungen nicht zurecht. Was kann ich tun?

A *Wenn Sie die Meldung nicht verstehen, verwenden Sie die kommentierte XML-Spezifikation unter* `http://www.xml.com/axml/testaxml.htm`*. Diese Spezifikation ist ausführlich erklärt und kann Ihnen helfen, den relevanten Abschnitt zu finden.*

F Welche anderen Parser für die Prüfung der Wohlgeformtheit gibt es?

A *Andere Beispiele für nichtauswertende Parser sind Ælfred von David Megginson (*`http://www.microstar.com/XML/index.html`*) und der Microsoft XML Parser in C++, der im Internet Explorer 4.0 enthalten ist.*

Einen vollständigen Überblick finden Sie unter den Links am Ende dieses Kapitels.

F Sind HTML-Dateien wohlgeformt?

A *Nicht unbedingt, aber sie können wohlgeformt sein.*

Die offensichtlichste Verletzung der Wohlgeformtheit in HTML-Dateien sind:

Nicht alle Start- und Ende-Tags werden explizit angegeben.

Die Syntax der leeren Elemente.

Beachten Sie, daß in XML leere Elemente in einer von zwei möglichen Codierungsarten angegeben werden müssen:

```
<name></name>
```

```
<name/>
```

Übungen

1. Sehen Sie sich die folgende XML-Datei an:

Listing 5.4: *function.xml* ***– XML-Datei, die auf Wohlgeformtheit geprüft werden soll***

```
1: <?xml version="1.0"?>
2: <!DOCTYPE functiondescription [
3: <!NOTATION GIF SYSTEM "" >
4: <!ENTITY idhelp782 SYSTEM "idhelp782.txt">
5: <!ENTITY idhelp785 SYSTEM "idhelp785.txt">
6: <!ENTITY idhelp645 SYSTEM "idhelp645.txt">
7: <!ENTITY buttonleft SYSTEM "http://www.protext.be/button.gif"
➥NDATA GIF>
8: ]>
9: <function>
10: <title>ctime</title>
11: <funcsynopsis>
12: <funcdef>
13: <function>ctime</function>
14: </funcdef>
15: <paramdef>
16: <parameter>time</parameter>
17: <parameter role="opt">gmt</parameter>
18: </paramdef>
19: </funcsynopsis>
20: <para>This function converts the value
21: <parameter>time</parameter>, as returned by <function>time()
➥</function>
22: or <function>file_mtime()</function>, into a string of the
➥form produced by <function>
23: time_date()</function>. If the optional argument <parameter>
➥gmt</parameter>
24: is specified and non-zero, the time is returned in <parameter>
➥gmt</parameter>.
25: Otherwise, the time is given in the local time zone.</para>
26: <para><emphasis role="strong" targetgroup="beginners role=
➥"strong">Related topics</emphasis></para>
27: <para>&doubleclick; the <mousebutton>
28: LEFT &buttonleft;</mousebutton> mouse button on a topic:
➥</para>
29: <itemizedlist>
30: <listitem><para><ulink url="&idhelp782;"><function>
➥file_mtime()</function>
```

```
31: built-in function</ulink></para>
32: </listitem>
33: <listitem><para><ulink url="&idhelp785;"><function>time()
➥</function> built-in
34: function</ulink></para>
35: </listitem>
36: </itemizedlist>
37:</function>
```

Gehen Sie wie folgt vor:

- Finden Sie die Fehler und sagen Sie voraus, welche Fehlermeldungen ein Parser Ihrer Wahl erzeugen wird.
- Prüfen Sie die Datei mit diesem Parser.
- Korrigieren Sie alle erkannten Fehler.

2. Verwenden Sie eine XML-Datei, die Sie in einem der vorangegangenen Kapitel erzeugt haben, um die folgenden Dinge zu erledigen:

- Erzeugen Sie Fehler.

Verwenden Sie die W3C-Empfehlungen als Referenz.

- Parsen Sie die Datei mit unterschiedlichen Parsern.
- Betrachten Sie die unterschiedlichen Ergebnisse unterschiedlicher Parser.

Gültige Dokumente anlegen

Woche 1

Bisher haben Sie die Tags, Elemente und Attribute von XML sowie einige einfache Werkzeuge zum Prüfen der Wohlgeformtheit von XML-Dokumenten kennengelernt. Dieses Kapitel geht einen Schritt weiter und zeigt Ihnen die offensichtlichste Funktionalität von XML, die es als enger verwandt mit SGML als mit HTML auszeichnet. Heute lernen Sie die folgenden Dinge:

- Dokumenttypdefinition (DTD)
- Inhalt und Aufgabe der internen Untermenge
- Inhalt und Aufgabe der externen Untermenge
- Grundlagen der DTD-Entwicklung

Dieses Kapitel kann keinen vollständigen Überblick über die Entwicklung von DTDs bieten, aber es kann Ihnen die Grundlagen verschaffen. In Kapitel 7 betrachten wir die komplexeren Aspekte der DTD-Entwicklung und beschreiben einige Ansätze für die Informationsmodellierung bei der DTD-Entwicklung.

XML und strukturierte Information

Wenn Sie nur ein »korrektes Tagging« brauchen – wobei der gesamte Text sauber in Elemente verpackt wird –, ist es möglicherweise gar nicht erforderlich, XML-Dokumente zu erzeugen. Sie erreichen womöglich dasselbe mit Hilfe der Standardelemente von HTML und durch das Anlegen von Klassen, die die Arbeit für Sie erledigen. Der in Listing 6.1 gezeigte HTML-Code beispielsweise kann fast genausoviel wie der äquivalente XML-Code.

Listing 6.1: HTML zur Nachbildung des XML-Verhaltens

```
1: <HTML>
2:   <HEAD>
3:     <TITLE>HTML Mimicking XML</TITLE>
4:   </HEAD>
5:   <BODY>
6:     <DIV ID="1" CLASS="CHAPTER">
7:      <H1 CLASS="HEAD1">Chapter 1</H1>
8:          <P CLASS="PARA">This actually comes quite close
9:             to being acceptable as pseudo-XML.</P>
10:         <DIV ID="1.1" CLASS="SECTION">
11:           <H2 CLASS="HEAD2">Section 1.1</H2>
12:              <P CLASS="PARA">We can even bring in some
13:                 kind of pseudo-structure by using
14:                 attributes. </P>
```

```
15:           </DIV>
16:    </BODY>
17: </HTML>
```

Listing 6.1 zeigt ein Beispiel dafür, wie HTML-Code fast so umfassend wie XML-Markup sein kann. Durch die Verwendung der `DIV`-Elemente von HTML (Zeilen 6 und 10) und die Anwendung der Attribute `ID` und `CLASS` auf die benötigten Elemente können Sie fast soviel Information in das Markup einbinden, wie mit den benutzerdefinierten XML-Elementen möglich ist.

Vergleichen Sie jetzt den in Listing 6.1 gezeigten HTML-Code mit dem fast äquivalenten XML-Code, den Sie in Listing 6.2 sehen.

Listing 6.2: XML verwendet Elemente, um die HTML-Attribute nachzubilden

```
1: <?xml version="1.0"?>
2:   <DOCUMENT>
3:     <TITLE>XML Being What it is</TITLE>
4:   <CHAPTER>
5:      <HEAD>Chapter 1</HEAD>
6:          <PARA>This is quite
7:             acceptable XML.</PARA>
8:         <SECTION>
9:           <HEAD>Section 1.1</HEAD>
10:              <PARA>The structure lies in the
11:                 elements themselves. </PARA>
12:           </SECTION>
13:    </CHAPTER>
14: </DOCUMENT>
15: </DIV>
16: </DIV>
17: </BODY>
18: </HTML>
```

In Listing 6.2 verwendet das XML-Markup Elemente, um das zu erreichen, wofür HTML gezwungenerweise Attribute verwendet, weil es dort nicht die Freiheit gibt, benutzerdefinierte Elemente einzuführen (oder zumindest nicht offiziell). In dieser reduzierten Dimension (mit einer künstlichen Komplexitätsebene) könnten Sie HTML genausogut wie XML verwenden.

Was also gewinnen Sie durch die Verwendung von XML anstelle von HTML? Der offensichtlichste Unterschied ist, daß Sie in HTML sehr darauf beschränkt sind, welche Attribute Sie einem Element zuordnen können (meistenteils ist das nicht mehr als `ID` und `CLASS`), und nicht alle Elemente können diese Attribute erhalten. XML dagegen erlaubt Ihnen nicht nur die Verwendung beliebig vieler Attribute, sondern auch beliebiger Namen für diese Attribute (solange sie den Namensregeln entsprechen). Diese Attribute ermöglichen Ihnen, umfassende Informationen für die Beschreibung eines Elements einzufügen, wie in der in Listing 6.3 gezeigten Deklaration am Beispiel eines Grafikelements gezeigt ist.

Die Grafikattribute, die Sie in Listing 6.3 sehen, leitet sich von einer SGML-DTD des amerikanischen Verteidigungsministeriums ab und ist viel komplexer als alles, was Ihnen sonst je begegnen wird. Und es ist mit Sicherheit sehr viel komplizierter als alles, was Sie je selbst entwickeln werden.

Listing 6.3: Eine typische Deklaration für ein Grafikelement

```
1:<!ELEMENT graphic  EMPTY >
2:<!ATTLIST graphic boardno     ENTITY         #REQUIRED
3:                  graphsty    NMTOKEN        #IMPLIED
4:                  llcordra    NMTOKEN        #IMPLIED
5:                  rucordra    NMTOKEN        #IMPLIED
6:                  reprowid    NMTOKEN        #IMPLIED
7:                  reprodep    NMTOKEN        #IMPLIED
8:                  hscale      NMTOKEN        #IMPLIED
9:                  vscale      NMTOKEN        #IMPLIED
10:                 scalefit    %yesorno;      #IMPLIED
11:                 hplace      (left ¦ right ¦
12:                             center ¦ none) #IMPLIED
13:                 vplace      (top ¦ middle ¦
14:                             bottom ¦ none)  #IMPLIED
15:                 coordst     NMTOKEN         #IMPLIED
16:                 coordend    NMTOKEN         #IMPLIED
17:                 rotation    NMTOKEN         #IMPLIED>
```

Die Attribute in Listing 6.3 beschreiben alle physischen und Positionseigenschaften für das Bild. Diese Information ist relevant für die Verarbeitung des Elements, stellt aber die Information auf andere Weise bereit als der Text in dem XML-Dokument. Diese Meta-Information wird in Form von Attributen bereitgestellt und nicht als Teil normalen Elementinhalts des Dokuments.

Wie Sie jedoch bereits gesehen haben, ist HTML zwar nicht in der Lage, neue Elemente zu definieren, aber es ist dort durchaus erlaubt, beliebig viele Attribute zu verwenden.

Und was ist mit der Struktur? XML ist von Natur aus strukturiert; HTML hat wenig Struktur. HTML hat wie jede andere SGML-Applikation eine DTD ..., aber wenn Sie nicht gerade die technischen Details von HTML kennen, werden Sie sie nie bemerken, und abhängig davon, mit welchem Softwarepaket Sie Ihren HTML-Code erzeugen, müssen Sie auch kaum wissen, daß es sie überhaupt gibt. Die HTML-DTD ist eine nicht weitreichend spezifizierte DTD, für die nur sehr wenig zwingend erforderlich ist und die viel Raum bietet, die Reihenfolge der Elemente zu wählen, die Ihnen am besten zusagt. Das ist jedoch nur eine Frage der Vorgehensweise. Wenn Sie möchten, könnten Sie auch hier jede Menge Selbstdisziplin aufbringen und extrem wohlstrukturierte Ergebnisse erzielen, wie in Listing 6.4 gezeigt.

Listing 6.4: Strukturiertes HTML

```
<HTML>
 <HEAD>
    <TITLE>A Tale of Two Computers</TITLE>
 </HEAD>
  <BODY>
    <DIV CLASS="CHAPTER">
      <H1>CHAPTER ONE</H1>
      <P>It was a dark and stormy night.</P>
           <DIV CLASS="SECTION">
             <H2>At Home</H2>
             <P>The author huddled over his manuscript.</P>
           </DIV>
           <DIV CLASS="SECTION">
             <H2>At Work</H2>
             <P>The writer huddled over his draft.</P>
           </DIV>
    </DIV>
  </BODY>
 </HTML>
```

Wenn Sie wirklich wollen, können Sie HTML auch strukturiert verwenden. Es ist sogar möglich, einfach `P`-Tags und `FONT`-Attribute zu verwenden, um HTML bei seiner Anzeige in einem Web-Browser wie ein sehr strukturiertes Dokument aussehen zu lassen. Wenn Sie den HTML-Quellcode sehen, ist es aber fast unmöglich, den eigentlichen Text im Code zu lesen, geschweige denn, sich etwas darunter vorstellen zu können.

Wenn Sie sorgfältig vorgehen, können Sie in HTML-Code fast genausoviel Information codieren wie in XML. Das Problem dabei ist, daß Sie auf keine Hilfe von Ihren Werkzeugen hoffen dürfen. Sie erhalten sicherlich keinerlei Unterstützung von einem Web-Browser; diese wurden bewußt so angelegt, daß sie beliebige Tags einfach akzeptieren. Wenn die Web-Browser nichts mit den Tags anfangen können, ignorieren sie sie einfach. Die frühen Web-Browser-Entwickler machten sich zu viele Gedanken darüber, die Leute möglicherweise abzuschrecken, als daß sie strenge Regeln erzwungen hätten.

Aber es gibt einige Auswertungsprogramme für HTML-Code (sie nennen sich selbst sogar *Validators*, also Auswerter, aber das führt die Interpretation dieses Begriffs wirklich etwas zu weit), und im allgemeinen sind sie sehr gut. Diese Prüfprogramme stellen fest, ob Sie die HTML-Tags und die Attribute korrekt verwendet haben, aber sie können Ihnen nicht dabei helfen, die Struktur Ihres Dokuments zu überprüfen. Wenn Sie ein `H4`-Tag vor einem `H2`-Tag verwenden, dann ist das ganz einfach Ihr Problem. Wenn Sie `<P><FONT SIZE="7">` statt `<H1>` verwenden wollen, dann ist das alles in »der« HTML-DTD erlaubt (es sind mindestens 19 verschiedene offiziell anerkannte HTML-DTDs im Umlauf).

Warum braucht man überhaupt eine DTD?

Wir haben bereits über DTDs gesprochen und daß es in HTML auch eine gibt. Später im heutigen Kapitel werden Sie erfahren, daß XML Ihnen erlaubt, mehrere zu benutzen, aber das wirft die Frage auf, warum man überhaupt eine DTD braucht. Wurde XML nicht als DTD-loses SGML angepriesen? Richtig, wie Sie bereits erfahren haben – solange es wohlgeformt ist, braucht man keine DTD. Sie werden heute auch lernen, daß es möglich ist, eine DTD abzuleiten, indem man ein XML-Dokument einfach nur ansieht. Es gibt jedoch einige wichtige Einschränkungen für XML-Dokumente, die keine DTDs verwenden.

Wenn Sie ein XML-Dokument ohne eine DTD auswerten wollen, muß folgendes gelten:

- Alle Attributwerte im XML-Dokument müssen angegeben werden; es ist nicht möglich, Vorgabewerte dafür zu spezifizieren.
- Es gibt im XML-Dokument keine Verweise auf Entities (außer natürlich auf `amp`, `lt`, `gt`, `apos` und `quot`).
- Es darf keine Attribute mit Werten geben, die der Normalisierung unterliegen. (Beispielsweise, wenn sie Entity-Verweise enthalten. Wir werden in Kapitel 9 noch einmal auf die Entities zurückkommen.)

- In Elementen, deren Inhalt wiederum nur aus Elementen besteht, darf es keine Whitespaces (Leerzeichen, Tabulator oder andere Whitespace-Zeichen) zwischen dem Start-Tag des Container-Elements und dem Start-Tag des ersten enthaltenen Elements geben. Folgendes beispielsweise wäre nicht erlaubt:

```
<KAPITEL>     <ABSCHNITT>... </ABSCHNITT></KAPITEL>
```

Das ist ein komplizierter Sachverhalt, aber ohne die Hilfe einer DTD, die dem XML-Interpreter mitteilt, ob dieses Whitespace-Zeichen als Information (wie PCDATA oder ein geschütztes Whitespace-Zeichen) gewertet werden soll, kann er nicht entscheiden, ob er es löschen soll oder nicht.

DTDs und Auswertung

Die DTD beschreibt ein Strukturmodell für den Inhalt eines XML-Dokuments. Dieses Modell besagt, welche Elemente zwingend vorhanden sein müssen, welche optional sind, welche Attribute sie haben und wie sie in Beziehung zueinander strukturiert werden können. Während HTML nur eine DTD hat, erlaubt Ihnen XML, für jede Ihrer Anwendungen eine eigene DTD anzulegen. Damit haben Sie volle Kontrolle über die Überprüfung des Inhalts und der Struktur eines XML-Dokuments, das für diese Anwendung angelegt wird. Diese Prüfung wird auch als *Auswertung* bezeichnet.

Von allen Markup-Sprachen sind SGML und XML dahingehend fast einzigartig, daß ihre Dokumente wirklich ausgewertet werden können. Es gibt auch Softwarepakete, die HTML-Code »auswerten«, aber dabei handelt es sich um eine ganz andere Art von Auswertung. Die Auswertung von HTML-Code ist in der Regel nicht sehr viel mehr als eine Syntaxprüfung der Tags – wobei nach Schreibfehlern oder Auslassungen gesucht wird. Die echte Auswertung in SGML und XML geht viel weiter. Ein gültiges XML-Dokument ist nicht nur syntaktisch korrekt, sondern hat auch eine interne Struktur, die der in der DTD deklarierten Modellstruktur entspricht.

Abhängig davon, was Sie als DTD-Entwickler erreichen möchten, können Sie fast volle Kontrolle über die Struktur ausüben und eine strenge DTD anlegen. Wenn Sie XML-Dokumente auswerten, die unter Verwendung dieser strengen DTD angelegt wurden, können Sie die Bereitstellung bestimmter Elemente erzwingen und ebenfalls die Reihenfolge, in der sie angegeben werden. Sie können prüfen, ob bestimmte Attributwerte gesetzt werden, und bis zu einem bestimmten Maß auch sicherstellen, ob diese Werte den richtigen allgemeinen Typ besitzen.

Auf der anderen Seite können Sie auch fast alles optional machen und eine lockere DTD anlegen. Sie könnten sogar parallele Versionen derselben DTD erzeugen, eine die Ihnen erlaubt, Skizzenversionen des XML anzulegen, die noch nicht vollständig sein müssen, und eine weitere, die streng darauf achtet, daß alles vorhanden ist. Man

kann noch weiter gehen, wie Sie in Kapitel 9 noch erfahren werden, und Schalter in eine DTD einfügen, mit der die strenge Auswertung aktiviert bzw. deaktiviert werden kann.

Ihre Deklarationen in der DTD bestimmen, was bei der Auswertung des vollständigen XML-Dokuments erlaubt ist und was nicht. Der Autor des Dokuments kann gewarnt werden, wenn zwingend erforderliche Elemente fehlen, wie in Abbildung 6.1 gezeigt, oder wenn Elemente nicht an der richtigen Position stehen, wie in Abbildung 6.2 gezeigt.

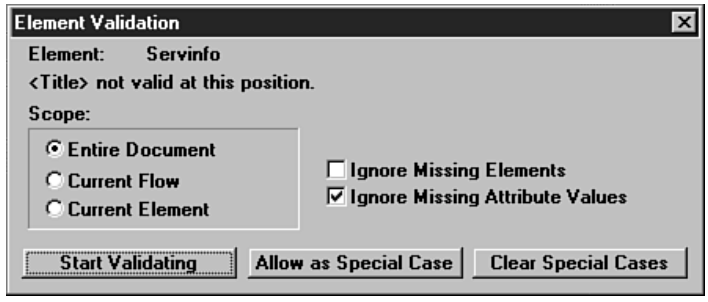


Abbildung 6.1: Warnung, daß ein Element fehlt.

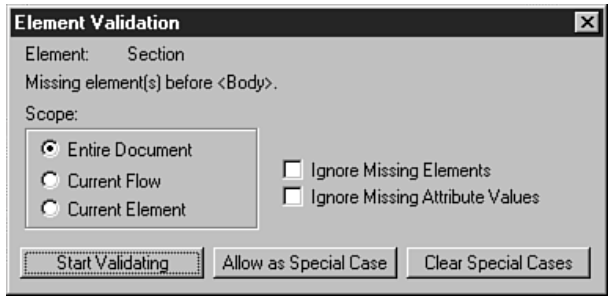


Abbildung 6.2: Warnung wegen einer fehlerhaften Struktur.

Dokumenttypdeklarationen

Sie wollen also eine DTD verwenden. Wie können Sie die DTD Ihrem XML-Dokument zuordnen? Sie verwenden eine Dokumenttypdeklaration. Verwirrend? Selbst in fest eingesessenen SGML-Kreisen herrscht große Verwirrung zwischen den sehr ähnlichen Begriffen der Dokumenttypdeklaration und Dokumenttypdefinition. Dabei handelt es sich jedoch um zwei völlig unterschiedliche Dinge, es ist also sinnvoll, den Unterschied zu kennen. Die Dokumenttypdefinition (DTD) ist eine XML- (und SGML-) Beschreibung des Inhaltsmodells für einen Dokumenttyp oder eine Dokumentklasse. Die Dokumenttypdeklaration ist eine Anweisung in einer XML- (oder SGML-)Datei,

die die DTD identifiziert, die zu dem Dokument gehört. Wenn eine externe DTD verwendet wird, gibt die Dokumenttypdeklaration an, wo das DTD-Entity (die Datei) gefunden werden kann.

In ihrer einfachsten Form sieht die Syntax für eine Dokumenttypdeklaration wie folgt aus:

```
<!DOCTYPE DTD.name [ interne.untermenge ]>
```

`DTD.name` ist der Name der DTD. Wenn Sie später in diesem Kapitel etwas über Gültigkeit lesen, werden Sie entdecken, daß der DTD-Name derselbe wie der des Wurzelelements für das Dokument sein sollte. Heißt also eine DTD für ein Dokument `buch`, dann muß auch das Wurzelelement in dem Dokument `buch` sein. Vergessen Sie nicht, daß XML die Groß-/Kleinschreibung berücksichtigt – wenn Sie die DTD als `BooK` bezeichnen, müssen Sie auch das Wurzelelement `BooK` verwenden.

`interne.untermenge` ist der Inhalt der internen DTD-Untermenge, dem Teil der DTD, die im XML-Dokument abgelegt ist. Wir werden die interne DTD-Untermenge kurz betrachten; sie enthält die Deklarationen für lokale Elemente, Attribute und Entities. Ohne die interne DTD-Untermenge wäre es kaum sinnvoll, überhaupt eine Dokumenttypdeklaration bereitzustellen.

Interne DTD-Untermenge

Damit ein XML-Dokument wohlgeformt ist, müssen alle externen Entities in der DTD deklariert sein. Wenn Sie Ihre Anwendung sorgfältig entwerfen, können Sie alle Deklarationen in der internen DTD-Untermenge anlegen. Befinden sich alle Deklarationen in der internen DTD-Untermenge, muß der XML-Interpreter keine externen Dokumente lesen und verarbeiten.

Die Existenz einer internen DTD-Untermenge hat keinen Einfluß auf den Status des XML-Dokuments als Standalone-Dokument. Das kann zunächst etwas verwirrend sein, weil Sie vielleicht annehmen, daß »standalone« bedeutet, daß das Dokument überhaupt keinen Verweis auf eine DTD braucht. Solange die Deklarationen in der internen DTD-Untermenge einigen grundlegenden Regeln gehorchen, ist das XML-Dokument ein Standalone-Dokument.

Wenn Sie ein XML-Dokument entwickeln, steht in der ersten Zeile die XML-Deklaration, die eine Standalone-Dokumentdeklaration beinhalten kann:

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

Die Anweisung `standalone="yes"` bedeutet, daß es keine externen Markup-Deklarationen für das Dokument-Entity gibt (Sie erinnern sich an unsere Beschreibung der physischen und logischen Strukturen in Kapitel 2). In dem XML-Dokument ist es immer noch möglich, auf externe Entities zu verweisen (Grafikdateien, eingebundenen Text usw.), vorausgesetzt, daß die Deklarationen der externen Entities innerhalb des Dokument-Entity enthalten sind (mit anderen Worten, innerhalb der internen DTD-Untermenge).

Standalone-XML-Dokumente

Sie brauchen nur eine Dokumenttypdeklaration und den Inhalt einer internen DTD-Untermenge, um die Struktur eines XML-Dokuments zu definieren. Auch ohne externe Unterstützung und ohne Verweise auf andere Dateien kann ein XML-Dokument mit einer internen DTD-Untermenge genug Informationen für die Entwicklung relativ komplexer Applikationen enthalten.

Mit Ihrem jetzigen Wissen über die Deklaration von Elementen und Attributen sollten Sie in der Lage sein, etwas zu entwickeln wie den in Listing 6.5 gezeigten einfachen Katalog.

Listing 6.5: Ein Standalone-XML-Dokument mit interner DTD-Untermenge

```
1:  <?XML version="1.0" standalone="yes"?>
2:  <!DOCTYPE CATALOG [
3:
4:  <!ELEMENT CATALOG (PRODUCT+)>
5:
6:  <!ELEMENT PRODUCT (SPECIFICATIONS+, PRICE+, NOTES?)>
7:  <!ATTLIST PRODUCT NAME CDATA #REQUIRED>
8:
9:  <!ELEMENT SPECIFICATIONS (#PCDATA)>
10: <!ATTLIST SPECIFICATIONS SIZE CDATA #REQUIRED
11:                          COLOR CDATA #REQUIRED>
12:
13: <!ELEMENT PRICE (#PCDATA)>
14: <!ATTLIST PRICE WHOLESALE NMTOKEN #REQUIRED
15:           RETAIL NMTOKEN #REQUIRED
16:           SALES.TAX NMTOKEN #IMPLIED>
17:
18: <!ELEMENT NOTES (#PCDATA)>
19: ]>
20:
```

```
21: <CATALOG>
22:   <PRODUCT NAME="T-shirt">
23:     <SPECIFICATION SIZE="XL" COLOR="WHITE"/>
24:     <PRICE WHOLESALE="9.95" RETAIL="19.95" SALES.TAX="2.56" />
25:     <NOTES>Dilbert</NOTES>
26:   </PRODUCT>
27:   <PRODUCT NAME="Shirt">
28:     <SPECIFICATION SIZE="38" COLOR="BLACK"/>
29:     <PRICE WHOLESALE="69.95" RETAIL="79.95" SALES.TAX="4.54">
30:     Euro</PRICE>
31:   </PRODUCT>
32: </CATALOG>
```

Das XML-Markup, das Sie in Listing 6.5 sehen, beginnt mit dem jetzt schon bekannten XML-Prolog, der angibt, daß der nachfolgende Code ein Dokument in der XML-Version 1.0 ist. Ich habe die Anweisung `STANDALONE` eingefügt, um den XML-Interpreter explizit darauf hinzuweisen, daß er nicht nach einer externen DTD suchen muß.

Zeile 2 deklariert den Dokumenttyp als `CATALOG`, und anschließend öffnen wir die interne DTD-Untermenge.

Den Hauptinhalt der DTD sehen Sie in den Zeilen 4 bis 18. Wir haben ein `CATALOG`-Element, das ein oder mehrere `PRODUCT`-Elemente enthält. Ein `PRODUCT`-Element (Zeile 6) enthält ein oder mehrere `SPECIFICATION`-Elemente, gefolgt von ein oder mehreren `PRICE`-Elementen und dann optionalen `NOTES`-Elementen.

Die `SPECIFICATION`- und `PRICE`-Elemente sind, wie Sie im Markup sehen (Zeilen 21 bis 31), leer, und ihre Information ist in Form von Attributen angegeben (Zeilen 23, 24, 28 und 29).

Bei der Auswertung kann das Markup geprüft werden, aber kaum, was sich zwischen diesem Markup befindet (der Inhalt). Wenn Sie möchten, haben Sie perfekt strukturierten Müll oder eben perfekt strukturierte Leerzeichen und Tabulatoren.

In diesem Fall wollte ich einfach kein Währungsattribut angeben, deshalb gehen wir davon aus, daß alle Preise (die Werte des `PRICE`-Attributs) als lokale Währung vorausgesetzt werden. Das ist vielleicht nicht immer so, aber ich habe immer noch die Möglichkeit, diese Information als Text innerhalb des `PRICE`-Elements anzugeben (Zeilen 29 und 30), ohne später wieder alles umzuändern.

Beachten Sie, daß ich Elemente für die Bereitstellung der wichtigsten Objekte und Eigenschaften dieser Objekte in der DTD verwendet habe; ein PRODUCT beinhaltet SPECIFICATION-Elemente. Die meisten Eigenschaftsdaten werden jedoch in der Praxis als die Attributwerte der Elemente SIZE, COLOR und Werte des PRICE-Attributs deklariert.

Die Entscheidung, wann Elemente und wann Attribute verwendet werden sollen, ist schwer. Wir werden später noch weiter auf dieses Problem eingehen. Ich habe die meisten Attributwerte als #REQUIRED deklariert, das bedeutet, ihre Angabe ist zwingend erforderlich. Es gibt jedoch eine Ausnahme: das Attribut SALES.TAX, das #IMPLIED ist. Wird sein Wert dem System nicht übergeben (genau gesagt, der Anwendung, die auf dem Computersystem läuft), ist dieses trotzdem in der Lage, ihn für mich zu berechnen.

Das XML-Dokument, das ich hier beschrieben habe, kann ausgewertet werden. Es enthält ausreichend viel DTD, die interne DTD-Untermenge, um zu ermöglichen, daß der Inhalt und die Struktur des XML-Dokuments geprüft werden. In dieser Form ist das XML-Dokument relativ gut portierbar, und es gibt nachvollziehbare Gründe, das Ganze so beizubehalten. Durch Hinzufügen von 14 Zeilen zum XML-Markup (viel weniger, wenn Sie die Zeilenumbrüche und die Leerzeichen weglassen, die das Ganze besser lesbar machen) haben wir das Dokument relativ selbstbeschreibend gemacht. Der Empfänger kann sogar zum Teil prüfen, ob alle Teile komplett sind. Die Tatsache, daß ein komplettes Produkt fehlt, könnte durch das Fehlen eines schließenden CATALOG-Tags erkannt werden (und es kann angenommen werden, daß die Datei bei der Übertragung gekürzt wurde). Sie könnten durch die Auswertung nicht erkennen, wie viele Produkte fehlen (obwohl Sie die DTD einfach modifizieren können, auch dynamisch, damit das geprüft werden kann).

Komplexe, externe DTDs anlegen

Sie haben bereits erfahren, daß Sie mit einem Standalone-XML-Dokument relativ viel erreichen können. Während Sie zwar alle Vorteile der Portierbarkeit genießen, wenn Sie die DTD innerhalb des XML-Dokuments ablegen, erreichen Sie auch nur die Spitze des Eisbergs, wenn Sie nicht den nächsten logischen Schritt tun und eine externe DTD-Untermenge verwenden.

Tatsache ist, daß der Name Dokument*typ*definition schon darauf hinweist, daß eine DTD zur Verwendung für mehrere Dokumente vorgesehen ist. Ohne die Verwendung einer externen DTD-Untermenge fehlen Ihnen viele XML-Funktionen und außerdem die Möglichkeit, die DTD als Art Schablone für beliebig viele XML-Dokumente zu verwenden.

Im XML-Kontext wird die externe DTD-Untermenge häufig auch als *externe* DTD bezeichnet, aber im SGML-Kontext spricht man meistens nur von *der* DTD. XML hat eine DTD, aber sie setzt sich aus beidem zusammen, der internen und der externen DTD-Untermenge.

In XML wird die interne vor der externen DTD-Untermenge gelesen und hat deshalb Priorität. Damit können Sie eine externe DTD-Untermenge verwenden, um globale Deklarationen vorzunehmen und sie dann individuell zu überschreiben, indem Sie andere Deklarationen in die interne DTD-Untermenge aufnehmen.

Sie haben bereits erfahren, wie man einem XML-Dokument eine interne DTD-Untermenge zuweist. Die Zuordnung einer externen DTD-Untermenge ist für ein XML-Dokument komplexer. Man benötigt einen Systembezeichner (Schlüsselwort `SYSTEM`) oder einen öffentlichen Bezeichner (Schlüsselwort `PUBLIC`) und einen Systembezeichner:

```
<!DOCTYPE name öffentlicher.bezeichner system.bezeichner [ interne.untermenge ]>
```

Sowohl der Systembezeichner als auch der öffentliche Bezeichner sind Entity-Bezeichner. Wir werden in Kapitel 8 noch einmal genauer darauf eingehen. Hier werden wir nur so viel darüber sagen, wie Sie brauchen, um sie für den Verweis auf externe DTDs nutzen zu können.

Systembezeichner

Ein Systembezeichner ist ein *URI* (*Universal Resource Identifier*), der genutzt werden kann, um die DTD zu ermitteln. Wenn Sie je eine Webseite in einem Browser geöffnet oder eine Datei von einer FTP-Site heruntergeladen haben, dann kennen Sie bereits eine Form des URI, den URL (*Uniform Resource Locator*). URLs sind Spezialfälle von URIs, die für die Verwendung im Netzwerk (Internet) vorgesehen sind. Sie werden den technisch präziseren Namen URI häufiger finden als URL, aber im Normalfall können die beiden Begriffe austauschbar verwendet werden.

Ein Systembezeichner kann auf eine absolute Position verweisen, etwa wie folgt:

```
<!DOCTYPE book SYSTEM "/mount/usr/home/dtds/book.dtd">
<!DOCTYPE book SYSTEM "http://www.in.synopsys.com/~north/dtds/book.dtd">
```

Er kann aber auch ein Verweis auf eine relative Position sein:

```
<!DOCTYPE book SYSTEM "dtds/book.dtd">
<!DOCTYPE book SYSTEM "../../dtds/book.dtd">
```

Öffentliche Bezeichner

Ein öffentlicher Bezeichner ist der offiziell aufgezeichnete Bezeichner für eine DTD. Offensichtlich wäre es unmöglich, jede DTD zu registrieren (das war schon bei SGML nicht möglich). Statt dessen registriert die Person oder das Unternehmen, die die DTD erzeugen, sie selbst. Die ISO (*International Standards Organization*) mit ihrer Abteilung ISO 9070 ist für die Registrierung zuständig, aber die Autorität, Bezeichner und die entsprechenden Aufzeichnungen zu erzeugen, wurde an die GCA (American Graphic Communication Association) delegiert.

Ein öffentlicher Bezeichner hat die folgende Form:

```
reg.typ // eigentümer //DTD beschreibung // sprache
```

`reg.typ` ist ein Pluszeichen (+), falls der Eigentümer gemäß dem ISO-9070-Standard registriert ist. Normalerweise ist das jedoch nicht der Fall; dann ist `reg.typ` einfach ein Minuszeichen (-).

- `eigentümer` ist der Name des Eigentümers: Ihr Name oder der Name Ihrer Firma.
- `beschreibung` ist eine einfache Textbeschreibung. Sie können diese Beschreibung beliebig lang machen, aber am besten halten Sie sie so kurz und informativ wie möglich. Leerzeichen sind in der Beschreibung erlaubt; sie können also richtige Informationen hineinschreiben, wie beispielsweise »Einfache E-Mail-Nachricht«.
- `sprache` ist der zweistellige Sprachcode aus dem ISO-639-Standard.

Ein Beispiel für den öffentlichen Bezeichner für eine von mir entwickelte DTD könnte also wie folgt aussehen:

```
-//Simon North//DTD Simple Web Page//EN
```

Ein XML-Interpreter, der versucht, den Inhalt der DTD zu finden, könnte den öffentlichen Bezeichner verwenden, um eine Position ausfindig zu machen. Aus Gründen, die in Kapitel 9 noch genauer erklärt werden, ist das nicht immer möglich, und der öffentliche Bezeichner muß einem sogenannten *Systemliteral* folgen, wobei es sich einfach um einen URI handelt. Unter Verwendung des öffentlichen Bezeichners und des Systemliterals würde die Dokumenttypdeklaration dann so aussehen:

```
<!DOCTYPE home.page PUBLIC "-//Simon North//DTD Simple Web Page//EN" "home.dtd">
```

Beachten Sie, daß vor der Suche nach einer Übereinstimmung mit einem öffentlichen Bezeichner alle Zeichenketten mit Whitespaces zu einzelnen Leerzeichen normalisiert und daß führende und folgende Whitespaces entfernt werden.

Entwicklung der DTD

Es gibt viele Möglichkeiten, Informationsmodelle zu beschreiben, technisch als *Schemata* bezeichnet. Es gibt mehrere XML-Entwicklungsaktivitäten, die der Definition von Schemata für die Beschreibung von XML-Daten gewidmet sind. Ein solches Schema ist die XML DTD, die direkt von SGML geerbt wurde.

Die Aufgabe, eine DTD zu entwickeln, können Sie sich beliebig einfach oder schwer machen. Das ist alles davon abhängig, was Sie mit der Information vorhaben, die Sie mit der DTD modellieren wollen, und was Sie mit der Information machen, nachdem das Markup eingefügt wurde. Keinesfalls kann ich das Thema der DTD-Entwicklung hier auch nur annähernd vollständig behandeln; dazu bräuchte es ein eigenes Buch, dicker als das vorliegende. In Kapitel 7 werde ich versuchen, einige der wichtigsten Schritte aufzuzeigen, die Sie bei der Entwicklung Ihrer DTD befolgen sollten. Hier wollen wir nur einige der schnellen und einfachen Methoden betrachten.

Es gibt zwei Methoden, die einem sofort als mögliche Abkürzungen zum Anlegen einer DTD einfallen:

1. Abändern einer bereits existierenden SGML DTD
2. Anlegen der DTD aus dem XML-Dokument, entweder automatisch oder manuell

Abändern einer SGML-DTD

SGML und XML verwenden DTDs. SML ist eine Untermenge von SGML. Es scheint also durchaus logisch, eine SGML-DTD (davon gibt es Hunderte im Internet) zu nehmen und sie für die Verwendung mit dem XML-Dokument abzuändern. Leider ist das Leben nicht immer so einfach.

Es gibt viele Unterschiede zwischen einer SGML-DTD und einer XML-DTD, und anders als SGML- und XML-Dokumente sind die beiden noch weit davon entfernt, Kompatibilität zu bieten. Die Umwandlung einer XML-DTD in eine SGML-DTD könnte relativ einfach sein (und irgendwann trivial, nachdem SGML es geschafft hat, XML SGML-konform zu machen). Auf der anderen Seite kann es auch eine extrem schwierige Aufgabe sein, eine SGML-DTD in eine XML-DTD umzuwandeln.

Die Tatsache, daß so viele SGML-DTDs öffentlich zur Verfügung stehen, könnte Sie in Versuchung führen, zu probieren, eine SGML-DTD in XML umzuwandeln. Ich rate Ihnen davon ab, zumindest solange Sie nicht genügend über die XML-DTDs wissen, um die Unterschiede zu erkennen und zu verstehen. Es gibt viele Funktionen, die SGML in der DTD unterstützt, XML dagegen nicht (und einige dieser Funktionen sind so grundlegend, daß sie fast routineartig verwendet werden), so daß Sie bestenfalls ge-

zwungen werden, die Struktur in einer XML-Annäherung noch einmal zu überarbeiten. Im schlimmsten Fall könnte es erforderlich sein, einen Schritt zurück zu gehen (falls Sie Glück haben und das möglich ist) und das SGML-Modell so zu ändern, daß die Umwandlung möglich wird. Wir werden die Probleme bei der Umwandlung von SGML-DTDs in XML-DTDs in Kapitel 7 noch einmal aufgreifen.

Entwicklung einer DTD aus XML-Code

Die schnellste und einfachste Methode, eine XML-DTD anzulegen, ist das Anlegen eines XML-Dokuments und rückwärts vorzugehen. Listing 6.6 zeigt eine einfache Webseite mit grundlegendem Markup.

Listing 6.6: Eine grundlegende Webseite

```
1:   <?xml version="1.0"?>
2:   <page>
3:     <head>
4:       <title>My Home Page</title>
5:     </head>
6:     <body>
7:       <title>Welcome to My Home Page</title>
8:       <para>
9:         Sorry, this home page is still
10:        under construction. Please come
11:        back soon!
12:      </para>
13:    </body>
14:  </page>
```

Anhand des XML-Dokuments aus Listing 6.6 können wir die allgemeine Elementhierarchie bereits abschätzen. Hier habe ich Leerzeichen verwendet, um die Hierarchie darzustellen:

```
<page>
       <head>
          <title>
       <body>
          <title>
          <para>
```

Wenn wir das jetzt in DTD-Syntax umwandeln, erhalten wir eine Annäherung wie die folgende:


```
<!DOCTYPE page [
      <!ELEMENT page (head, body)>
      <!ELEMENT head (title)>
      <!ELEMENT body (title, para)>
]
```

Beachten Sie, daß ich diese DTD so entworfen habe, als würde sie als interne DTD-Untermenge am Anfang des XML-Dokuments erscheinen. Wenn man die DTD bei der Entwicklung innerhalb eines Dokuments behält, erspart man sich eine Menge an Dateiwechseln, bis man sicher ist, daß die DTD funktioniert. Anschließend können Sie die DTD in eine externe Datei verschieben, nachdem die Datei fertiggestellt wurde.

Damit habe ich folgendes ausgedrückt:

- `page` ist das Wurzelelement.
- Das `page`-Element besteht aus einem `head` gefolgt von einem `body`.
- Ein `head`-Element enthält ein `title`-Element.
- Ein `body`-Element enthält ein `title`-Element gefolgt von einem `para`-Element.

Dies ist die Hierarchie der Elemente. Jetzt brauche ich Anzeigen für das Vorkommen, um alle Einschränkungen in der Reihenfolge der Elemente zu spezifizieren und wie oft sie vorkommen dürfen (wenn überhaupt). (Machen Sie sich über diese Dinge jetzt noch keine Sorgen, Sie werden im nächsten Kapitel mehr darüber erfahren.)

Ohne zu weit auf die Details der DTD-Entwicklung vorzugreifen, gibt es einige Dinge, die wir über die Elementmodellierung bereits sagen können. Am Anfang sollten Sie versuchen, alles so offen wie möglich zu gestalten. Durch die Einführung unnötiger Einschränkungen machen Sie es schwieriger, die DTD zu nutzen, und Sie riskieren eine spätere Überarbeitung. Am besten gehen Sie von einem uneingeschränkten, offenen Modell aus, hin zu einem engen, geschlossenen und nicht umgekehrt. Vielleicht stellen Sie sich das folgende vor, weil es sicherstellt, daß ein `title` innerhalb eines `head` ist und mindestens ein `para` innerhalb des `body`:

```
<!ELEMENT page (head, body)>
<!ELEMENT head (title)>
<!ELEMENT body (title?, para)+>
```

Sie sollten zunächst alles so einfach wie möglich gestalten, und Sie könnten die DTD wie folgt belassen:

```
<!ELEMENT page (head, body)>
<!ELEMENT head (title)>
<!ELEMENT body (title?, para)*>
```

Hier habe ich nur die Angabe des Sterns * für das `body`-Element in mein Inhaltsmodell aufgenommen. Dies ist ein Platzhalter für die *optionale Wiederholung*, d.h. ein `body` kann leer sein oder eine endlose Zeichenkette aus `title`- und `para`-Elementen in beliebiger Reihenfolge enthalten.

Um die DTD fertigzustellen, müssen wir nur noch die Terminal-Elemente (Blätter) ausfüllen, die den eigentlichen Text enthalten, oder zumindest keine Elemente mehr. Am besten deklarieren Sie diese als Text:

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT para (#PCDATA)>
```

Machen Sie sich jetzt noch keine Sorgen um die Details – Sie werden in Kapitel 7 mehr darüber erfahren. Ich versuche hier nur, Ihnen ein Gefühl für die wichtigsten Schritte der DTD-Entwicklung zu vermitteln.

Das Anlegen einer DTD aus einer Instanz (einem einzelnen XML-Dokument) wie hier geht den meisten SGML-Verfechtern völlig gegen den Strich, aber SGML wurde hauptsächlich für die Langlebigkeit ausgelegt. Dabei verfolgte man die Idee, daß Sie Ihre Daten in SGML mit einem Markup versehen und sie lange Zeit benutzen können, und zwar weit über die Lebensdauer eines Softwarepakets oder eines Computersystems hinaus. XML ist die Rückseite der Medaille – dort geht es um sofortige Informationsweitergabe mit einer maximalen Lebensdauer, die eher in Monaten denn in Jahren ausgedrückt werden kann.

Wenn Sie XML als Backend einer Datenbank verwenden wollen, müssen Sie natürlich in längeren Zeitabschnitten als nur in wenigen Monaten rechnen. Die Datenbank selbst kann auf vielerlei Arten bereits einen Großteil des DTD-Entwurfs vorgeben, ohne daß Sie zuviel Mehraufwand betreiben müssen (mehr über die Datenbankmodellierung in DTDs erfahren Sie in Kapitel 7).

Eine DTD manuell anlegen

Es gibt zwar bereits Werkzeuge, die Ihnen die Aufgabe, eine DTD anzulegen, viel einfacher machen, aber es gibt kaum einen Ersatz dafür, sie mit einem Stift und einem Blatt Papier zu entwickeln.

Ich persönlich bevorzuge eine Technik, die locker auf dem Storyboarding basiert, das wir aus der Filmindustrie kennen, wo damit kurze Drehfolgen skizziert werden. Wichtig dabei ist, daß man visuell arbeitet, Elemente erkennt, sie in einer Struktur anordnet und dann alle zusätzlichen Details einfügt, wie beispielsweise Attribute. Nachdem alles an Ort und Stelle ist, können Sie die Struktur vereinfachen, gemeinsame Attribute sammeln und Unterstrukturen für die Elemente in Parameter-Entities einordnen.

Der Versuch, eine DTD so komplex wie möglich zu machen, ist manchmal sehr groß. Beachten Sie jedoch, daß Sie sich mit einer DTD nicht profilieren können. Denken Sie nur an die armen Seelen, die irgendwann versuchen müssen, was Sie da erzeugt haben und warum (und das könnten auch Sie selbst sein – mehrere Jahre später). Sie sollten deshalb der Versuchung widerstehen, komplexe Funktionen wie beispielsweise Parameter-Entities zu verwenden, es sei denn, Sie können sie rechtfertigen (beispielsweise aufgrund einer vereinfachten Modifizierung), und dann sollten Sie sie ausreichend dokumentieren.

Egal was Sie verwenden wollen, um Ihre DTDs zu entwickeln (es sind viele verschiedene Modellierungs- und Notationsschemata im Umlauf), Sie werden sehr wahrscheinlich irgendwann Ihre eigene bevorzugte Methode finden. Die folgenden Abschnitte beschreiben die wichtigsten Schritte und besondere Aspekte einer DTD, die mir immer geholfen haben.

Elemente identifizieren

Für relativ einfache Web-Seiten und einige der einfachen Anwendungen, die wir bisher beschrieben haben, kann Ihnen die Eingabe des gewünschten XML-Dokuments und seine Kennzeichnung mit Markup Ihnen sehr wahrscheinlich schon einen ausreichenden Einblick in die Entwicklung Ihrer DTD bieten. Bevor Sie jedoch beginnen, sollten Sie sich eine klare Vorstellung davon machen, was Sie mit der DTD erreichen möchten. Sie werden in Kapitel 7 mehr darüber erfahren, aber Sie müssen wissen, welches Markup Sie unterstützen wollen. Dabei kann es sich um folgendes handeln:

- **Inhalt** – Hier versuchen Sie zu beschreiben, was die Information darstellt. Sie würden also nach Abstraktionen suchen, die Objekte aus der realen Welt darstellen, wie etwa Teilenummern oder Hausadressen.

- **Struktur –** Hier beschäftigen Sie sich mehr mit der Gruppierung von Elementen, wie etwa Listen, Absätze und Tabellen. Diese Elemente brechen die Information in Einheiten auf, aber fügen keine eigentlichen Informationen hinzu.
- **Darstellung –** Hier geht es nur noch darum, wie die Dinge aussehen. Sie sollten dabei an Zeilenumbrüche, horizontale Linien und Zeichenattribute denken, wie etwa Blinken und Unterstreichen. Vermeiden Sie sie.

Vermeiden Sie Darstellungs-Markup

Was das generische Markup betrifft (sowie die Portierbarkeit Ihres XML-Dokuments) sind die Elemente für die Darstellung die schlimmsten und sollten soweit wie möglich vermieden werden.

Wenn Sie glauben, in einem XML-Dokument typographische Verschönerungen anbringen zu müssen, etwa einen Fettdruck, dann sollten Sie versuchen, sich über seine Funktion im klaren zu werden. Fragen Sie sich, warum Sie meinen, es soll fett dargestellt werden. Ist es ein Schlüsselwort? Dann machen Sie ein `KEYWORD`-Element daraus.

Vergleichen Sie den folgenden Abschnitt, in dem nur Darstellungs-Markup verwendet wird:

```
<para><bold>Dies ist ein <em>sehr</em> wichtiger Aspekt.</bold></para>
```

mit dem folgenden Abschnitt, der einen strukturierteren, semantischen Ansatz verwendet:

```
<note><para>Dies ist ein <emphasis>sehr</emphasis> wichtiger Aspekt.</para></note>
```

Beachten Sie, daß ich durch das Anlegen eines separaten `note`-Elements die Flexibilität des Codes wesentlich steigern kann. Wenn ich beschließe, daß der Text überhaupt nicht mehr so wichtig ist, kann ich das `para`-Element einfach »auspacken«. Ich kann den `note`-Elementen das Erscheinungsbild des Texts überlassen, und ihre Behandlung als Informationseinheiten macht es einfacher, sie zu finden und zu verwalten, als scheinbar zufällig gewählte Formatanweisungen.

Während Sie zwar in der Lage sein sollten, alle reinen Darstellungselemente loszuwerden, gibt es immer noch einige, die unvermeidbar sind. Ein Zeilenumbruchelement könnte praktisch sein (obwohl diese Situation besser durch das Einfügen einer Verarbeitungsanweisung für die formatierende oder Seitenlayout-Anwendung gelöst werden sollte), aber einige der vertrauteren Kandidaten, von denen Sie vielleicht denken, Sie bräuchten sie, können Sie vergessen.

Ein Beispiel für ein solches Darstellungselement ist das Element für die horizontale Linie, HR, in HTML. Sie können damit sicher gestalten, aber Sie sollten sich fragen, wo Sie sie wirklich brauchen. Wenn Sie eine Darstellungsfunktion an ein Element koppeln können, wie etwa das Einfügen einer horizontalen Linie vor dem Abschnittsanfang, sollten Sie sorgfältig darüber nachdenken, ob Sie denselben Effekt nicht durch ein Stylesheet erzielen können. Betrachten Sie beispielsweise das folgende XML-Codefragment:

```
<section>
      <title>Abschnitt 4</title>
      <rule/>
</section>
```

Es könnte viel besser wie folgt ausgedrückt werden:

```
<section>
      <title type="rule under">Abschnitt 4</title>
</section>
```

Sie könnten aber auch einfach folgendes schreiben und dem Stylesheet alle darstellungstechnischen Aspekte überlassen:

```
<section>
      <title>Abschnitt 4</title>
</section>
```

Strukturierung der Elemente

Nachdem Sie die Elemente in Ihrem XML-Dokument zugeordnet und benannt haben, müssen Sie sie im nächsten Schritt irgendwie hierarchisch (baumförmig) anordnen. Wie komplex der dabei angelegte Baum ist, hängt größtenteils von Ihrer Anwendung ab. Wenn Sie eine Datenbank modellieren, wollen Sie die Hierarchie relativ flach halten. Immer jedoch müssen Sie die Regeln für die Einrichtung wohlgeformter Elemente beachten und nur ein Wurzelelement verwenden, das alle anderen Elemente aufnimmt. Sie werden feststellen, daß Papier und Bleistift für die Skizzierung einer solchen Struktur sehr hilfreich sind (in Kapitel 7 lernen Sie einige der Modellierungstechniken kennen). Ich persönlich schreibe den Namen jedes Elements auf einen gelben Post-it-Zettel und positioniere diese irgendwo an der Wand hierarchisch. Damit ist es mir möglich, die Elemente ganz einfach beliebig zu verschieben, und wenn ich ein allgemeines Bild brauche, kann ich einfach ein paar Schritte zurücktreten und die Struktur aus objektiver Distanz betrachten.

Bei dieser Strukturierung suchen Sie nach Gruppen- und Container-Elementen. Gruppenelemente sind Dinge wie Listen, Definitionslisten und Glossare, die bestimmte Elemente als Einheiten anordnen. Manchmal möchte man bewußt zusätzliche Elemente anlegen, die andere Elemente aufnehmen, so daß es einfach ist, sie zu verarbeiten.

Beispielsweise macht es eine Container-Liste für mehrere numerierte Absätze einfacher, diese Absätze zu numerieren und die Zahl bei jedem Anfang einer neuen Menge numerierter Absätze zurückzusetzen.

Ein praktischer Hinweis, wann Sie ein Container-Element verwenden sollten, ist, wenn man sich eine Menge, eine Liste oder eine Gruppe von Elementen vorstellen kann. Ist das der Fall, sollten Sie automatisch einen Container vorsehen.

Beispiele für Container-Elemente sind `HEAD` und `BODY` in HTML oder vielleicht separate `MESSAGE.HEADER`- und `MESSAGE.TEXT`-Elemente für eine DTD für eine E-Mail-Nachricht.

Eine ganz allgemeine Regel für das Anlegen von Containern ist, das Inhaltsmodell für das Element zu betrachten. Wenn Sie nicht sofort verstehen, was es bedeutet (versuchen Sie, es laut zu lesen), ohne warten und darüber nachdenken zu müssen, ist es zu komplex und sollte mit Hilfe eines Container-Elements vereinfacht werden. Außerdem werden Sie feststellen, daß diese Container jemandem, der eine XML mit einer DTD anlegt, die Container implementiert, sehr viel mehr Flexibilität beim Verschieben von Elementen ermöglicht. Listing 6.7 zeigt einen Ausschnitt aus einem XML-Dokument, das mit Hilfe mehrerer Container eine Liste anlegt.

Listing 6.7: Ein typischer XML-Ausschnitt unter Verwendung von Container-Elementen

```
1:   <para>There are several techniques:</para>
2:   <list>
3:      <item>
4:           <body>
5:                <para>Blame someone else</para>
6:           </body>
7:      </item>
8:      <item>
9:           <body>
10:               <para>Pretend it wasn't you</para>
11:          </body>
12:     </item>
13:     <item>
14:          <body>
15:               <para>Be honest</para>
16:               <para>Not advisable!</para>
17:          </body>
18:     </item>
19:  </list>
```

Beachten Sie, wie das Aufbrechen eines `item`-Elements in ein `body`-Element, das wiederum `para`-Elemente enthält, dem Autor ermöglicht, die `para`-Elemente zwischen den Einträgen zu verschieben, ohne sich darum kümmern zu müssen, ständig irgendwelche Elemente ein- und wieder auszuwickeln. Diese Struktur würde es auch sehr viel einfacher machen, Listen innerhalb von Listen zu verschachteln usw.

Regeln erzwingen

Bisher haben Sie sich darauf konzentriert, einfach alles festzulegen und zu organisieren. Erst nachdem diese Phasen abgeschlossen sind, sollten Sie sich damit beschäftigen, wie Sie die Realisierung Ihres Modells erzwingen und dabei Elemente optional und zwingend erforderlich machen. Wie ich bereits vorgeschlagen habe, sollten Sie sorgfältig überlegen, wie streng Sie dabei vorgehen wollen, und wieviel Auswertung Sie brauchen. Wenn Ihr XML-Code vom Computer erzeugt wird, vielleicht aus einer Datenbank, ist es vielleicht nicht besonders sinnvoll, Zeit und Arbeit darauf zu verwenden, die DTD besonders streng zu machen. Schließlich haben Sie volle Kontrolle darüber, wie das XML-Markup erzeugt wird. Wird das XML-Markup dagegen von Menschen erzeugt, sollten Sie schon sicherstellen, daß bestimmte Elemente nicht einfach vergessen werden.

Seien Sie jedoch vorsichtig, wenn Sie Elemente optional machen, um das Inhaltsmodell nicht mehrdeutig zu machen.

Beachten Sie, daß XML-Interpreter nicht besonders komplex und nicht in der Lage sind, vorauszulesen, was im XML-Dokument als nächstes steht. An jeder Stelle im Inhaltsmodell muß absolut klar für den Prozessor sein, welches Element als nächstes erlaubt ist, basierend auf dem, was er zu diesem Zeitpunkt bereits kennt.

Wie Sie im nächsten Kapitel lernen werden, ist es manchmal erforderlich, clevere Tricks zu benutzen, wenn Sie diese Mehrdeutigkeiten umgehen möchten.

Zuweisung von Attributen

Nachdem Sie Ihre Elemente in einer hierarchischen Struktur angeordnet und wie erforderlich gruppiert haben, können Sie ihnen Attribute zuweisen (Größe, Farbe, ID usw.). Sie werden feststellen, daß Sie einige der Informationen in Attribute einbinden können, wodurch Sie die beiden Aufgabenbereiche separat halten können.

Es gibt keine richtigen Regeln dazu, wann man Attribute und wann Elemente benutzt, aber es wird viel darüber diskutiert. Sie werden im nächsten Kapitel mehr darüber erfahren, aber hier sollten Sie einfach nach Ihrem Gefühl und dem gesunden Menschen-

verstand vorgehen. Ich beispielsweise habe einen Kopf und eine Größe. Man kann ganz einfach sagen, daß mein Kopf ein Element sein soll, eine physische Komponente, und daß meine Größe, die ich nicht in der Hand halten kann, als Attribut angegeben werden soll. Sie könnten das in der XML-DTD etwa wie folgt modellieren:

```
<!ELEMENT ich (kopf)>
<!ATTLIST ich größe CDATA "  ">
```

Der entsprechende Code für das XML-Dokument würde dann etwa wie folgt aussehen:

```
<ich größe="176 cm">
      <kopf/>
</ich>
```

Hilfe durch Werkzeuge

Es gibt bereits mehrere XML-Editoren. Einige davon befinden sich zwar noch in frühen Entwicklungsphasen, und es ist noch unklar, welche Funktionen von den Benutzern wirklich gebraucht werden, aber es zeichnet sich bereits ein Trend ab. In weniger als einem Jahr haben wir uns auf einem Pfad, der ähnlich dem der HTML-Werkzeuge verlief, von Markup-fähigen Texteditoren bis hin zu einer Art Textverarbeitungspaket zur Bearbeitung von XML-Code bewegt. Ein gutes Beispiel hierfür ist das Office-2000-Paket, dessen Applikationen heute HTML auf der Basis von XML zur Speicherung von Dokumenten verwenden.

Die XML-Editoren, die heute auf dem Markt erscheinen, bieten schon spezifischere XML-Unterstützung, beispielsweise die Auswertung.

XML Pro (`http://www.vervet.com`) und Stilo WebWriter (`http://www.stilo.com`) sind beides ausgezeichnete Editoren. XML Pro bietet eine ausgezeichnete und billige Bearbeitungsumgebung, die Ihnen erlaubt, XML-Dokumente zu bearbeiten und auszuwerten. WebWriter geht noch einen Schritt weiter und erlaubt Ihnen, direkt aus dem XML-Dokument eine Erstfassung der DTD zu erzeugen. Ich bin mir sicher, daß es immer mehr XML-Werkzeuge mit diesen Möglichkeiten geben wird, weil es relativ einfach ist, zumindest ein Grundgerüst einer DTD aus dem Markup in einem XML-Dokument zu erzeugen.

Eine DTD für eine Homepage

Die Grundlagen der XML-DTDs und der DTD-Modellierung haben Sie bereits kennengelernt. Jetzt sollten Sie dieses Wissen in die Praxis umsetzen und eine DTD für einen sehr einfachen Dokumenttyp erzeugen, eine Web-Homepage für einen Anfänger. Wir verwenden dafür noch einmal das Beispiel aus Kapitel 2, das Sie in Listing 6.8 sehen.

Listing 6.8: Eine typische XML-Instanz

```
<?xml version="1.0"?>
 <home.page>
   <head>
     <title>
       My Home Page
     </title>
     <banner source="topbanner.gif"/>
   </head>
   <body>
     <main.title>
       Welcome to My Home Page
     </main.title>
     <rule/>
     <text>
       <para>
         Sorry, this home page is still
         under construction. Please come
         back soon!
       </para>
     </text>
   </body>
   <footer source="foot.gif"/>
 </home.page>
```

Wenn wir dieses Markup in den Editor XML Pro kopieren, ohne eine DTD anzugeben (siehe Abbildung 6.5), können wir anschließend beliebig Elemente hinzufügen, entfernen und verschieben. Nachdem Sie genug experimentiert haben und mit dem Ergebnis zufrieden sind, können Sie anhand der im linken Fenster gezeigten Baumstruktur erkennen, wie Sie das Ganze strukturiert haben. Sie können Ihren Baum ganz einfach durchlaufen, indem Sie die Zweige auf- und dann wieder zusammenklappen. Die Baumstruktur gibt Ihnen einen guten Ansatzpunkt, die Struktur für Ihre DTD zu beschreiben.

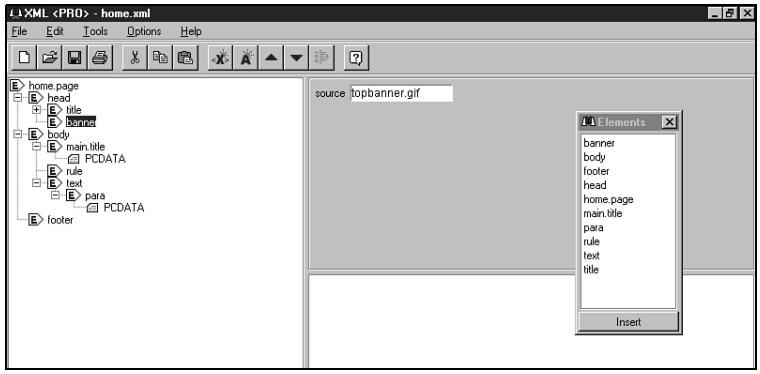


Abbildung 6.3: Experimentieren mit der Struktur im Editor XML Pro.

Wenn Sie dasselbe XML-Markup in den Stilo WebWriter kopieren, ist es nicht ganz so einfach, sich innerhalb der Struktur zu bewegen, wie es bei XML Pro der Fall ist, aber der WebWriter gleicht diese Schwäche aus, indem er Ihnen erlaubt, die DTD wirklich anzulegen, wie in Abbildung 6.4 gezeigt.

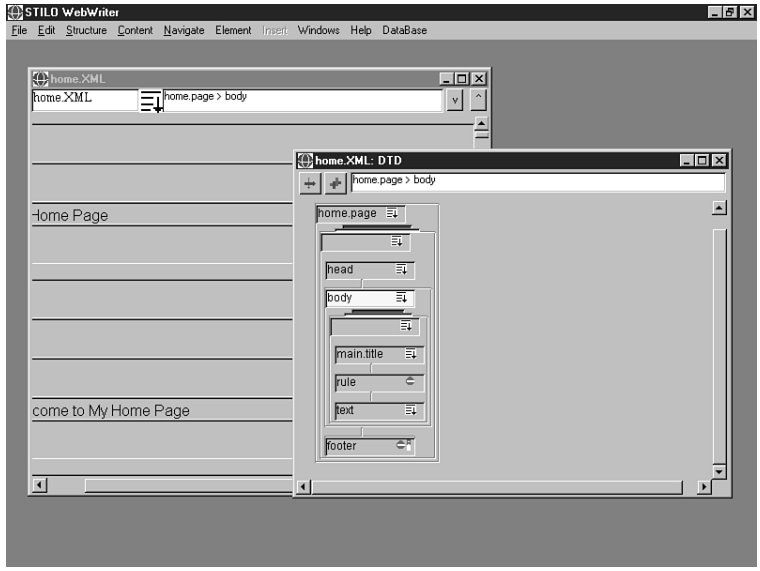


Abbildung 6.4: Anlegen der DTD im Stilo WebWriter.

Jetzt wollen wir betrachten, was der WebWriter aus unserer einfachen Homepage macht. Listing 6.9 zeigt die extrahierte DTD (ich habe ein paar leere Zeilen und Leerzeichen eingefügt, um sie etwas lesbarer zu machen).

Listing 6.9: Die extrahierte DTD

```
1: <?xml version = "1.0" ?>
2: <!ELEMENT home.page  (#PCDATA ¦ head ¦ body ¦ footer)* >
3:
4:  <!ELEMENT head    (#PCDATA ¦ title ¦ banner)* >
5:
6:  <!ELEMENT title    (#PCDATA) >
7:
8:  <!ELEMENT banner     EMPTY >
9:   <!ATTLIST banner     source CDATA "" >
10:
11:  <!ELEMENT body    (#PCDATA ¦ main.title ¦ rule ¦ text)* >
12:
13:  <!ELEMENT main.title (#PCDATA) >
14:
15:  <!ELEMENT rule     EMPTY >
16:
17:  <!ELEMENT text    (#PCDATA ¦ para)* >
18:
19:  <!ELEMENT para    (#PCDATA) >
20:
21:  <!ELEMENT footer     EMPTY >
22:  <!ATTLIST footer     source CDATA "" >
23:
24:  <!ENTITY lt "<">
25:  <!ENTITY gt ">">
26:  <!ENTITY apos "'">
27:  <!ENTITY quot """>
28:  <!ENTITY amp "&">
```

Offensichtlich ist der WebWriter dabei ganz vorsichtig vorgegangen und hat jedem Element einen gemischten Inhalt zugestanden (`#PCDATA` ist in allen Inhaltsmodellen erlaubt), und es war sinnvoll, keinerlei Annahmen über die Art der Attribute zu treffen, aber grundsätzlich ist ein gutes Gerüst entstanden. Jetzt müssen Sie nur noch die Inhaltsmodelle straffen und die Attributdeklarationen einfügen. Ein mögliches Ergebnis sehen Sie in Listing 6.10.

Listing 6.10: Die endgültige DTD

```
1: <!ELEMENT home.page  (head ¦ body ¦ footer) >
2:
3:  <!ELEMENT head    (title ¦ banner?) >
4:
5:  <!ELEMENT title    (#PCDATA) >
6:
7:  <!ELEMENT banner     EMPTY >
8:  <!ATTLIST banner
9:            src CDATA  #REQUIRED
10:           alt CDATA #IMPLIED
11:           align (top ¦ middle ¦ bottom) #IMPLIED >
12:
13:  <!ELEMENT body    (main.title ¦rule ¦ text) >
14:
15:  <!ELEMENT main.title (#PCDATA) >
16:
17:  <!ELEMENT rule     EMPTY >
18:  <!ATTLIST rule
19:            align (left¦right¦center) #IMPLIED
20:            size  NMTOKEN #IMPLIED
21:            width CDATA #IMPLIED >
22:
23:  <!ELEMENT text    (#PCDATA ¦ para)* >
24:
25:  <!ELEMENT para    (#PCDATA) >
26:
27:  <!ELEMENT footer     EMPTY >
28:  <!ATTLIST footer
29:            src CDATA  #REQUIRED
30:            alt CDATA #IMPLIED
31:            align (top¦middle¦bottom) #IMPLIED >
32:
33:  <!ENTITY lt "<">
34:  <!ENTITY gt ">">
35:  <!ENTITY apos "'">
36:  <!ENTITY quot """>
37:  <!ENTITY amp "&">
```

Beachten Sie, daß wir jetzt gemäß der XML-Spezifikation eine DTD verwenden und damit die Deklarationen der Standardzeichen-Entities eingebunden haben (`lt`, `gt`, `apos`, `quot` und `amp`).

Zusammenfassung

Sie haben die grundlegenden Informationen über XML-DTDs und ihre Unterteilung in zwei Untermengen kennengelernt: eine interne Untermenge, die im XML-Dokument enthalten ist, und eine externe Untermenge, die in einer externen Datei angelegt wird. Außerdem haben Sie erfahren, wie man unter Verwendung eines öffentlichen Bezeichners und eines Systembezeichners auf die externe DTD-Untermenge verweist.

Sie haben einige der Faktoren kennengelernt, die bestimmen, ob man eine DTD verwenden und ob sie intern oder extern angelegt werden soll. Außerdem sind Sie bereits vertraut mit dem wichtigsten Inhalt einer DTD.

Damit sollten Sie einen allgemeinen Überblick über die Grundlagen der DTD-Modellierung und einige der dabei möglicherweise auftretenden Probleme haben. In Kapitel 7 werden wir die DTDs noch einmal detailliert beschreiben. Lassen Sie sich nicht entmutigen – DTDs sind kompliziert, aber sie können auch sehr, sehr einfach sein. Wenn alles andere fehlschlägt, ist folgendes immer noch ein perfektes, gültiges XML-Dokument mit interner DTD-Untermenge, und vielleicht brauchen Sie ja auch gar nicht mehr:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE DOC [
<!ELEMENT DOC (#PCDATA)>
]>
<DOC>Ich kann damit den gesamten Text in
meinem Dokument unterbringen und das gesamte
andere Markup vergessen. </DOC>
```

F&A

F Gibt es eine Begrenzung für die Länge des Attributnamens?

A *Nein, in XML gibt es das nicht. Solange Sie nur die erlaubten Zeichen verwenden, kann ein Name beliebig lang sein (in SGML müssen Sie dazu spezielle Vorkehrungen treffen – Sie müssen die SGML-Deklaration abändern, um lange Namen zu erlauben).*

F Wie viele Elemente sollten Sie in einer DTD anlegen?

A *Das ist ganz von Ihrer Anwendung abhängig. HTML beinhaltet etwa 77 Elemente, und einige der industriellen DTDs haben mehrere hundert. Es ist nicht immer die Anzahl der Elemente, die die Komplexität der DTD bestimmt. Durch die Verwendung von Container-Elementen in der DTD (die ebenfalls als Elemente gezählt werden) ist die Authoring-Software in der Lage, die möglichen Auswahlen zu beschränken und zwingend erforderliche Elemente automatisch einzugeben. Die Arbeit mit einer der großen industriellen DTDs kann aber tatsächlich viel einfacher sein, als HTML zu schreiben! Weil HTML Ihnen mehr freie Auswahl bietet, haben Sie eine viel bessere Vorstellung von der Aufgabe aller Elemente statt nur derer, die Sie brauchen.*

F Muß man alle XML-Dokumente auswerten?

A *Nein, aber wenn Sie nicht wirklich sicher sind, daß die Dokumente gültig sind, können Sie nicht vorhersagen, was auf der empfangenden Seite damit passiert. Die XML-Spezifikation legt zwar Regeln für die XML-Interpreter fest und bestimmt, was sie tun sollen, wenn ungültiger Inhalt erkannt wird, aber diese Anforderungen sind häufig recht locker. Bei der Auswertung passiert nichts – außer daß ein bißchen zusätzliche Arbeit für Sie anfällt.*

F Ich kann meine XML-Dokumente auswerten, aber wie kann ich prüfen, ob meine DTD korrekt ist?

A *Werten Sie einfach ein XML-Dokument aus, das diese DTD benutzt. Es gibt nicht so viele Werkzeuge speziell für das Prüfen von DTDs, wie es für die Auswertung von Dokumenten gibt. Wenn ein XML-Dokument jedoch mit einer bestimmten DTD ausgewertet wird, wird auch die DTD geprüft, und ihre Fehler werden gemeldet.*

Übungen

1. Legen Sie eine inhaltsbasierte XML-DTD für eine einfache E-Mail-Nachricht an.
2. Legen Sie eine (partielle) DTD an, die Listen verarbeiten kann. Versuchen Sie, zwei Versionen der Elementstrukturen zu erzeugen, wobei eine ein Attribut für die Beschreibung der Numerierungsart (beispielsweise numeriert oder mit Spiegelstrichen versehen) benutzt und die andere separate Eintragselemente (wie etwa `numberlist` und `bulletlist`). Vergleichen Sie die beiden Versionen, und beschreiben Sie ihre Vor- und Nachteile in Hinblick auf das einfache Authoring und auf die einfache Formatierung mit Hilfe eines Stylesheet.

Entwicklung komplexerer DTDs

Woche 1

In Kapitel 6 haben Sie XML-DTDs und die Grundlagen für ihre Entwicklung kennengelernt. Heute werden Sie Ihr Wissen über DTDs erweitern und vertiefen. Unter anderem werden Sie folgendes tun:

- Einige der Aspekte der DTD-Entwicklung und der Informationsmodellierung erweitern, die gestern aufgetaucht sind.
- Sich die technischen Aspekte von DTDs ansehen.
- Funktionen und Tricks erlernen, die es viel einfacher machen, eine DTD anzulegen.

Ein Wort zur Warnung: Ich habe dem Titel dieses Kapitels das Wort *komplexerer* hinzugefügt, um Sie abzuschrecken. Bisher habe ich in diesem Buch vorausgesetzt, daß Sie mehr Wissen über HTML mitbringen, als mir lieb ist. Außerdem habe ich öfter auf SGML verwiesen, als ich es hätte tun sollen. Aber wenn es um DTDs geht, ist XML dem SGML sehr ähnlich, und es kann nicht völlig ignoriert werden.

Sie brauchen nichts über SGML zu wissen, um dieses Kapitel zu verstehen, aber wenn Sie es tun, wird Ihnen vieles klarer sein.

Informationsfülle

Gestern haben Sie erfahren, daß der Inhalt und die Komplexität einer DTD stark durch die jeweilige Anwendung bestimmt werden. Im wesentlichen kann man für XML-Dokumente und ihr Markup von einer alten Bauernregel ausgehen: *Von nichts kommt nichts*. Das bedeutet, wenn Sie mit einem Informationsabschnitt in einem XML-Dokument etwas tun wollen, müssen Sie ihn identifizieren. Dabei ist es egal, ob Sie dazu Elemente oder Attribute verwenden. Wenn Sie jedoch kein Markup angegeben haben, werden Sie entweder nicht in der Lage sein, ihn überhaupt zu finden, oder *wenn* Sie ihn finden, dann können Sie nichts damit anfangen.

Und noch etwas, was ich Ihnen mit auf den Weg gebe: Sie können nichts extrahieren, was Sie nicht mit einem Tag versehen haben.

Eine DTD, die viel (informatives) Markup enthält, wird auch als *umfassende* DTD bezeichnet. Als DTD-Entwickler werden Sie immer einem schwierigen Kompromiß zwischen der Komplexität der DTD und dem Informationsreichtum des betreffenden XML-Dokuments gegenüberstehen. Wenn Menschen die Autoren dieser Dokumente sind, kann die Komplexität der DTD zu einer großen Hürde werden. Wählen Sie immer das umfassendste Informationsmodell, das möglich ist:

- Es ist viel einfacher, Informationen zu verwerfen, als sie nachträglich einzufügen.

  Überlegen Sie, was passiert, wenn Sie von XML oder auch von HTML ausgehen, hin zu einfachem Text, der überhaupt kein Markup enthält. Diese Umwandlung ist sehr einfach, weil man von einer höheren Ebene des Informationsreichtums hin zu einer tieferen Ebene geht. Viel schwieriger ist es, in die andere Richtung zu gehen. Dasselbe gilt für die Bewegung von SGML nach XML und von XML nach HTML. Bei jeder Umwandlung geht Information verloren, wenn man von einer informationsreichen Ebene (SGML) zu einer relativ informationsarmen Ebene (HTML) geht.

- Sie können nicht immer alle Medien und alle möglichen Anwendungen berücksichtigen. Egal wie sorgfältig Sie Ihre Applikation planen, es wird immer irgend etwas geben, woran Sie nicht gedacht haben.

  Bei SGML passiert es häufig nach dem Markup aller Daten, daß Sie plötzlich erkennen, was Sie damit alles tun könnten. Wenn Sie sich selbst auf das minimal Notwendige für die aktuelle Applikation beschränken, manövrieren Sie sich womöglich selbst in eine Ecke, und es gibt keine Möglichkeit zur Weiterentwicklung mehr.

  Wie in mechanischen Systemen unterliegen Informationsobjekte häufig einer Art Entropiegesetz. In mechanischen Systemen besagt das Entropiegesetz, daß ein System dazu tendiert, sich immer weiter zu desorganisieren. Wenn man Gas in einem Raum ausströmen läßt, werden sich nach einer Weile alle seine Moleküle relativ gleichmäßig verteilt haben. Die Wahrscheinlichkeit, daß alle Moleküle auf magische Weise wieder ihren Weg zurück in den ursprünglichen Container finden, ist relativ gering (aber auf der Quantenebene ist es durchaus möglich). Auf dieselbe Weise zeigt der Informationsreichtum eines XML-Dokuments immer die Tendenz, in den Unrat abzugleiten (hohe Informationsentropie). Je höher man sich auf dieser Rampe befindet (je umfassender Ihre Ausgangssituation ist), desto länger kann man den völligen Informationsverlust hinauszögern.

- Merken Sie sich den Verlauf der Legacy- (Vererbungs-)Daten.

  In SGML-Kreisen wurden Daten, für die es entweder kein Markup gab oder für die ein Markup in einer früheren Version einer DTD erfolgt ist, als *Legacy-Daten* bezeichnet. Die Sünden der Väter vererben sich auf die Söhne – wie in der Bibel. Häufig handelt es sich aber auch um ein viel größeres Problem, als mit allem neuen erzeugten Material auftreten könnte.

  Sie könnten durch die Neuzuordnung der Attribute viel erreichen (wie Sie in Kapitel 11 noch erfahren werden), aber mehr oder weniger sind Sie auf die Elemente beschränkt, mit denen Sie angefangen haben.

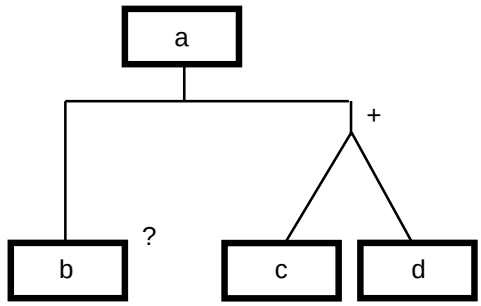
Es gibt einen Vorschlag, in XML eine Funktion aus HyTime zu implementieren, die sogenannten *architektonischen Formulare*, aber es ist eher unwahrscheinlich, daß dieser Vorschlag angenommen wird. Während heute zwar viele Leute SGML kennen, haben sich selbst die SGML-Gurus bisher leider blind gegenüber dem Erscheinen von HyTime gezeigt. Architektonische Formulare sind nicht einfach zu verstehen, und es gibt nur wenig Software, die sie unterstützt.

Unterschätzen Sie nicht die Geschwindigkeit, in der Sie Legacy-Daten anhäufen, und unterschätzen Sie nicht die Langlebigkeit Ihrer Daten. Zu leicht beginnt man mit einer funktionierenden DTD, glaubt, daß sie nur relativ begrenzte Aufgaben übernimmt und daß sie in ein paar Monaten schon ersetzt wird. Wenn Sie das tun, öffnen Sie sich die Tür zu einer lebenslangen Karriere als Editor.

Sie müssen nämlich jedes bisher erzeugte XML-Dokument abändern, und immer wenn Sie glauben, Sie seien fertig, ändert sich wieder etwas, und Sie müssen von vorne beginnen. Das ist der Lauf der Legacy-Daten.

Visuelle Modellierung

In Kapitel 6 haben Sie gelernt, daß der Aufbau der Hierarchie einer der grundlegenden Schritte im Entwicklungsprozeß einer DTD ist – Sie verwenden die Inhaltsmodelle in der DTD, um die Elemente in einem XML-Dokument, das die DTD benutzt, in einer hierarchischen, baumähnlichen Struktur anzuordnen.



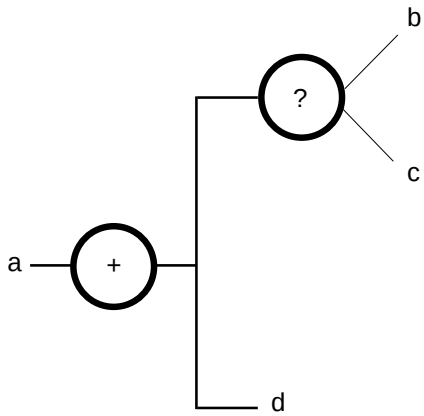

```
<!ELEMENT a (b?.(c | d)+>
```

Abbildung 7.1:
DTD-Modellierung in Form eines Blockdiagramms.

Es gibt viele Techniken zur Informationsmodellierung, die jedoch in diesem Buch nicht detailliert beschrieben werden können. Wichtig dabei ist, daß sie alle eine Art visueller Hilfe empfehlen, die Sie unterstützen soll. Dasselbe gilt für die DTD-Entwicklung, aber

jeder Experte verwendet eigene Symbole. Einige betreiben die Informationsmodellierung unter Verwendung von Rechtecken, die Sie bereits aus Flußdiagrammen kennen, wie in Abbildung 7.1 gezeigt.

Als Alternative gibt es die etwas spärlichere Darstellungsform, die Sie in Abbildung 7.2 sehen.




```
<!ELEMENT a ((b | c)? d )+>
```

Abbildung 7.2:
Vereinfachte Diagramm-DTD-Modellierung.

Beachten Sie, daß in beiden Modellierungsgraphen versucht wird, die Abfolge- und ODER-Modelle darzustellen (ein Modell, in dem es zwei oder mehr Alternativen gibt, aus denen eine ausgewählt werden muß), wobei unterschiedliche Verbindungslinien zwischen den Objekten verwendet werden (diagonale Linien für eine ODER-Auswahl, gerade Linien für eine einfache Beziehung und rechtwinklige Verbindungen für Abfolgen). Diese Art der grafischen Modellierung wurde auch in das beste SGML-DTD-Modellierungspaket aufgenommen, Near & Far von Microstar (`http://www.microstar.com`), für das Sie in Abbildung 7.3 ein Beispiel sehen.

Man kann sich kaum eine bessere Methode vorstellen, DTDs visuell zu modellieren, als mit diesen Modellarten, insbesondere in einem Softwarepaket, wo man alle Zweige des DTD-Baums bei der Bearbeitung beliebig auf- und zuklappen kann. Wenn die DTD jedoch immer größer wird, wird es immer schwieriger, einen aussagekräftigen Überblick zu bewahren. Sie sollten nicht vergessen, daß es sich hierbei hauptsächlich um SGML-Werkzeuge handelt und daß man sie hauptsächlich in einer völlig anderen Größenordnung einsetzt, als in XML sinnvoll ist. Während man in einer XML-Anwen-

dung möglicherweise ein paar Dutzend Elemente und mehrere hundert Zeilen in einer DTD benötigt, verwenden SGML-Applikationen in der Regel mehrere hundert Elemente und mehrere tausend Zeilen mit Deklarationen in einer DTD.

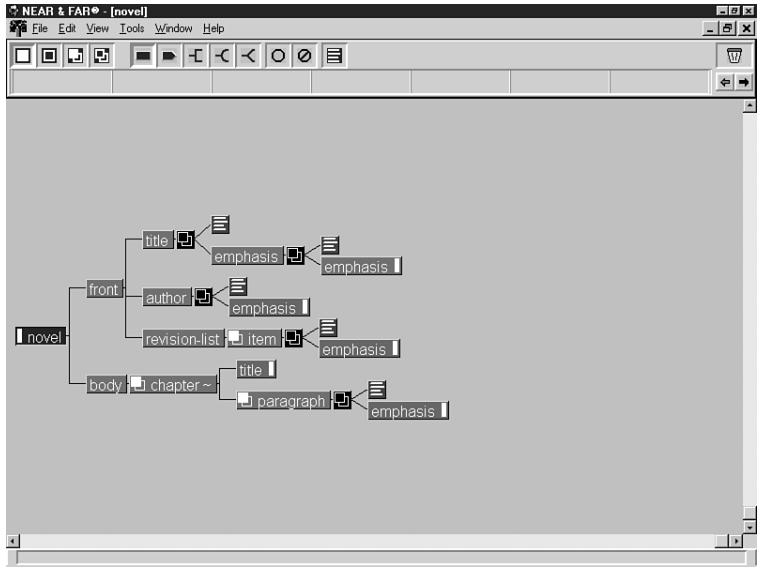


Abbildung 7.3: Visuelle Modellierung in Near & Far.

Es ist jedoch nicht nur die Anzahl der Elemente, die die DTD unhandlich macht. Häufig werden Elemente mit einer darstellungstechnischen Funktion (wie hervorgehobener Text) in fast jedem Element wiederverwendet. Wenn Sie diese Modelle expandieren, kann selbst eine wohldurchdachte visuelle Darstellung zu unübersichtlich werden, und Sie brauchen eine andere Methode, DTDs dieser Dimension oder Komplexität zu verwalten (siehe Abbildung 7.4).

Es gibt natürlich Techniken für die Handhabung von DTDs dieser Größe, wie beispielsweise die Aufteilung der DTD in Module einer sinnvolleren Größe (wie Sie später in diesem Kapitel noch erfahren werden). Aber auch dabei gelangt man irgendwann zu einem Punkt, wo sie sich fragen, ob es sinnvoll ist, so weiterzumachen. Abbildung 7.5 zeigt ein Public-Domain-Softwarepaket, EZDTD, das eine Art tabellarischer Anzeige der Elemente in der DTD ausgibt (besuchen Sie `http://www.download.com`, und suchen Sie nach »EZDTD«).

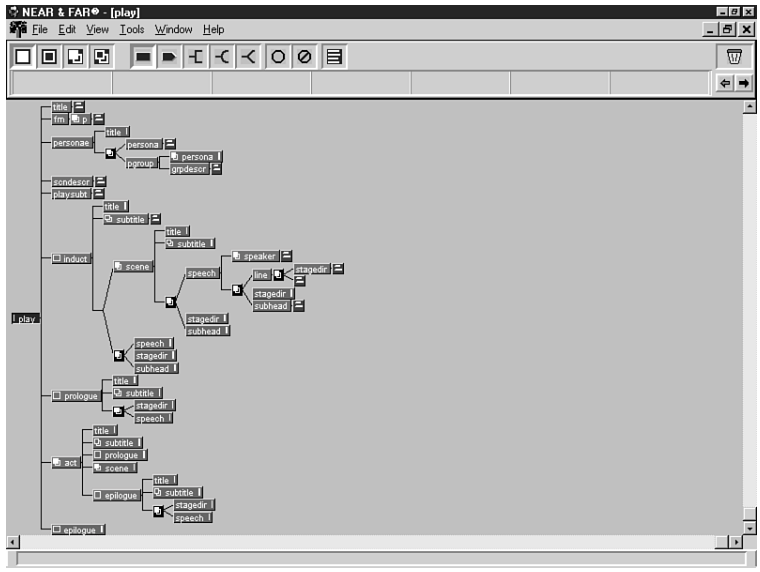


Abbildung 7.4: Auch die visuelle Modellierung kann ihre Grenzen erreichen.

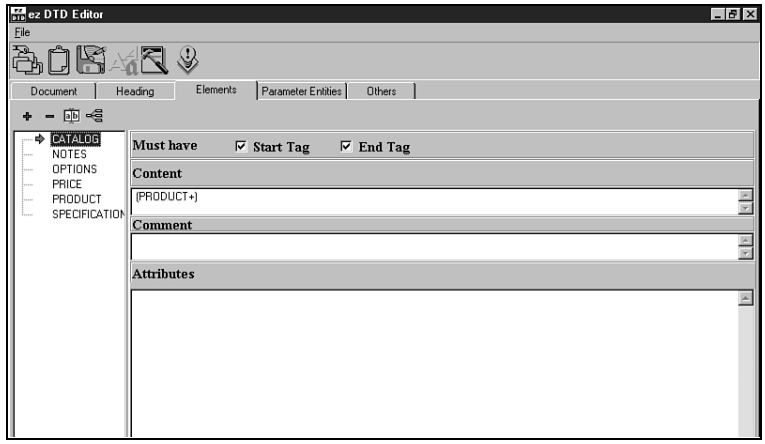


Abbildung 7.5: Tabellarische Modellierung in EZDTD.

Hier haben Sie nur eine Liste der Elemente, und Sie können das Inhaltsmodell für jeweils ein Modell gleichzeitig anzeigen. Für einfache XML-Anwendungen und kleinere DTDs ist diese Art Werkzeug vielleicht ausreichend, insbesondere wenn Sie erst einmal gelernt haben, DTDs anzulegen. Leider gibt es keine Möglichkeit, einen Baum an-

zuzeigen, um die Beziehungen zwischen den Elementen aufzuzeigen. Und auch hier ist die Nützlichkeit fraglich, wenn die DTD erst einmal eine bestimmte Größe angenommen hat (siehe Abbildung 7.6).

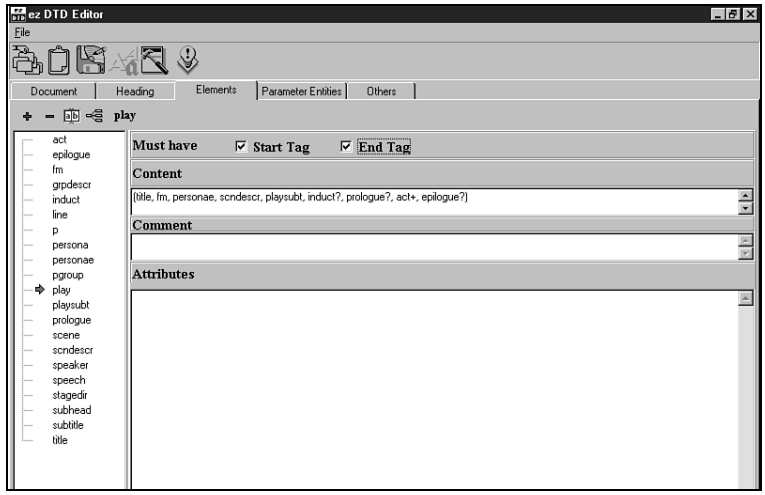


Abbildung 7.6: Auch eine vereinfachte Form der visuellen Modellierung erreicht irgendwann die Grenzen der Nützlichkeit.

Es ist vielleicht sinnvoller, völlig aufzugeben, die ganze DTD visualisieren zu wollen, obwohl ich das Public-Domain-Paket DTD2HTML für Perl von Earl Hood wirklich empfehlen kann (`http://www.oac.uci.edu/indiv/ehood/perlSGML.html`). Dieses Paket wandelt eine DTD in eine Menge miteinander verbundener HTML-Seiten um. Die Ausgabe wird Ihnen während der eigentlichen Entwicklung der DTD keine große Hilfe sein, aber sie ist extrem nützlich für die Prüfung der Ergebnisse Ihrer Arbeit, und sie bietet eine ausgezeichnete Möglichkeit, die fertige DTD zu dokumentieren.

Statt die DTD visuell zu modellieren, werden Sie es irgendwann einfacher finden, ein konservativeres Paket zu nutzen, wie etwa den DTD Generator von Innovation Partner (`http://www.mondial-telecom.com/oddyssee`), das Sie in Abbildung 7.7 sehen.

Dieses eher schmucklose Paket stellt einen Mittelweg zwischen den beiden Lagern zur Verfügung. Sie haben die Leichtigkeit all dieser praktischen Schaltflächen, die Ihnen helfen, die Teile des Inhaltsmodells schnell hinzuzufügen, aber es gibt keine visuelle Anzeige. Statt dessen können Sie aber sehr schnell zwischen der einfach zu verwendenden Oberfläche und dem reinen DTD-Code umschalten.

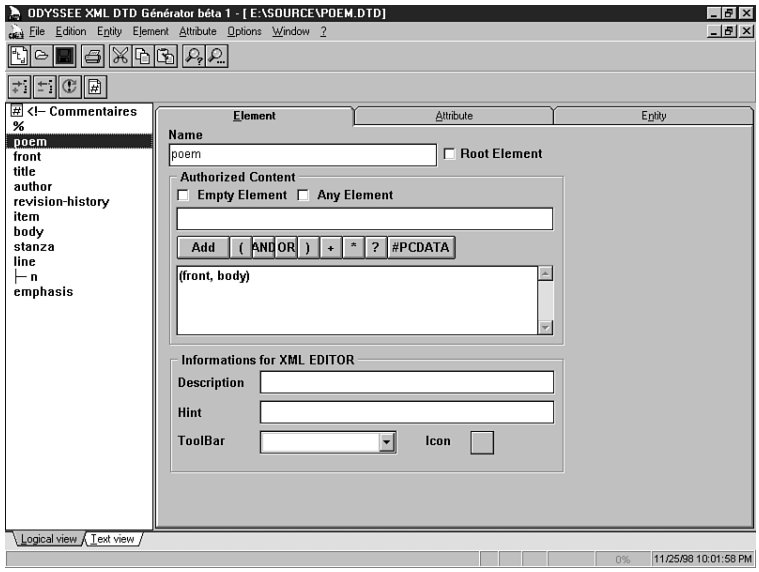


Abbildung 7.7: Anlegen einer DTD mit dem XML-DTD-Generator-Paket von Innovation Partners.

XML-DTDs aus anderen Quellen

Wie bereits vorgeschlagen, ist eine Alternative, die Komplexitäten beim Entwurf und der Entwicklung von DTDs zu umgehen, den DTD-Entwurf beiseite zu lassen und einfach mit einem repräsentativen XML-Dokument zu arbeiten. Sie müssen natürlich alle wahrscheinlichen Variationen des Dokuments berücksichtigen, und Sie brauchen vielleicht mehrere Dokumente, wenn Ihre XML-Anwendung größer wird. Nachdem Sie jedoch alle Möglichkeiten des XML-Dokuments erkundet haben, können Sie Werkzeuge benutzen, um zumindest den Kern der DTD anzulegen. Während das in einer SGML-Anwendung undenkbar, ja ketzerisch wäre, stellt dieser Ansatz für eine XML-Anwendung häufig eine ganz realistische Möglichkeit dar. Abbildung 7.8 zeigt ein Dokument aus der Praxis, das in XML Pro geladen wurde (`http://www.vervet.com`). Sie sehen, daß ein guter XML-Editor Ihnen erlaubt, die Struktur des Dokuments zu durchlaufen und bei Bedarf Elemente und Attribute hinzuzufügen. Sie werden vielleicht feststellen, daß Sie einige Schritte unnötig wiederholen, aber es ist ganz einfach, die Mehrfachdeklarationen anschließend zu konsolidieren.

Es gibt außerdem einen sehr wesentlichen Vorteil bei diesem Ansatz, den Ihnen ein spezielles DTD-Werkzeug nicht bietet: Wenn Sie sehen, inwieweit ein fertiges XML-Dokument Ihre (vorgeschlagene) DTD nutzt, erhalten Sie einen ersten Eindruck, wie es für einen menschlichen Autor aussieht, mit Ihrer DTD zu arbeiten.

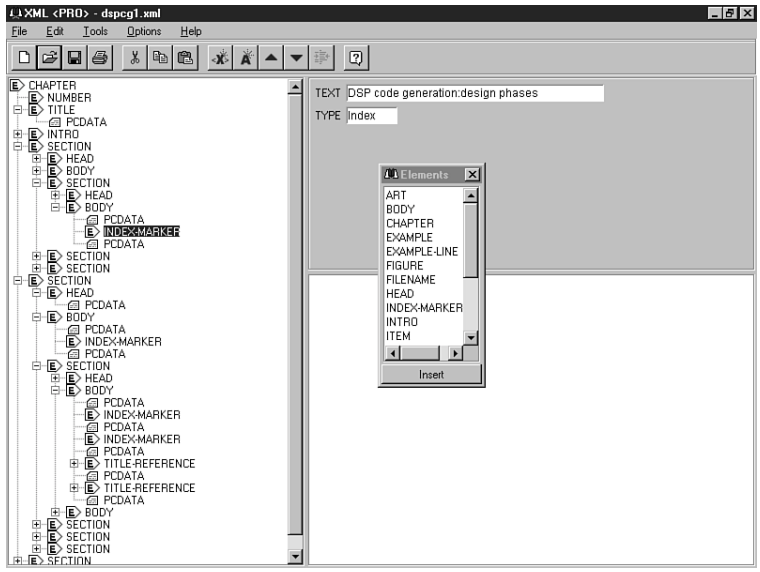


Abbildung 7.8: Modellierung des Dokuments vor dem Anlegen einer DTD.

Es gibt noch eine Möglichkeit, sich selbst mitten in den DTD-Entwicklungsprozeß zu katapultieren – eine Hintertür. Viele der großen Softwarepakete, wie etwa der FrameMaker von Adobe und sogar Microsoft Office, unterstützen in der nahen Zukunft XML als Ausgabeformat. Das eröffnet uns viele Möglichkeiten, die Struktur eines geplanten XML in einem dieser Softwarepakete zu perfektionieren und die DTD dann aus diesem Dokument zu erzeugen oder sie ganz zu überspringen.

Eine interessante Entwicklung in dieser Richtung ist das Softwarepaket Majix von Tetrasix (`http://www.tetrasys.fr/ProduitsFrame.htm`). Dieses Softwarepaket (siehe Abbildung 7.9) erlaubt Ihnen, ein XML-Dokument aus einem Microsoft-RTF-Dokument zu erzeugen. Wenn Sie beispielsweise Microsoft Word einsetzen und genug Disziplin aufbringen, um die Stile korrekt zuzuweisen (Sie wissen, eine DTD ist nur eine Methode, die Struktur eines XML-Dokuments zu beschreiben), können Sie diese Stile nutzen, um die Umwandlung in XML-Elemente vorzubereiten.

Ich bin kein Hellseher, aber dennoch ziemlich sicher, daß es in der Zukunft noch mehr solcher Umwandlungswerkzeuge geben wird und daß sie uns allen das Leben sehr viel leichter machen.

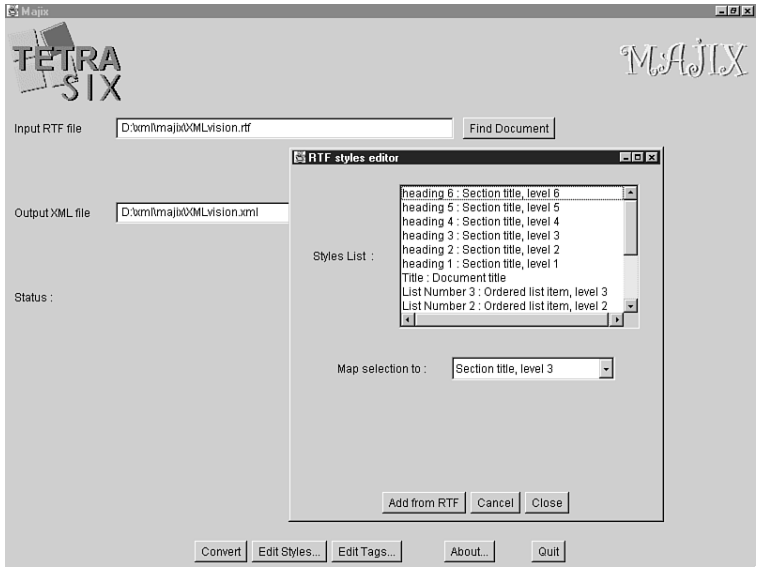


Abbildung 7.9: Konvertierung von RTF in XML mit Majix von Tetrasix.

Modellierung relationaler Datenbanken

Während der ganzen Diskussion der DTD-Entwicklung habe ich immer wieder davon gesprochen, daß die Elemente hierarchisch angeordnet werden sollen. Manchmal will oder braucht man aber gar keine Hierarchie, und man will das Modell »flach« halten. Beispielsweise könnten Sie XML verwenden, um eine relationale Datenbank zu modellieren, wofür XML optimal geeignet ist (die geplanten XML-Datentyp-Initiativen beginnen, Früchte zu tragen).

Betrachten Sie die relativ einfache Beschreibung einer relationalen Datenbank in Listing 7.1. Das Thema der Datenbankmodellierung in XML verdient zwar eine sehr viel umfassendere Behandlung, als ich sie hier bieten kann (und es gibt einige ausgezeichnete Bücher zu diesem Thema), aber selbst ein erster Blick zeigt, daß man gar nicht so viel Tiefe braucht. Die hier gezeigte einfache Datenbank braucht nur vier Ebenen:

- Die Datenbank (das Wurzelelement `music`)
- Die Datenbanktabellen (die Elemente `artists` und `disks`)
- Die Datensätze (die Elemente `artist` und `disk`)
- Die Datenbankfelder (die Elemente `name`, `label`, `title`, `data` und `number`)

Listing 7.1: Eine einfache Datenbank in XML

```
1: <!DOCTYPE music SYSTEM "mymusic.dtd">
2: <music>
3:   <artists>
4:     <artist>
5:       <name>Cyndi Lauper</name>
6:       <label>Sony</label>
7:       <title>Twelve Deadly Guns</title>
8:     </artist>
9:     <artist>
10:      <name>Kate Bush</name>
11:      <label>EMI</label>
12:      <title>Hounds of Love</title>
13:    </artist>
14:   </artists>
15:   <disks>
16:     <disk>
17:       <title>Twelve Deadly Guns</title>
18:       <date>1994</date>
19:       <number>EPC 477363 2</number>
20:     </disk>
21:     <disk>
22:       <title>Hounds Of Love</title>
23:       <date>1985</date>
24:       <number>CDP 7 46164 2</number>
25:     </disk>
26:   </disks>
27: </music>
```

Nachdem Sie diese Struktur haben, ist es ganz einfach, Tabellen- und Datensatzschlüssel einzufügen, indem man Attribute zuweist. Auf diese Weise könnten Sie eine erzwungene Teilung anlegen, wobei Sie Elemente nutzen, um den Inhalt und die Struktur der eigentlichen Datenbank zu beschreiben, und die Attribute für die (internen) Daten reserviert werden, die die Beziehungen zwischen den Daten beschreiben.

Elemente oder Attribute?

Die Separierung von Elementen und Attributen in einer XML-Datenbankbeschreibung ist nur eine der vielen Methoden, eine Unterscheidung zwischen Elementen und Attributen zu treffen. Diese Frage taucht in SGML-Kreisen immer wieder auf und wurde nicht wirklich zufriedenstellend beantwortet. Das Datenbankbeispiel zeigt eine mögli-

che Antwort, aber nur eine von vielen. Ich habe auch keine schnelle Antwort, aber ich kann die folgenden Gedankengänge anbieten, die Ihnen bei der Auswahl helfen könnten:

- Elemente können mit Hilfe von architektonischen Formularen neu zugeordnet und Attribute mit XLink neu zugeordnet werden. Aus dieser Perspektive gibt es keinen entscheidenden Grund, sich für Element oder Attribut zu entscheiden (weil beide neu zugeordnet werden können).
- Historisch gesehen, waren Attribute für nicht berührbare, abstrakte Eigenschaften vorgesehen, wie beispielsweise Farbe und Größe, während Elemente für physische Komponenten reserviert sind. Die Abweichung von dieser Vorgehensweise könnte die Benutzer verwirren, insbesondere in einem SGML-Kontext.
- Es könnte einfacher sein, mit den XML-Werkzeugen Elemente zu bearbeiten statt Attribute, und Attributwerte werden möglicherweise nicht sofort als Elementinhalt angezeigt. Das würde die Verwendung von Elementen bevorzugen, wenn menschliche Autoren damit zu tun bekommen.
- Ein XML-Interpreter kann den Inhalt eines Attributwerts einfacher prüfen als den Inhalt eines Elements. Er kann den Attributwert nicht auf viel überprüfen, aber den Elementwert überhaupt nicht.
- Die Attribute von Elementen separater XML-Dokumente können ganz einfach kombiniert werden. Es kann extrem schwierig sein, Elemente zu kombinieren; deshalb sind Attribute in einer verteilten Umgebung möglicherweise zu bevorzugen.
- In Umgebungen, wo mehrere Autoren im Team zusammenarbeiten, ist es basierend auf der Elementstruktur einfacher, XML-Dokumente in Fragmente aufzuteilen als mit Hilfe von Attributwerten.
- Sinnvoll benannte Elemente können dem Autor die Arbeit vereinfachen und sind weniger verwirrend. Wenn beispielsweise jemand ein Element verwenden will, das eine Liste aufnehmen soll, dann ist es offensichtlich, daß er ein `list`-Element braucht. Die Auswahl des richtigen Attributs muß dagegen nicht immer offensichtlich sein und ist dann auch nicht deutlich sichtbar (man sieht häufig so kryptische Attributwerte wie `ordered` und `unordered`, die von HTML kommen). Werden statt dessen Container-Elemente benutzt, wie etwa `list.numbered` und `list.bulleted`, wird dem Autor das Leben stark vereinfacht.

Hier erkennen Sie vielleicht einen allgemeinen Trend. Wenn Menschen mit XML-Dokumenten arbeiten sollen, dann ist es besser, Elemente anstelle von Attributen zu verwenden.

Sparen Sie sich Schreibarbeit durch Parameter-Entities

In Kapitel 9 werden Sie mehr über Entities erfahren. Hier geht es nur um einen bestimmten Entity-Typ, das Parameter-Entity, das in XML-DTDs extrem praktisch sein kann.

Parameter-Entities sind im Grunde dasselbe wie die Zeichen-Entities, die Sie bereits kennengelernt haben. Auch sie verhalten sich wie Makros und können als Abkürzungen für Zeichenketten genutzt werden. Während Zeichen-Entities jedoch als Abkürzungen für Zeichenketten verwendet werden, dienen Parameter-Entities als Abkürzungen für Markup-Deklarationen und Deklarationsabschnitte. Weil sie mit Markup-Deklarationen zu tun haben, werden sie offensichtlich nur in DTDs verwendet. Und es gibt sogar noch mehr Einschränkungen:

- In der internen DTD-Untermenge können Parameter-Entities nur zwischen anderen Markup-Deklarationen benutzt werden.
- In der externen DTD-Untermenge können Parameter-Entities sowohl zwischen als auch innerhalb anderer Markup-Deklarationen genutzt werden.

  Um sicherzustellen, daß es keine Verwechslung zwischen Parameter-Entities und Zeichen-Entities gibt, unterscheidet sich die Syntax für die Deklaration und den Verweis auf Parameter-Entities wesentlich von der Syntax für Zeichen-Entities. Ein Parameter-Entity verwendet ein Prozentzeichen (%), wie im folgenden Beispiel gezeigt:

```
<!ENTITY % heading "H1 ¦ H2 ¦ H3 ¦ H4 ¦ H5 ¦ H6" >
<!ENTITY % body.content "%heading ¦ %text ¦ %block ¦ ADDRESS)*">
<!ELEMENT BODY %body.content>
```

Das Parameter-Entity `heading` erspart Ihnen viel Schreibarbeit (beachten Sie, daß Sie diesen Trick nur in der externen DTD-Untermenge verwenden können).

Sie können ein Parameter-Entity auch nutzen, um Ihre DTD-Deklarationen ein bißchen verständlicher zu machen, wie in Listing 7.2 gezeigt.

Listing 7.2: Vereinfachung einer DTD mit einem Parameter-Entity

```
1:<!ENTITY % color "CDATA" >
2:
3:<!ENTITY % body-color-attrs "
4:        bgcolor %color; #IMPLIED
```

```
5:        text    %color; #IMPLIED
6:        link    %color; #IMPLIED
7:        vlink   %color; #IMPLIED
8:        alink   %color; #IMPLIED
9:        ">
10:
11:<!ELEMENT BODY  %body.content>
12:<!ATTLIST BODY
13:          background %URL #IMPLIED
14:          %body-color-attrs; >
```

Beide hier aufgeführten Beispiele zeigen die Verwendung interner Parameter-Entities. Für den XML-Interpreter ist keine externe Datei erforderlich, um den Inhalt der Entities zu ermitteln. Außerdem ist es möglich, externe Parameter zu verwenden, wie Sie gleich sehen werden, die dann für ganz andere Aufgaben genutzt werden.

Modulare DTDs

Als ich früher in diesem Kapitel über die Größe und Komplexität von DTDs gesprochen habe, habe ich auch die Möglichkeit erwähnt, die DTD in Module aufzusplitten. Das ist ebenfalls eine Nutzungsmöglichkeit für externe Parameter-Entities. Der folgende DTD-Ausschnitt beispielsweise deklariert mehrere Zeichen-Entities, die in einer externen Datei enthalten sind, und greift dann unmittelbar darauf zu (das ist eine so gebräuchliche Nutzung von Parameter-Entities, daß es sehr viele öffentliche Bezeichner gibt, die die meisten der weniger häufig gebrauchten Zeichen abdecken):

```
<!ENTITY % ISOnum PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special
Graphic//EN">
%ISOnum;
```

Externe Parameter-Entities sind sehr praktisch, wenn Sie die Ergebnisse einer Arbeit wiederverwenden wollen, die Sie bei der Entwicklung anderer DTDs geleistet haben. Mit den Jahren haben sich beispielsweise die Deklarationen der Elemente in einer Tabelle um ein Modell herum standardisiert, das sogenannte CALS-Tabellen-Modell (benannt nach der Computer-Assisted-Logistic-Support-Initiative des amerikanischen Verteidigungsministeriums, unter der viele frühe SGML-Entwicklungen gegründet wurden). Es ist schwierig, dieses Modell zu verbessern (die Tabellen sind geradezu Standard), deshalb wird es häufig als allgemeines DTD-Fragment bereitgestellt, auf das alle anderen DTDs verweisen können, wenn sie es brauchen (es ist ebenfalls Teil der offiziellen HTML-DTDs):

```
<?xml version="1.0" standalone="no"?>
      <!DOCTYPE book SYSTEM "book.dtd" [
         <!ENTITY % calstable SYSTEM "cals.dtd">
         %calstable;
]>
```

Beachten Sie, daß diese Deklaration des externen Parameter-Entity und der Verweis darauf sich in der internen DTD-Untermenge befinden. Beachten Sie außerdem, daß ich explizit angegeben habe, daß das XML-Dokument kein Standalone-Dokument ist.

Wenn Sie ein XML-Dokument nicht auswerten wollen, müssen Sie immer daran denken, was mit externen Paremeter-Entities, die Markup-Deklarationen enthalten, schiefgehen könnte.

Listing 7.3 zeigt ein externes Parameter-Entity, das in einer internen DTD-Untermenge verwendet wird, um einige Deklarationen aus einer externen Datei aufzunehmen.

Listing 7.3: Ein gefährliches externes Parameter-Entity

```
1: <?xml version="1.0" standalone="yes"?>
2: <!DOCTYPE menu [
3:   <!ELEMENT menu (#PCDATA, front, meals+, back)* >
4:   <!ATTLIST menu title CDATA "Carte Blanche">
5:
6:   <!ENTITY % entrees SYSTEM "entrees.xml">
7:   %entrees;
8:   <!ATTLIST menu desserts CDATA "Sweet Temptations">
9:  ]>
10: <menu><front>Sunday, July 11, 1998</front> ... </menu>
```

Wenn dieses XML-Dokument geparst, aber nicht ausgewertet wird, ist der Wert des `title`-Attributs für den Menüeintrag gleich `Carte Blanche`. Der XML-Interpreter sieht die Deklaration des `title`-Attributs (Zeile 4), auch wenn diese sich innerhalb der internen DTD-Untermenge befindet (Zeilen 2 bis 9). Weil er XML-Dokumente nicht auswertet (das Dokument behauptet in Zeile 1, daß es sich um ein Standalone-Dokument handelt), kann der XML-Interpreter keine externen Entity-Verweise verarbeiten. Der Wert des `desserts`-Attributs bleibt unbekannt.

Der Vorgabewert (`Sweet Temptations`) wurde deklariert, aber der XML-Interpreter darf ihn nicht verwenden. Vielleicht hört es sich seltsam an, aber was ist in der Datei `entrees.xml` enthalten, auf die das externe Parameter-Entity `entrees` verweist? Ich weiß es nicht, aber es ist auch nicht besonders wichtig, weil der XML-Interpreter es auch nicht wissen darf. Nehmen wir jedoch an, daß die Datei `entrees.xml` Elementdeklarationen enthält; der Entity-Verweis befindet sich innerhalb der internen DTD-Untermenge; es wäre also völlig in Ordnung, darauf zu verweisen. Nehmen wir außerdem an, daß die Datei die folgende Attributdeklaration enthält:

```
<!ATTLIST menu desserts CDATA "Bitter Experiences">
```

Das würde bedeuten, daß das Attribut `desserts` nicht den Wert `Sweet Temptations` haben könnte.

Seien Sie also vorsichtig mit externen Parameter-Entities in der internen DTD-Untermenge, und achten Sie auf die möglichen Verwirrungen, die sie verursachen könnte, wenn der Empfänger nicht auswertet (etwas, worüber Sie kaum Kontrolle haben). Die einzige Möglichkeit, sich davor zu schützen, ist, immer eine Standalone-Dokumentdeklaration zu verwenden.

Die Aufgabe dieser Deklaration ist letztlich, dem Empfänger mitzuteilen, daß die DTD das Dokument ändern könnte und daß die DTD ermittelt werden sollte, wenn der Empfänger sicher sein will, daß er dasselbe sieht wie die Anwendung, in der das Dokument erzeugt wurde. Durch Parsen der DTD werden externe Entity-Verweise dereferenziert, die externe Deklaration würde also gefunden. Die Deklaration hat zwar keine Auswirkung auf den XML-Interpreter, aber durch Änderung des Werts der Standalone-Dokumentdeklaration in »no« wird der Empfänger darüber informiert, daß eine Auswertung erforderlich ist, wenn das Dokument so angezeigt werden soll, wie beabsichtigt.

Bedingtes Markup

In Kapitel 6 haben ich beim DTD-Entwurfsschritt zur Beschränkung des Modells gesagt, daß Sie immer in Betracht ziehen sollten, zwei Versionen derselben DTD anzulegen. Sie könnten während des Schreibens eine eher lockere DTD verwenden, wenn beispielsweise noch nicht der gesamte Inhalt eingetragen wurde, Sie aber das Dokument dennoch auswerten möchten, und eine andere, strengere DTD für das fertige XML-Dokument. Für genau diesen Zweck gibt es die sogenannten *bedingten Abschnitte* der DTD und Parameter-Entities. (Mehr über bedingte Abschnitte einer DTD finden Sie in Kapitel 9, wo es um Entity-Deklarationen geht.)

Dieser Trick läuft darauf hinaus, daß man die beiden Deklarationsvarianten der DTD in Blöcke verpackt und jeden Block mit einem Verweis auf ein Parameter-Entity versieht, wie im folgenden Abschnitt gezeigt:

```
<![%AuthoringSwitch;[
      <!ENTITY % body "chapter, intro?, section*">
]]>
<![%FinalSwitch;[
      <!ENTITY % body "chapter, intro, section+">
]]>
```

Die Parameter-Entities `AuthoringSwitch` und `FinalSwitch` können dann als die Schlüsselwörter `INCLUDE` oder `IGNORE` deklariert werden, abhängig davon, welche Version dieses Entity Sie verwenden möchten. Für die erste, weniger spezifizierte Version würden Sie die Deklarationen vor den bedingten Abschnitten wie folgt einfügen:

```
<!ENTITY % AuthoringSwitch "INCLUDE"
<!ENTITY % FinalSwitch "IGNORE">

<![%AuthoringSwitch; [
      <!ENTITY % body "chapter, intro?, section*">
]]>
<![%FinalSwitch; [
      <!ENTITY % body "chapter, intro, section+">
]]>
```

Und für die zweite, strengere Version würden Sie die Deklarationen vor den bedingten Abschnitten wie folgt einfügen:

```
<!ENTITY % AuthoringSwitch "IGNORE">
<!ENTITY % FinalSwitch "INCLUDE"

<![%AuthoringSwitch; [
      <!ENTITY % body "chapter, intro?, section*">
]]>
<![%FinalSwitch; [
      <!ENTITY % body "chapter, intro, section+">
]]>
```

Bedingte Abschnitte in DTD scheinen eine triviale Funktion zu haben, wobei nur ein kleiner Bereich betroffen ist (wie in dem Beispiel), aber Sie können Paare mit Parameter-Entities verwenden, um viele separate Abschnitte einer DTD zu steuern. Durch Änderung der Werte nur zwei dieser Parameter-Entities könnten Sie die gesamte DTD radikal ändern.

Optionale Inhaltsmodelle und Mehrdeutigkeiten

Wenn Sie eine DTD betrachten, mit der Sie arbeiten, achten Sie auf die Angaben zum Elementvorkommen. Wenn es in einem Inhaltsmodell sehr viele optionale Elemente gibt, dann stimmt mit dem Modell vielleicht etwas nicht.

Ein Beispiel dafür stammt aus der SGML-Welt, wo Bücher übliche Objekte für Markup sind. Betrachten Sie den hinteren Teil eines Buchs. Ich lege eine XML-DTD für ein Buch ein und erzeuge das Container-Element `BACK`, um es einfacher zu machen, die Teile zusammen zu verarbeiten. Mein `BACK`-Element besteht, wie Bücher es häufig tun, aus mehreren Anhängen (auch einen mit der Nummer Null), gefolgt von einem optionalen Glossar und von einem optionalen Index. Wenn Sie ein Inhaltsmodell in natürlicher Sprache beschreiben können, ist es relativ einfach zu modellieren:

```
<!ELEMENT BACK (appendix*, glossary?, index?)
```

Das Inhaltsmodell sieht möglicherweise ganz gut aus, aber ich habe jetzt ein Problem. Alles ist optional, so daß auch ein leeres `BACK`-Element angelegt werden kann, was sicher nicht der Sinn der Sache war.

Ein Ausweg wäre, das Inhaltsmodell in innere Gruppen aufzusplitten, wobei jede Gruppe explizit eine der Möglichkeiten beschreibt:

```
<!ELEMENT BACK ( (appendix+, glossary?, index?) ¦
         (appendix*, glossary, index?) ¦
         (appendix*, glossary?, index) )>
```

Damit scheint alles abgedeckt zu sein. Wir wollen es prüfen:

- Gruppe 1: Ich kann einen oder mehrere Anhänge erzeugen, gefolgt von einem optionalen Glossar und dann einem optionalen Index.
- Gruppe 2: Ich kann null oder mehr Anhänge erzeugen, denen ein Glossar folgen muß und optional ein Index.
- Gruppe 3: Ich kann null oder mehr Anhänge erzeugen, gefolgt von einem optionalem Glossar und einem zwingend vorgeschriebenen Index.

  Das scheint vollständig zu sein – und es sind keine leeren `BACK`-Elemente erlaubt – , aber jetzt habe ich noch ein größeres Problem. Dieses Markup funktioniert nicht!

XML-Interpreter sind nicht besonders intelligent. Sie können sich nicht merken, was sie gesehen haben, können nicht vorauslesen, was als nächstes kommt, und dann zurückgehen und bestätigen, daß Ihr Markup zumindest mit dem Inhaltsmodell übereinstimmt. Die Prozessoren können nur ein Stück Markup lesen, seine Gültigkeit prüfen, das nächste Stück lesen, es

prüfen usw. Aufgrund dieser Beschränkungen der Prozessoren müssen Elementinhaltsmodelle absolut klar sein und nur eine einzige Interpretationsmöglichkeit zulassen.

Wenn der XML-Interpreter ein `appendix`-Element sieht, kann er nicht feststellen, was als nächstes passieren soll – soll es ein `glossary`-Element geben oder nicht? Dieses Inhaltsmodell ist mehrdeutig.

Dieses Problem kann man umgehen, indem man einen sogenannten Wasserfall-Ansatz verfolgt. Dabei nehmen Sie nacheinander jedes Element und erarbeiten nur die neuen Fälle, die Sie abdecken wollen. Für das erste Element müßten Sie dabei sehr viele Möglichkeiten in Betracht ziehen, für das zweite schon weniger und für das dritte noch weniger, bis schließlich das letzte Elemente erreicht ist, für das es nur noch ganz wenige Möglichkeiten gibt. Diesen Ansatz wollen wir hier anwenden und schrittweise verarbeiten. Dabei betrachten wir noch einmal die drei Gruppen, die wir zuvor definiert haben:

- Gruppe 1: Ich kann einen oder mehrere Anhänge erzeugen, gefolgt von einem optionalen Glossar und dann einem optionalen Index. Die erste Gruppe scheint in Ordnung zu sein.
- Gruppe 2: Ich kann null Anhänge erzeugen (ich habe die Möglichkeit »einen oder mehrere« bereits abgedeckt), denen ein Glossar folgen muß, gefolgt von einem optionalen Index.
- Gruppe 3: Ich kann null Anhänge und kein Glossar erzeugen, aber es muß einen Index geben.

Das ist viel besser, und wenn wir es in die DTD schreiben, haben wir die Antwort:

```
<!ELEMENT BACK ( (appendix*, glossary?, index?)  ¦
      (glossary, index?) ¦
      (index) )>
```

Konflikte mit Namensräumen vermeiden

XML wurde in Hinblick auf das Internet entwickelt, das steht fest. XML besitzt außerdem (wie Sie später noch erfahren werden) extrem leistungsfähige Mechanismen für die Verknüpfung von Dokumenten, die viel weiter gehen, als HTML oder sogar SGML bieten kann. Die Verknüpfungsfunktionen in XML erlauben Ihnen, noch einen Schritt weiter zu gehen, als nur auf ein anderes Dokument zu »zeigen«: Sie erlauben Ihnen, das verknüpfte Dokument (das Verknüpfungsziel) in das aktuelle Dokument zu ziehen und es so einzubinden, als wäre es physischer Bestandteil des aktuellen Dokuments. Diese Art der Verknüpfung wird auch als *Transklusion* bezeichnet.

Transklusion ist ein zusammengesetztes Wort, gebildet aus *Transversal* und *Inklusion*. Es beschreibt die Aktivität, einem Hypertext-Link von seiner Quelle zu seinem Ziel zu folgen und das Ziel dann an die Stelle des Verweises zu kopieren, als wäre es physisch dort eingefügt worden (ähnlich wie Grafiken als Verweis in HTML-Dateien eingebunden werden).

Angenommen, das Verknüpfungsziel ist ebenfalls ein XML-Dokument. Das muß nicht unbedingt ein Problem sein, weil XML feste Regeln für solche Konflikte vorgibt:

- Wenn die Dokumente unter Verwendung derselben DTD erzeugt wurden, mit denselben Elementen und denselben Elementinhaltsmodellen, handelt es sich sehr wahrscheinlich überhaupt nicht um ein Problem.
- Wenn sie dieselben Elemente enthalten, die Elemente aber unterschiedliche Attribute haben, werden die Attribute einfach gemischt, und die zusammengesetzten Dokumentelemente erhalten alle Attribute.
- Wenn sie unterschiedliche Elementdeklarationen oder dieselben Attribute mit unterschiedlichen Werten haben, wird es komplizierter. Wenn Sie ein XML-Dokument auswerten, kann ein Element nur ein einziges Mal deklariert werden.

Um dieses Problem zu umgehen, wenn irgendwo dieselben Namen verwendet werden, wird für jedes Schema (das gilt auch für andere Schemata als für DTDs) ein privater »Namensraum« angenommen, in dem alle Deklarationen eindeutig sind und die dort ihre eigene Bedeutung haben. Zwischen der XML-Deklaration und der Dokumenttypdeklaration fügen Sie eine Deklaration ein (eine XML-Verarbeitungsanweisung) wie im folgenden Codeabschnitt gezeigt, die die Namensräume als zu den Schemata gehörig kennzeichnet:

```
<?xml version="1.0" standalone="no">
<?xml:namespace name="http://www.synopsys.com/"

                        href="http://www.synopsys.com/~north/xml/version1.dtd"
                        as="v1"?>
```

Der Wert des `name`-Attributs bezeichnet den Eigentümer des Namensraums, das `href`-Attribut bezeichnet das eigentliche Schema, und das `as`-Attribut deklariert ein Präfix, das vor jedem Element und vor jedem Attribut dieses Namensraums verwendet wird. Wenn ich also zwei Attribute habe, die in unterschiedlichen Namensräumen unterschiedliche Bedeutungen haben, muß ich nicht eines von beiden verwerfen, und ich erhalte bei der Auswertung keine fatalen Fehler, weil Elemente zweimal deklariert werden. Statt dessen kann ich sie unterscheiden und erkennen, daß sie aus unterschiedlichen Namensräumen stammen, und die von ihnen dargestellte Information beibehalten, wie im folgenden Codeausschnitt gezeigt:

```
<object v1.type:v2.sort="shirt"
<v1:size>10</size>
<v2:size>XXL</size>
</object>
```

Beachten Sie, daß das Präfix, mit dem der Namensraum bezeichnet wird, lokal auftritt. Das bedeutet nicht nur, daß Sie es beliebig benennen können, sondern auch, daß es keine der unvermeidlichen Probleme gibt, die entstehen, wenn Sie versuchen, die für die unendliche Anzahl von Namensräumen verwendeten Namen zu zentralisieren.

Bei der Realisierung der Namensräume handelt es sich um einen Vorschlag, der noch geändert werden kann. Sie wurden zwar schon in verschiedenen XML-Anwendungen übernommen (auch in XSL), aber einige der Implementierungen entsprechen nicht dem Standard, und es kann noch hitzige Diskussionen geben, bevor ein geeigneter Kompromiß gefunden wird.

Ein Testfall

Bisher war dieses Kapitel eher theoretisch. Um zu vervollständigen, was Sie über die DTD-Entwicklung gelernt haben, wollen wir eine praktische Demonstration betrachten.

Der begrenzte Platz und die beschränkte Zeit erlauben nur wenige Details (für SGML-DTDs ist es ganz normal, daß ihre Entwicklung mehrere Jahre dauert und daß mehrere hundert Leute daran beteiligt sind). Aber genau so sollte es hier ja auch sein. Schließlich ist XML viel besser für schnelle, schlanke Applikationen geeignet.

Wir wollen etwas Einfaches betrachten, etwa mehrere Adreßdatensätze. (Ich bin fast versucht, das Ganze als »Adreßdatenbank« zu bezeichnen, was es nicht ist, aber diese Art der datenbankähnlichen Information ist hervorragend für XML-basierte Anwendungen geeignet.) Listing 7.4 zeigt einen kleinen Ausschnitt aus den Rohdaten.

Listing 7.4: Die Rohdaten für eine XML-Anwendung

```
1:  Fred Bloggs MISTC
2:  22 Chancery Lane
3:  London SW17 4QP
4:  England
5:  44 1 800 3451
6:  44 12 446 3382
7:  fbloggs@hk.co.uk
8:
9:  Dr. Jon F. Spencer
10: El Camino Real 44621
11: Sunnyvale
12: 95144 California
13: USA
14: 1 650 445 1273
15: 1 405 227 1877
16: jdspencr@hiflier.com
```

Die in Listing 7.4 gezeigten Rohdaten bestehen aus zwei unterschiedlichen Arten von Adreßdetails. Daran erkennen Sie einige der Probleme, denen Sie irgendwann vielleicht gegenüberstehen. In Großbritannien beispielsweise kommt die Postleitzahl nach dem Namen, aber sie kann auch vor dem Namen stehen, wie in dem Datensatz für die USA gezeigt. In vielen europäischen Ländern erscheint sie vor dem Namen (wie in dem niederländischen Beispiel mit »Prof. Mr. Dr. Ing. D.H.J. Heyden-Loods«). Beachten Sie auch den Unterschied in der Positionierung und der Formatierung der Postleitzahl (verschiedene Länder verwenden unterschiedliche Schemata) und der Telefon- und Faxnummern (auch hier verwenden verschiedene Länder unterschiedliche Schemata).

Glücklicherweise kann die Reihenfolge, in der diese Einträge in einer gerenderten Ausgabe erscheinen, einem Stylesheet überlassen werden. Wir müssen hier nur die Informationen kennzeichnen und nicht angeben, wie diese letztlich dargestellt werden sollen. Dieser Aspekt kann jedoch nicht völlig ignoriert werden, weil es mit Sicherheit Informationen gibt, die Sie in Elementen kennzeichnen wollen, um einfach nur eine bestimmte Darstellung dafür zu veranlassen.

Im ersten Schritt abstrahieren Sie den Informationsinhalt und kennzeichnen ihn als Elemente. Dabei müssen Sie vorausschauen und Möglichkeiten in Betracht ziehen, die in den Testdaten vielleicht nicht auftreten (beispielsweise Adressen, die aus mehreren Zeilen bestehen als den in Listing 7.4 gezeigten). Listing 7.5 zeigt meinen Versuch, die Daten als Elementhierarchie zu formalisieren.

Listing 7.5: Die formale Elementstruktur

```
1: People
2:    Person
3:        Name
4:            FirstName
5:            MiddleName
6:            FamilyName
7:            Title
8:        Address
9:           Street1
10:          Street2
11:          City
12:          State
13:          Country
14:          ZipCode
15:       PhoneNumber
16:       FaxNumber
17:       Email
18:       Notes
```

Beachten Sie, daß ich ein Element für den mittleren Namen und eines für eine Straßenadresse mit zwei Zeilen eingefügt habe, obwohl das in keinem der beiden zuvor gezeigten Beispiele erforderlich war. Beachten Sie außerdem, daß ich die Telefon- und Faxnummern in separate Nummern für die Ländervorwahl, die Vorwahl usw. unterteilt habe. Das sollten Sie auch für Ihre eigenen Datensätze so handhaben, insbesondere wenn Sie manchmal nur Adressen mit einer bestimmten Vorwahl auswählen wollen. In dieser Anwendung sind wir nur an den vollständigen Nummern interessiert, durch das Hinzufügen zusätzlicher Elemente können wir also nichts gewinnen.

Nachdem Sie den grundlegenden Elementbaum vorliegen haben, können Sie anfangen, das Ganze in DTD-Syntax zu übertragen. Beginnen Sie mit den obersten Elementen:

```
1: <!DOCTYPE People [
2:
3:   <!ELEMENT People  (Person)>
4:
5:   <!ELEMENT Person  (Name, Address, PhoneNumber,
6:                      FaxNumber, Email, Notes)>
7:
8:   <!ELEMENT Name    (FirstName, MiddleName, FamilyName,
```

```
9:                      Title)>
10:
11:  <!ELEMENT Address (Street1, Street2, City, State,
12:                      Country, ZipCode)>
```

Nachdem Sie die Elemente der obersten Ebene aussortiert haben, können Sie mit der zweiten Ebene fortfahren und dann mit den untersten Elementen (Blättern):

```
13: <!ELEMENT FirstName   (#PCDATA)>
14: <!ELEMENT MiddleName  (#PCDATA)>
15: <!ELEMENT FamilyName  (#PCDATA)>
16: <!ELEMENT Title       (#PCDATA)>
17: <!ELEMENT Street1     (#PCDATA)>
18: <!ELEMENT Street2     (#PCDATA)>
19: <!ELEMENT City        (#PCDATA)>
20: <!ELEMENT State       (#PCDATA)>
21: <!ELEMENT Country     (#PCDATA)>
22: <!ELEMENT ZipCode     (#PCDATA)>
23: <!ELEMENT PhoneNumber (#PCDATA)>
24: <!ELEMENT FaxNumber   (#PCDATA)>
25: <!ELEMENT Email       (#PCDATA)>
26: <!ELEMENT Notes       (#PCDATA)>
27: ]>
```

Widerstehen Sie der Versuchung, den genauen Elementinhalt anzugeben, es sei denn, das ist absolut notwendig, und selbst dann sollten Sie es nur tun, wenn Sie sicher sind, daß die Daten immer diesen Typ haben. Die Angabe einer Telefonnummer als numerische Daten (`NMTOKENS`) kann nur Probleme für Sie schaffen, wenn Sie auf Nummern treffen, die auch nicht-numerische Daten (beispielsweise Trennstriche) enthalten.

Ich habe mich in dieser DTD für die Sicherheit entschieden. Alle Sheet-Elemente sind einfach nur vom Typ PCDATA (Text), weil ich erstens nicht sicher bin, welchen Inhalt die Elemente annehmen, und zweitens durch die Einschränkung des Dateninhalts nichts gewinne.

Wir sind noch nicht ganz fertig. Sie müssen noch die Angaben für das Elementvorkommen eintragen, wie in Listing 7.6 gezeigt.

Listing 7.6: Die vollständige DTD

```
1: <!DOCTYPE People [
2:
3:   <!ELEMENT People  (Person)+>
4:
```

```
5:   <!ELEMENT Person  (Name, Address?, PhoneNumber?,
6:                      FaxNumber?, Email?, Notes?)>
7:
8:   <!ELEMENT Name    (FirstName?, MiddleName?, FamilyName,
9:                      Title?)>
10:
11:  <!ELEMENT Address (Street1?, Street2?, City?, State?,
12:                     Country?, ZipCode?)>
13:
14:  <!ENTITY % Data "(FirstName, MiddleName,
15:                   FamilyName, Title, Street1, Street2,
16:                   City, State, Country, ZipCode,
17:                   PhoneNumber, FaxNumber, Email, Notes)">
18:
19:  <!ELEMENT %Data; (#PCDATA)>
```

Fast jedes Element in der in Listing 7.6 gezeigten vollständigen DTD kann nur einmal auftreten oder ganz weggelassen werden. Diese Elemente werden mit einem Fragezeichen ? gekennzeichnet. Es gibt einige Ausnahmen. Beispielsweise muß es mindestens eine `Person` geben, und es kann beliebig viele davon geben; das `Person`-Element wird deshalb mit einem Plus + gekennzeichnet. Eine Person muß einen Namen haben, sonst wäre der Datensatz relativ sinnlos (obwohl alle anderen Daten optional sind). Deshalb habe ich das `Name`-Element zu einem zwingend erforderlichen Teil des `Person`-Elements gemacht, und das `FamilyName`-Element darf nur ein einziges Mal im `Name`-Element vorkommen (`Name` und `FamilyName` haben deshalb keine Angabe zum Vorkommen).

Wir sind fast fertig. Im letzten Schritt betrachten wir das Modell noch einmal und legen die Attribute fest. In diesem Beispiel habe ich eine Auswahl offengelassen. Ich habe ein `Notes`-Element eingefügt, das mir erlaubt, beschreibende Informationen einzufügen, aber ich kann nicht sehr viel mit den Dingen tun, die in dem Element enthalten sind, denn was die XML-Anwendung angeht, enthält es einfach Text. Sie könnten auch ein Attribut einfügen, das beschreibt, um welche Art Person es sich handelt. (Das ist ein Beispiel, wo Ihre Entscheidungen nur dadurch bestimmt werden, was Sie letztlich mit den Daten vorhaben.) Beispielsweise könnten Sie folgendes deklarieren:

```
1:   <!ELEMENT Person  (Name, Address?, PhoneNumber?,
2:                      FaxNumber?, Email?, Notes?)>
3:
4:   <!ATTLIST Person Type (Business ¦ Personal) "Business">
5:
6:   <!ATTLIST Title Position (Before ¦ After) "Before">
```

Hier habe ich ein `Type`-Attribut für eine Person deklariert. Weil der Inhalt des Attributs ein String ist, ist es standardmäßig `CDATA`, und ich muß den Attributtyp nicht deklarieren. Ich kann jedoch `Business` als den Vorgabewert deklarieren, d.h. ich muß nur einen Wert in den XML-Daten für alle Leute mit persönlichen Typeinträgen angeben.

Wo die Attribute ihren Sinn haben, ist bei der Lösung von Darstellungs- und Rendering-Problemen, etwa das Problem mit der Positionierung des Titels, das ich bereits erwähnt habe. Hier habe ich ein `Position`-Attribut für den Titel angegeben, mit zwei Optionen, `Before` und `After`. Der `Before`-Wert ist der Vorgabewert; ich muß also das Attribut nur dann explizit angeben, wenn der `Title` hinter dem Namen erscheinen soll. Die Anwendung und das Stylesheet steuern die gerenderte Position.

Bisher habe ich die DTD so geschrieben, daß sie in einem Testdokument als interne DTD-Untermenge verwendet werden kann. Ich muß nur noch die erste und die letzte Zeile entfernen (`<!DOCTYPE People [` und `]>`) und die DTD in einer separaten Datei ablegen.

Sie werden mehr als nur ein paar DTDs entwickeln; wählen Sie also Dateinamen, die auf den Inhalt schließen lassen. Im allgemeinen ist es ausreichend, der Datei denselben Namen wie dem Wurzelelement zu geben, aber Sie können auch spezifischer werden (wie beispielsweise `DOC_SHORT` und `DOC_FULL`). Legen Sie sich als erstes eine Strategie dafür zurecht; wenn Sie irgendwann alle Verweise auf eine DTD in mehreren Dokumenten ändern müssen, dann kann das eine Qual sein, wenn Sie eigentlich nur einen Dateinamen ändern wollten.

Jetzt sollten Sie bereit sein, das XML mit Markup zu versehen und für die DTD auszuwerten. Listing 7.7 zeigt meine mit Markup gekennzeichneten Daten.

Listing 7.7: Die mit Markup gekennzeichneten Daten für eine XML-Anwendung

```
1:  <?xml version="1.0" ?>
2:  <!DOCTYPE People SYSTEM "People.DTD">
3:
4:  <People>
5:
6:    <Person Type="Personal">
7:      <Name>
8:        <FirstName>Fred</FirstName>
9:        <FamilyName>Bloggs</FamilyName>
10:       <Title Position="After">MISTC</Title>
```

```
11:     </Name>
12:     <Address>
13:       <Street1>22 Chancery Lane</Street1>
14:       <City>London</City>
15:       <Country>England</Country>
16:       <ZipCode>SW17 4QP</ZipCode>
17:     </Address>
18:     <PhoneNumber>44 1 800 3451</PhoneNumber>
19:     <FaxNumber>44 12 446 3382</FaxNumber>
20:     <Email>fbloggs@hk.co.uk</Email>
21:   </Person>
22:
23:   <Person>
24:     <Name>
25:       <FirstName>Jon</FirstName>
26:       <MiddleName>Jefferson</MiddleName>
27:       <FamilyName>Spencer</FamilyName>
28:       <Title>Dr.</Title>
29:     </Name>
30:     <Address>
31:       <Street1>El Camino Real 44621</Street1>
32:       <City>Sunnyvale</City>
33:       <State>California</State>
34:       <Country>USA</Country>
35:       <ZipCode>915144</ZipCode>
36:     </Address>
37:     <PhoneNumber>1 650 445 1273</PhoneNumber>
38:     <FaxNumber>1 405 227 1877</FaxNumber>
39:     <Email>jdspencr@hiflier.com</Email>
40:   </Person>
41: </People>
```

Fertig! Sie haben eine XML-Instanz mit eigener DTD.

Zusammenfassung

Heute haben Sie sehr viel über DTDs und XML-Inhaltsmodellierung gelernt. Darüber hinaus haben Sie auch ein paar der komplexeren Tricks kennengelernt, die Ihnen das Leben sehr viel einfacher machen können. Sie haben selbst einige der Probleme gesehen, die auftreten können – und erfahren, wie Sie sich aus der Affäre ziehen können. Damit sollten Sie in der Lage sein, selbst komplexe CML-DTDs zu entwickeln, und Sie wissen auch, welche Werkzeuge Sie dazu am besten benutzen. In den Kapiteln 8 und 9 werden wir weiter auf einige der komplexeren Themen eingehen, die in diesem und dem letzten Kapitel angesprochen wurden.

F&A

F Warum sind Mehrdeutigkeiten so wichtig?

A *Mehrdeutige Inhaltsmodelle sind eine der häufigsten Ursachen für Probleme bei der DTD-Entwicklung, insbesondere für Anfänger. Außerdem sind sie schwer zu erkennen und zu korrigieren. Alles, was Sie jetzt darüber lernen, kann Ihnen später viel Kopfzerbrechen ersparen.*

F Wann sollte ich eine DTD in mehrere Komponenten zerlegen?

A *Es gibt keinen absoluten Richtwert; dabei geht es einfach darum, wie gut oder wie schlecht es Ihnen im Umgang mit einer riesigen Einheit geht. Größe und Komplexität sind jedoch nicht die einzigen Kriterien. Sie könnten eine DTD auch viel früher aufteilen, um einfach eine gewisse Modularität zu schaffen.*

F Ich kann modulare DTDs anlegen, kann ich dann auch modulare Dokumente verwenden?

A *Natürlich. Um ein XML-Dokument zu modularisieren, zerlegen Sie es einfach in Text-Entities und kombinieren sie unter Verwendung von Zeichen-Entities im Hauptdokument. Vergessen Sie nicht die Regeln zu den parallelen logischen und physischen Strukturen. Außerdem können Sie Transklusionen verwenden (Inklusionen durch Verweise), wie Sie in Kapitel 10 noch erfahren werden.*

Übungen

1. Entwerfen Sie eine DTD für eine grundlegende Web-Homepage (Sie können schummeln und in früheren Beispielen nachsehen).
2. Erweitern Sie mit Ihrer Homepage-DTD als Grundlage die DTD so, daß sie genutzt werden kann, um eine vollständige Web-Site abzudecken. Verwenden Sie dazu interne DTD-Untermengen und externe Parameterverweise.
3. Nehmen Sie eine beliebige DTD, und skizzieren Sie sie als Baumdiagramm (verwenden Sie beliebige Symbole). Jetzt setzen Sie die DTD-Elemente in eine Tabelle, wobei die erste Spalte das Wurzelelement enthält, die zweite die ihm untergeordneten Elemente usw. Vergleichen Sie die beiden Darstellungen. Jetzt sollten Sie davon überzeugt sein, wie wichtig die Modellierung einer DTD unter Verwendung einer klaren visuellen Darstellung ist.

2

Auf einen Blick

XML-Objekte: Entities

Woche 2

In Kapitel 7 haben Sie viel darüber gelernt, was Sie über Markups wissen müssen. Jetzt wollen wir einige dieser Dinge noch einmal betrachten und mehr technische Details zu den Besonderheiten von XML-Dokumenten dabei berücksichtigen – die Daten, die Sie dort ablegen. In diesem Kapitel lernen Sie die folgenden Dinge kennen:

- XML-Entities
- Notationen
- XML-Zeichensätze
- Unterschiedliche Zeichencodierungen

Entities

Ohne zu sehr in die Terminologie einzutauchen (insbesondere, weil sich dieses Kapitel in Hinblick auf einige der Begriffe gewisse Freiheiten erlaubt), bringt XML der objektorientierten Welt das Markup etwas näher, beispielsweise in der objektorientierten Programmierung. Das grundlegende Objekt in XML ist das Entity, egal ob es sich dabei um das eigentliche XML-Dokument-Entity handelt, die Elemente, die es enthält, oder die internen und externen Entities, auf die es verweist.

Die DTD selbst ist natürlich ebenfalls ein externes Entity, auf das das Dokument verweist (eigentlich ein spezieller Typ von Parameter-Entity), aber die Beziehung ist etwas komplizierter. Die DTD beschreibt eine Klasse von XML-Dokument-Entities, von denen das eigentliche XML-Dokument eine Instanz ist.

Die Entities sind in drei Arten unterteilt (außer dem XML-Dokument-Entity selbst, das in diesem Kontext nicht von weiterem Interesse ist): Zeichen-Entities, allgemeine Entities und Parameter-Entities. Allgemeine Entities können weiter unterteilt werden in interne Entities und externe Entities. Um das Ganze noch komplizierter zu machen, werden externe Entities unterteilt in geparste Entities (die Zeichendaten enthalten) und nichtgeparste Entities (die in der Regel binäre Daten enthalten). Schließlich ist es üblich, daß geparste allgemeine Entities als interne und externe Entities bezeichnet werden, und nichtgeparste allgemeine Entities einfach nur als binäre Entities.

Ich persönlich finde es einfacher, sich Text-Entities vorzustellen, die entweder intern oder extern sein können, und binäre Entities, die extern sein müssen. Diese minimale Klassifizierung wird durch die Art und Weise, wie Sie die Entities deklarieren, weiter unterstützt. Die Deklaration eines internen Text-Entity sieht wie folgt aus:

```
<!ENTITY name "ersatztext">
```

Die Deklaration eines externen Text-Entity sieht wie folgt aus (Sie lernen gleich mehr über die Syntax dieser Deklarationen):

```
<!ENTITY name SYSTEM "system.bezeichner">
<!ENTITY name PUBLIC "öffentlicher.bezeichner" "system.bezeichner">
```

Die Deklaration eines Zeichen-Entity (wobei es sich nur um einen Sonderfall eines internen Text-Entity handelt) sieht wie folgt aus:

```
<!ENTITY name "&#code;" >
```

Beim Verweis auf diese Entities sehen die Text- und Zeichen-Entity-Verweise wie folgt aus:

```
&name;
```

Nachdem Sie nun die grundlegenden Entity-Typen kennengelernt haben, wollen wir sie detaillierter betrachten.

Interne Entities

In internen Entities sind deren Werte bereits in der grundlegenden Definition enthalten, z.B.:

```
<!ENTITY intent "Ich bin ein internes Entity, meine Deklaration ist
in mir selbst enthalten">
```

Es gibt kein separates physisches Speicherobjekt (Datei), und der Inhalt des Entity wird durch die Deklaration vorgegeben. (Es könnte erforderlich sein, daß der XML-Interpreter alle Entity- und Zeichenverweise in dem Entity auflöst, um den korrekten Ersatztext zu erzeugen.)

Interne Entities werden geparst und dürfen keine Verweise auf sich selbst enthalten, weder direkt noch indirekt.

In Kapitel 4 haben Sie erfahren, wie man Entities in einer DTD deklariert, so daß Sie mehrere Vorkommen des Texts in eine Datei einfügen können, ohne ihn jedesmal neu tippen zu müssen. Ich werde den Mechanismus hier nicht im Detail beschreiben, aber wir wollen ein praktisches Beispiel betrachten:

Ich gebe in meine DTD den folgenden Code ein:

```
<!ENTITY schnell "ein überraschend langes und langweiliges Stück
Text, das ich wirklich nicht öfter eingeben möchte
als ein einziges mal.">
```

Wenn ich diesen Text in meinem XML-Dokument immer wieder brauche, kann ich an beliebiger Stelle darauf verweisen:

```
<p>Der erste Teil ist &schnell;,
der zweite Teil ist &schnell;, und
der dritte Teil ist &schnell;.</p>
```

Wenn diese Verweise in einem XML-Browser angezeigt werden, werden sie aufgelöst und durch den Entity-Text ersetzt, wie in Abbildung 8.1 gezeigt.

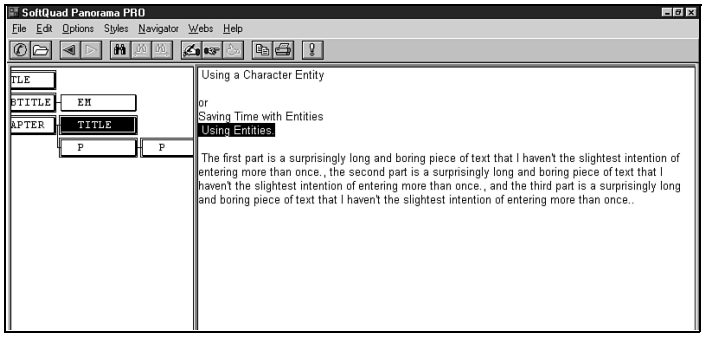


Abbildung 8.1: Wie ein Entity in einem XML-Browser aufgelöst wird.

Es wäre viel zufriedenstellender gewesen, einen echten XML-Browser in Abbildung 8.1 zu zeigen, aber keiner der aktuellen Web-Browser kann Entity-Verweise auflösen. Statt dessen ist hier der SGML-Browser Panorama von SoftQuad dargestellt. Glücklicherweise können Sie ganz einfach XML-Code anzeigen, der sich nicht zu sehr vom normalen SGML-Code unterscheidet (und der eine DTD hat). Eine Testversion von Panorama erhalten Sie unter `http://www.sq.com`.

Binäre Entities

Binäre Entities enthalten nichtgeparste Daten (Grafik, Sound usw.). Bei der Deklaration müssen sie durch die Notation gekennzeichnet werden. Die Notation muß in der DTD deklariert worden sein:

```
<!NOTATION notation.name PUBLIC "öffentlicher.bezeichner" "hilfs.anwendung">
<!ENTITY entity.name NDATA notation.name>
```

Der Verweis auf binäre Entities kann nur innerhalb des Werts eines Attributs erscheinen, das in der DTD mit dem Typ `ENTITY` oder `ENTITIES` deklariert wurde:

```
<!ELEMENT element.name EMPTY>
<!ATTLIST element.name
      attribut.name NDATA notation.name>
```

Und der Verweis auf dieses binäre Entity könnte im XML-Dokument wie folgt verwendet werden:

```
<element.name attribut.name="entity.name"/>
```

Listing 8.1 zeigt ein typisches Beispiel für einen Verweis auf ein binäres Entity in einer HTML-Datei. Derselbe Verweis könnte auch in einer XML-Datei verwendet werden.

Listing 8.1: Ein typischer Verweis auf ein binäres Entity

```
1:  <H4>
2:    <A HREF="http://www.w3.org/">
3:      <IMG alt="W3C" SRC="http://www.w3.org/
4:           pub/WWW/Icons/WWW/w3c_home.gif"
5:        WIDTH="72" HEIGHT=48>
6:    </A>Parsing Entities
7:  </H4>
```

Wenn dieser Code in einem Web-Browser angezeigt wird, sehen Sie die Grafik an der Stelle, wo der Verweis erfolgt ist, wie in Abbildung 8.2 gezeigt.

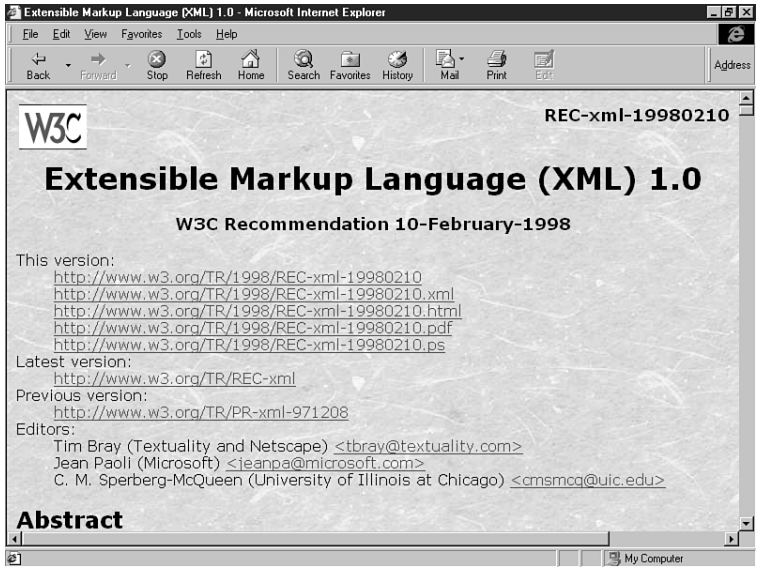


Abbildung 8.2: Wie ein binäres Entity (eine Grafikdatei) in einem Web-Browser aufgelöst wird.

Beachten Sie, daß der Verweis auf ein binäres Entity als Wert eines Elementattributs erscheint. Verweise auf Text-Entities können im Elementinhalt erscheinen, aber Verweise auf binäre (nichtgeparste) dürfen nur innerhalb von Attributen erscheinen.

Notationen

Notationen kennzeichnen durch einen Namen das Format nichtgeparster Entities (binäre Dateien wie etwa externe Grafikdateien), das Format der Elemente, die ein Notationsattribut tragen, oder die Hilfsanwendung (die Daten verarbeiten kann), für die eine Verarbeitungsanweisung vorgesehen ist.

Eine Notation muß in der DTD deklariert werden, bevor sie benutzt werden kann. Die absolute Mindestform einer Notationsdeklaration sieht wie folgt aus:

```
<!NOTATION Name SYSTEM "">
```

Dabei ist `Name` ein für Sie sprechender Name oder der registrierte öffentliche Bezeichner für ein bestimmtes Format.

Wenn Ihr System es unterstützt (wie etwa Microsoft Windows, vorausgesetzt, der Anwendung wurde diese Erweiterung hinzugefügt), können Sie auch eine Notationsdeklaration wie die folgende verwenden:

```
<!NOTATION GIF SYSTEM "GIF">
```

Diese Deklaration nutzt die Tatsache, daß das, was den XML-Interpreter betrifft, nicht auf dem System vorhanden sein muß, was Daten in dieser Notation interpretieren kann. Die Interpretation von Daten oder die Verarbeitung eines Fehlers, falls die Daten nicht interpretiert werden können, ist ausschließlich Sache der Applikation.

Wenn Sie wissen, daß sich auf dem System eine Applikation befindet, die in der Lage ist, Daten in einer bestimmten Notation zu verarbeiten, können Sie auf diese Applikation verweisen:

```
<!NOTATION TIFF SYSTEM "C:\Programme\PaintShop Pro 5\psp.exe">
```

Das funktioniert aber offensichtlich nur, wenn Sie den Namen und die Position der Anwendung kennen, die eine bestimmte Notation verarbeiten kann, und wenn niemand sie an eine andere Position verschiebt – was im Internet also nicht besonders sinnvoll ist.

Außerdem ist es möglich, eine existierende SGML-Funktion und einen öffentlichen Bezeichner zu verwenden. Es gibt zahlreiche registrierte öffentliche Bezeichner für Notationen, die fast alles abdecken, von der Programmiersprache C (`ISO/IEC 9899:1990//NOTATION` (Programmiersprachen – C)) bis zur Zeit selbst (`ISO 8601:1988//NOTATION` (Darstellung von Zeit und Datum)). Einige der gebräuchlichsten Notationen und ihre öffentlichen Bezeichner in SGML, wie sie in einer SGML-Notationsdeklaration verwendet werden, sehen Sie in Listing 8.2.

Listing 8.2: Einige standardmäßige SGML-Notationsdeklarationen

```
1: <!NOTATION JPEG PUBLIC "ISO/IEC 10918:1993//NOTATION
2:   Digital Compression and Coding of
3:   Continuous-tone Still Images (JPEG)//EN">
4:
5: <!NOTATION BMP PUBLIC
6:    "+//ISBN 0-7923-9432-1::Graphic Notation//NOTATION
7:   Microsoft Windows bitmap//EN">
8:
9: <!NOTATION CGM-CHAR PUBLIC
10:  "ISO 8632/2//NOTATION Character encoding//EN">
11:
12:  <!NOTATION CGM-BINARY PUBLIC
13:  "ISO 8632/3//NOTATION Binary encoding//EN">
14:
15: <!NOTATION CGM-CLEAR PUBLIC
16:  "ISO 8632/4//NOTATION Clear text encoding//EN">
17:
18: <!NOTATION FAX PUBLIC "-//USA-DOD//NOTATION
19:   CCITT Group 4 Facsimile Type 1 Untiled Raster//EN">
20:
21: <!NOTATION GIF87a PUBLIC "-//CompuServe//NOTATION
22:   Graphics Interchange Format 87a//EN">
23:
24: <!NOTATION GIF89a PUBLIC "-//CompuServe//NOTATION
25:   Graphics Interchange Format 89a//EN">
26:
27: <!NOTATION PCX PUBLIC
28: "+//ISBN 0-7923-9432-1::Graphic Notation//NOTATION
29:   ZSoft PCX bitmap//EN">
30:
31: <!NOTATION WMF PUBLIC
32: "+//ISBN 0-7923-9432-1::Graphic Notation//NOTATION
33:   Microsoft Windows Metafile//EN">
```

Es gibt keinen Grund, der dagegen spricht, diese SGML-Deklarationen in XML-Dokumenten zu verwenden, aber es gibt einige gute Gründe, die dafür sprechen. Die öffentlichen Bezeichner in SGML müssen jedoch mit Systembezeichnern kombiniert werden, um Ihnen das zu erlauben:

```
<!NOTATION GIF89a PUBLIC "-//CompuServe//NOTATION
      Graphics Interchange Format 89a//EN"
      'C:\Programme\lviewpro.exe'>
```

Beachten Sie, daß Sie einen Systembezeichner verwenden, aber das Schlüsselwort SYSTEM nicht brauchen.

Nachdem (und *nur* nachdem) Sie die Notation deklariert haben, können Sie diese über ihren Namen und das Schlüsselwort NDATA (Notationsdaten) in einer Entity-Deklaration verwenden:

```
<!ENTITY figure1 SYSTEM 'figure1.gif' NDATA BMP>
```

Sie können die Notation auch in einer der Attributdeklarationen für ein Element verwenden, indem Sie das Schlüsselwort NOTATION angeben:

```
<!ELEMENT IMG EMPTY >
<!ATTLIST IMG
      srcURL   #REQUIRED
      altCDATA #IMPLIED
      type  NOTATION (GIF ¦ JPEG ¦ BMP) "GIF" >
```

Beachten Sie bei der Verwendung aufzählender Notationen in einer Attributdeklaration, daß jede dieser Notationen in der DTD deklariert werden muß, bevor der XML-Interpreter diesen Teil der DTD erreicht. Wenn Sie beispielsweise in der internen DTD-Untermenge Notationen verwenden, müssen Sie die Notation auch in der internen DTD-Untermenge deklarieren und nicht in der externen DTD-Untermenge. (Wie Sie sich vielleicht erinnern, wird die interne DTD-Untermenge vor der externen DTD-Untermenge gelesen.)

Die Eingabe der öffentlichen Bezeichner für Notationen kann sehr schnell mühselig und fehleranfällig werden. Es ist sehr wahrscheinlich, daß Sie dieselben Notationen immer wieder verwenden. Es ist also sinnvoll, alle Notationsdeklarationen in einer einzigen Datei zu sammeln. Geben Sie der Datei einen sprechenden Namen, wie beispielsweise graphics.ent, so daß Sie bei einem Verweis darauf in einem XML-Dokument immer sofort erkennen, um was es sich handelt.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE chapter SYSTEM "chapter.dtd" [
<!ENTITY % myentities SYSTEM "mysymbols.ent">
%myentities;
]>
<chapter><number/> ... </chapter>
```

Durch die Verwendung eines sprechenden Namens für eine Datei, auf die in allen Ihren DTDs mit Hilfe einer externen Entity-Deklaration verwiesen wird, können Sie sich eine Menge unnötiger Arbeit ersparen. Warum einen sprechenden Dateinamen? Ganz einfach. Irgendwann bekommen Sie es wieder mit der DTD zu tun, vielleicht nach mehreren Wochen oder gar Monaten, und dann wird jeder zusätzliche Hinweis über ihre Aufgabe hilfreich sein.

Und allein die Verwaltung all Ihrer Notationsdeklarationen in einer separaten Datei bedeutet, daß Sie nur eine einzige Datei anpassen müssen, wenn sich etwas ändert, statt jede separate DTD anpassen zu müssen. Außerdem ist es weniger wahrscheinlich, daß Sie eine Deklaration vergessen, die Sie brauchen.

Kennzeichnung externer Entities

Interne Entities sind in sich abgeschlossen, d.h. ihre Definition ist in ihrer Deklaration enthalten. Externe Entities befinden sich an anderer Stelle, wie ihr Name schon sagt. Es gibt zwei Möglichkeiten, die Position des Inhalts eines externen Entity anzugeben: durch einen Systembezeichner oder durch einen öffentlichen Bezeichner.

Systembezeichner

Eine Systembezeichner kann entweder ein relativer Pfad zu einem Dateinamen sein (beispielsweise `..\..\graphics\home.gif`) oder ein absoluter Pfad zu einem Dateinamen (beispielsweise `c:\Programme\Lview\lview.exe`):

```
<!ENTITY my.file SYSTEM "c.\inetroot\filez\file1.xml">
```

Ein Systembezeichner kann auch eine URI (*Universal Resource Identifier*) sein. Die URI ist eine Erweiterung des URL-Systems (*Universal Resource Locator*), das für WWW-Adressen verwendet wird:

```
http://www.xs4all.nl/~sintac/books.htm
```

In diesem Fall ist `www.xs4all.nl` der Web-Server meines Service-Providers und `~sintac` ein Zeiger, den das Unix-System in mein Anmeldeverzeichnis übersetzt. Der Web-Server leitet den Web-Browser dann an das Web-Seiten-Verzeichnis weiter, das meinem Anmeldenamen zugeordnet ist. `books.htm` ist offensichtlich der Name der Datei. Eine URI ist eine Art URN (*Universal Resource Name*), eine Art Obermenge.

Die beiden sind – sofern über das Internet gesprochen wird – synonym zu verwenden, deshalb spreche ich hier immer von »URL« und bleibe dabei. (Alte Gewohnheiten wird man nicht so leicht los.) Die Syntax einer vollständigen URL sieht wie folgt aus:

```
schema://login-name:passwort@host:port//pfad
```

In dieser URL ist `schema` das Protokoll. Ein Großteil der Host-Information (`login-name`, `passwort` und `prot`) wird nur eingegeben, wenn sie gebraucht wird. Das Schema könnte `http` (*Hypertext Transfer Protocol*), `ftp` (*File Transfer Protocol*), `gopher` (das Gopher-Protokoll), `news` (Usenet News – was die Regel durchbricht, weil das Protokoll eigentlich `nntp` ist, *Net News Transfer Protocol*), `wais` (*Wide Area Information Servers*) oder `file` (für den lokalen Dateizugriff) sein. Es gibt noch einige andere Schemata, aber viele davon, wie beispielsweise `mailto`, sind für die Informationssuche nicht besonders sinnvoll.

Öffentliche Bezeichner

Ein Systembezeichner ist relativ einfach: Er verweist einfach auf eine Datei. Der öffentliche Bezeichner ist ein Vermächtnis von SGML. Was die Entities betrifft, bewirken öffentliche Bezeichner dasselbe wie Systembezeichner: Irgendwo werden sie in Dateinamen aufgelöst. Für Notationen stellen sie eine formalere Methode der Kennzeichnung dar. Die Verwendung öffentlicher Bezeichner erlaubt die Einbindung zusätzlicher Details, wie beispielsweise der Sprache, des Eigentümers (oder Copyright-Besitzers) und des Autors.

Es gibt in XML immer noch keine *offizielle* Methode, öffentliche Bezeichner aufzulösen. Es gibt jedoch eine SGML-Methode, die bereits seit Jahren dafür eingesetzt wird. Die meisten der großen XML-Werkzeuge stammen von SGML-Entwicklern, so daß dieselbe Methode stillschweigend in XML-Werkzeugen implementiert wurde, ohne daß darüber diskutiert wurde, ob sie erforderlich ist oder nicht.

SGML verwendet einen Mechanismus zur Auflösung von öffentlichen Bezeichnern, der auf einem Industrieabkommen basiert, das 1997 vom SGML Open Consortium (das jetzt seinen Namen in OASIS geändert hat – *Organization for the Advancement of Structured Information Standards*) als Technical Resolution 9401 veröffentlicht wurde. Bekannter unter dem Namen SGML Open Catalog (SOC), verwendet dieser Mechanismus eine *Katalogdatei*, die sich im selben Verzeichnis wie das Dokument befindet (die Applikation kann diese Position aber selbstverständlich ändern). Diese Datei wird normalerweise als `catalog.soc` bezeichnet und noch häufiger einfach als `catalog`. (In einer Applikation steht es Ihnen in der Regel frei, welche Datei als Katalogdatei verwendet werden soll.)

Es ist nicht sinnvoll, zu tief in die technischen Details einzusteigen, weil die Katalogdatei eigentlich eine SGML-Funktion ist, die wiederum auf einige Funktionen von HyTime zurückgeht und HyTime und SGML im Rahmen dieses Buchs nicht besprochen werden können. Uns betrifft nur, daß es sich bei der Katalogdatei um eine ASCII-Datei handelt, die aus Zeilen besteht, in denen ein öffentlicher Bezeichner (offiziell ein FSI, Formal System Identifier) mit einem Systemobjekt-Bezeichner gepaart wird. Ein *Systemobjekt-Bezeichner* ist grundsätzlich eine Datei, könnte aber auch eine andere Art Bezeichner sein, den das System in etwas Sinnvolles umwandeln könnte. Ein Beispiel für eine typische Katalogdatei sehen Sie in Listing 8.3.

Listing 8.3: Eine typische Katalogdatei

```
1:  -- catalog: SGML Open style entity catalog for HTML --
2:  -- $Id: catalog,v 1.3 1995/09/21 23:30:23 connolly Exp $ --
3:  -- Hacked by jjc --
4:  -- Ways to refer to Level 2: most general to most specific --
5:  PUBLIC     "-//IETF//DTD HTML//EN"                    "html.dtd"
6:  PUBLIC     "-//IETF//DTD HTML 2.0//EN"                "html.dtd"
7:  PUBLIC     "-//IETF//DTD HTML Level 2//EN"            "html.dtd"
8:  PUBLIC     "-//IETF//DTD HTML 2.0 Level 2//EN"        "html.dtd"
9:
10: -- Ways to refer to Level 1: most general to most specific --
11: PUBLIC     "-//IETF//DTD HTML Level 1//EN"            "html-1.dtd"
12: PUBLIC     "-//IETF//DTD HTML 2.0 Level 1//EN"        "html-1.dtd"
13:
14: -- Ways to refer to Strict Level 2: most general to most specific --
15: PUBLIC     "-//IETF//DTD HTML Strict//EN"             "html-s.dtd"
16: PUBLIC     "-//IETF//DTD HTML 2.0 Strict//EN"         "html-s.dtd"
17: PUBLIC     "-//IETF//DTD HTML Strict Level 2//EN"     'html-s.dtd"
18: PUBLIC     "-//IETF//DTD HTML 2.0 Strict Level 2//EN" "html-s.dtd"
19:
20: -- Ways to refer to Strict Level 1: most general to most specific --
21: PUBLIC     "-//IETF//DTD HTML Strict Level 1//EN"     "html-1s.dtd"
22: PUBLIC     "-//IETF//DTD HTML 2.0 Strict Level 1//EN" "html-1s.dtd"
23:
24: -- ISO latin 1 entity set for HTML --
25: PUBLIC     "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML" ➥ISOlat1.sgm
```

Beachten Sie, daß es sich bei dem in Listing 8.3 gezeigten Beispiel um eine modifizierte XML-Version einer SGML-Katalogdatei handelt. In XML muß der Dateiname in Anführungszeichen eingeschlossen werden, was bei SGML nicht der Fall ist.

Es hindert Sie nichts daran, eine Katalogdatei in einem Texteditor anzulegen, aber es gibt einige kostenlose Pakete zur Katalogverwaltung (auch als Pakete für die Entitiy-Verwaltung bezeichnet), die Sie aus dem Internet herunterladen können. Einige Softwarepakete haben eine eigene eingebaute Funktion, häufig als Entity-Manager bezeichnet, die Entities auflöst. Alles was Sie wissen müssen, damit der Mechanismus funktioniert, ist, daß der linke Teil einer Zeile (die FSI) in einer Katalogdatei exakt mit der Deklaration im XML-Dokument übereinstimmen muß.

Parameter-Entities

Ein Parameter-Entity ist eine ganz andere Art von Entity, weil es eine wichtige Einschränkung hat: Verweise auf ein Parameter-Entitiy dürfen nur innerhalb einer DTD erfolgen.

```
<!ENTITY %book "front ¦ body ¦ back" >
```

Um Parameter-Entities von allgemeinen Entities zu unterscheiden (und zu verhindern, daß sie in einem Dokument verwendet werden), wird bei ihrer Deklaration und beim Verweis darauf ein Prozentzeichen (%) angegeben:

```
<!ENTITY % "front ¦ body ¦ back" >
```

Parameter-Entities sind extrem praktisch als Abkürzungen für Deklarationsteile, die in einer DTD häufig vorkommen. Sie können jedoch kein Markup enthalten (vollständige Deklarationen). Sie können nur Teile von Deklarationen enthalten:

```
<!ENTITY common "(para ¦ body ¦ text)">
<!ELEMENT chapter ((%common;)*, section+)>
<!ELEMENT section (%common;)>
```

Bei der Auflösung eines Verweises auf ein Parameter-Entity werden ein führendes Leerzeichen und ein nachfolgendes Leerzeichen an den Ersatztext angefügt, um sicherzustellen, daß er eine ganzzahlige Menge grammatikalischer Token enthält.

Entity-Auflösung

Die Regeln für die Entity-Auflösung – wann Entities interpretiert und wo sie ignoriert werden – können relativ kompliziert sein. Tabelle 8.1 zeigt, was mit Entity-Verweisen und Zeichenverweisen passiert. Die linke Spalte beschreibt, wo der Entity-Verweis auftritt:

- **Innerhalb eines Elements** – Der Verweis erscheint irgendwo hinter dem Start-Tag und vor dem Ende-Tag eines Elements.
- **In einem Attributwert –** Der Entity-Verweis erfolgt innerhalb des Werts eines Attributs in einem Start-Tag oder in einem Vorgabewert in einer Attributdeklaration.
- **Als Name in einem Attributwert –** Der Entity-Verweis erscheint als Name, nicht als Entity-Verweis, als Wert eines Attributs, das mit dem Typ `ENTITY` oder `ENTITIES` deklariert wurde.

```
<?xml version="1.0" standalone="yes"?>
          <!DOCTYPE graphic SYSTEM "graphic.dtd" [
            <!ELEMENT graphic (icon)+>
            <!ELEMENT icon EMPTY>
            <!ATTLIST icon
                      source   ENTITY #REQUIRED
                      height   NMTOKEN #IMPLIED
                      nsoffset NMTOKEN #REQUIRED
                      width    NMTOKEN #IMPLIED >
            <!ENTITY icon8 SYSTEM "icon813.gif" NDATA gif>
          ]>
          <graphic>
            <icon source="icon8"
                  height="0.391in" nsoffset="0.000in"
                  width="0.429in"/>
          </graphic>
```

- **In einem Entity-Wert** – Der Verweis erscheint in einem Parameter oder im Entity-Wert in der Entity-Deklaration.
- **In der DTD –** Der Verweis erscheint innerhalb der internen oder der externen Untermenge der DTD, aber außerhalb eines Entity oder Attributwerts.

Stelle, an der der Verweis erfolgt	**Entity-Typ**			**Zeichenverweis**
	Parameter	**Intern**	**Extern**	
Innerhalb eines Elements	Ignoriert	Ersetzt	Ersetzt bei der Auswertung	Ersetzt
In einem Attributwert	Ignoriert	Ersetzt	Nicht erlaubt	Ersetzt
Name in einem Attributwert	Ignoriert	Nicht erlaubt	Der Applikation übergeben	Ignoriert

Tabelle 8.1: Entity-Auflösung

Stelle, an der der Verweis erfolgt	Entity-Typ			Zeichenverweis
	Parameter	**Intern**	**Extern**	
In einem Entity-Wert	Ersetzt[a]	Ignoriert	Nicht erlaubt	Ersetzt
In der DTD	Ersetzt bei der Auswertung	Nicht erlaubt	Nicht erlaubt	Nicht erlaubt

Tabelle 8.1: Entity-Auflösung

a. Wenn der Entity-Verweis in einem Attributwert erscheint oder ein Parameter-Verweis in einem Entity-Wert auftritt, werden einfache und doppelte Anführungszeichen ignoriert, so daß der Wert nicht vorzeitig abgeschlossen wird.

Der Ersatztext für ein Entity kann Zeichenverweise, Verweise auf Parameter-Entities sowie Verweise auf allgemeine Entities enthalten. Zeichenverweise und Verweise auf Parameter-Entities im Wert eines Entity werden bei der Auflösung des Entity aufgelöst. Allgemeine Entity-Verweise werden ignoriert.

Wenn ein Entity-Verweis ersetzt wird, wird der Ersatztext ermittelt und anstelle des eigentlichen Verweises verarbeitet, so als wäre er Teil des Dokuments an der Position, wo der Verweis erkannt wurde. Der Ersatztext darf Zeichendaten und Markup enthalten (außer bei Parameter-Entites). Diese werden auf die übliche Weise erkannt, außer daß der Ersatztext der Entities `amp`, `lt`, `gt`, `apos` und `quot` immer als Daten behandelt wird.

Bevor ein Dokument ausgewertet werden kann, muß auch der Ersatztext für geparste Entities geparst werden. Handelt es sich um ein externes Entity und das Dokument wird nicht ausgewertet, muß der Ersatztext nicht geparst werden. Wird der Ersatztext nicht geparst, bleibt es der Applikation überlassen, wie sie ihn behandelt.

Wenn der Name eines nicht geparsten Entities im Wert eines Attributs erscheint, das mit dem Typ `ENTITY` oder `ENTITIES` deklariert wurde, werden der System- und (gegebenenfalls) der öffentliche Bezeichner für das Entity und die ihm zugeordnete Notation einfach der Applikation übergeben. Die Applikation ist dann für die weitere Verarbeitung verantwortlich (wie beispielsweise die Anzeige der Grafik in einem Fenster).

Wie man das meiste aus Entities macht

Wie Sie bereits wissen, gibt es eine feste Reihenfolge, in der externe und interne DTD-Untermengen gelesen und interpretiert werden. Als erstes wird die interne DTD-Untermenge gelesen, anschließend die externe DTD-Untermenge. Wenn in der internen DTD-Untermenge etwas deklariert wird, kann diese Deklaration in der externen DTD-Untermenge nicht geändert werden. Die Deklarationen können jedoch zum Teil ergänzt werden (beispielsweise durch zusätzliche Attribute).

Bei einer sorgfältigen Planung können Sie diese hierarchische Anordnung nutzen, um Ihre Informationen zu konstruieren, beispielsweise eine Web-Site, so daß die gemeinsamen Daten so gut wie möglich genutzt werden (siehe Abbildung 8.3):

- Globale Deklarationen können in einer zentralen DTD abgelegt werden, die alle Dokumente abdeckt. Was die einzelnen Dokumente betrifft, ist diese zentrale DTD die (gemeinsame) externe DTD-Untermenge.

  Die zentrale DTD verweist auf Entities, die allgemeine Grafiken beinhalten (Firmenlogos und Briefköpfe, Sonderzeichen usw.).

- Lokale Deklarationen können in der internen DTD-Untermenge der einzelnen Dokumente angelegt werden. Damit ist es den einzelnen Dokumenten möglich, die globalen Deklarationen zu überschreiben und sie ihren eigenen Bedürfnissen anzupassen.

Die interne DTD-Untermenge in jedem einzelnen Dokument verweist auf alle Grafiken und Texte, die sie benötigt.

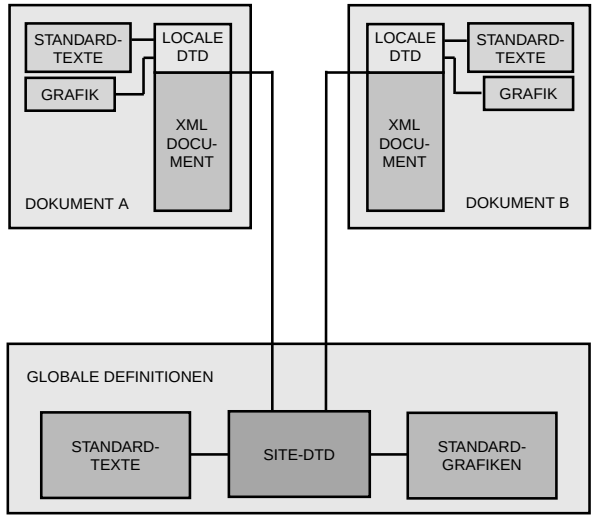


Abbildung 8.3:
Lokale und globale Entities.

Durch das Aufbrechen der Dokumente in Module und die Ausnutzung der Parsing-Reihenfolge interner und externer DTD-Untermengen können Sie viele Dokumente modularisieren, aber Sie können gegebenenfalls auch noch weiter gehen. Sie können sogar die DTD selbst modularisieren und die Teile zusammenfassen, die Sie für einen bestimmten Dokumenttyp benötigen (obwohl man ihn in diesem Kontext fast als Untertyp bezeichnen könnte).

Es gibt viele große industrielle Implementierungen von SGML, die dieses Schema verwenden. Einige Leute nennen es auch das Pizza-Modell, wobei man seine DTD mit einer Grundmenge der zwingend erforderlichen Elemente aufbaut und dann die Zutaten ergänzt, die für das jeweilige Dokument benötigt werden. Beispiele für große DTDs, die dieses Modell verwenden, sind die Text Encoding Initiative DTD und die DTDs, die das Amerikanische Verteidigungsministerium für IETM (*Interactive Electronic Technical Manuals*) verwendet.

Derselbe Ansatz könnte ganz einfach auch in XML verwendet werden. Und er könnte hier sogar sehr viel einfacher verwendet werden, weil XML Internet-URLs als Positionsangaben für Fragmente verarbeiten kann (was dann wiederum in SGML eingebaut wird). Damit ist es viel einfacher, Daten physisch zu verteilen und dabei in der Lage zu sein, das Ganze als kohärente Informationsmenge zusammenzustellen.

Zeichendaten und Zeichensätze

Wie Sie bereits erfahren haben, enthalten XML-Entities Daten, die entweder geparst (durch den XML-Interpreter verarbeitet) oder nicht geparst (normalerweise Nicht-XML-Daten) werden. Geparste Daten bestehen aus Zeichen, die zum Teil Zeichendaten bilden und zum Teil Markup.

Ganz allgemein ausgedrückt, besteht ein XML-Entity einfach aus einer Folge von Zeichen, die wiederum in grundlegende Texteinheiten und nicht weiter zerlegt werden können.

Intern verwendeten die Computer bisher in der Regel sieben Bit, um Ziffern und Buchstaben in digitaler Form darzustellen. Diese Darstellung wurde als ISO/IEC 646 standardisiert, dem heute gebräuchlichen ASCII-Schema, wobei jedem Buchstaben, jeder Ziffer und jedem Interpunktionszeichen ein eigener 7-Bit-Code zugeordnet wird, beispielsweise:

- Der Buchstabe x ist als `1111000` abgelegt.
- Der Buchstabe *m* ist als `1101101` abgelegt.
- Der Buchstabe *l* ist als `1101100` abgelegt.

Statt die Binärdarstellung auszuschreiben, werden diese Muster in der Regel durch eine *Hexadezimalzahl* ausgedrückt, wie etwa *6B*, was für den Buchstaben *k* steht.

Der Bereich der erlaubten Zeichen, also der Zeichen, die in einem XML-Dokument erscheinen dürfen, umfaßt die folgenden hexadezimalen Werte:

- `09` (das Tabulator-Zeichen)
- `0D` (das CR-Zeichen)

- `0A` (das LF-Zeichen)
- `20` bis `D7FF`, `E000` bis `FFFD` und `10000` bis `10FFFF` (die legalen Grafikzeichen von Unicode und ISO 10646)

Unicode und ISO 10646 sind standardisierte Zeichensätze, die Sie gleich kennenlernen werden.

Zeichensätze

Im grundlegenden ASCII-Alphabet gibt es nur 128 verschiedene mögliche 7-Bit-Muster, so daß nur 128 unterschiedliche Zeichen dargestellt werden können. Das achte Bit (acht Bit sind ein Byte, die grundlegende Einheit im Computerspeicher) wird als *Prüfbit* verwendet, um sicherzustellen, daß das Byte korrekt gespeichert oder übertragen wird. Diese 128 Zeichen werden auch als Standard-ASCII-Zeichensatz bezeichnet und waren jahrelang die Grundlage der Programmierung.

Als die Computer komplexer wurden und sich auf der ganzen Welt verbreiteten, brauchte man zusätzliche Zeichen, um Dinge wie etwa die akzentuierten Zeichen aus verschiedenen europäischen Sprachen darzustellen. Das achte Bit wurde neu definiert, so daß man 8-Bit-Zeichensätze erhielt, wodurch sich die Anzahl möglicher Zeichen auf 256 erhöhte. Dies ist als ISO-8859-Zeichensatz standardisiert. Es gibt zahlreiche Varianten für ISO 8859, die jeweils auf eine bestimmte Sprache abgestimmt sind. Die Version, die Sie wahrscheinlich am häufigsten sehen werden, ist 8859/1, der Zeichensatz für HTML, der auch von den Web-Browsern verstanden wird. Dieser Zeichensatz beinhaltet akzentuierte Zeichen, Zeichenumrisse und eine Auswahl der gebräuchlichsten griechischen Buchstaben für Wissenschaft und Technik sowie verschiedene andere Symbole. Die ersten 128 Zeichen von ISO 8859/1 stimmen exakt mit ISO 646 überein, es ist also abwärtskompatibel.

Acht Bit sind ausreichend für die meisten westlichen Sprachen, aber geradezu nutzlos für Sprachen wie Arabisch, Chinesisch, Urdu usw. Um diese Sprachen abzudecken, haben Unicode (mit 16-Bit-Codierung) und ISO 10646 den nächsten logischen Schritt unternommen, um 32-Bit-Muster für die Zeichendarstellung zu unterstützen. Damit können mehr als zwei Milliarden Zeichen dargestellt werden. ISO 10646 bietet eine Standarddefinition für alle Zeichen der meisten europäischen und asiatischen Sprachen. Unicode wird auch in Microsoft Windows NT verwendet.

Unicode beinhaltet zahlreiche unterschiedliche Codierungsschemata, die nach der dabei verwendeten Bitanzahl benannt sind. UCS-2 verwendet 16 Bit (zwei Byte), was identisch mit Unicode ist, und UCS-4 verwendet 32 Bit (vier Byte).

ISO 10646 ist noch komplexer. Durch die Zuordnungsschemata, die als *UCS Transformation Formats* (UTF) bezeichnet werden, erlaubt ISO 10646 die Verwendung ei-

ner variablen Bitanzahl. Es ist wenig sinnvoll, so viele Bits zu verwenden, wenn Sie nur die grundlegenden 128 ASCII-Zeichen versenden wollen; deshalb ermöglicht Ihnen ISO 10646, bei Bedarf zusätzliche Zeichen anzufordern.

XML unterstützt zwei UTF-Formate: UTF-8 (acht Bit- bis 48-Bit-Codierung) und UTF-16 (bis zu 32-Bit-Codierung, aber unter Verwendung einer Zuordnung, die mehr als eine Million Zeichen erlaubt).

Entity-Codierung

Jedes Text-Entity in XML verwendet eine andere Codierung für seine Zeichen. Beispielsweise können Sie separate Text-Entities oder Elemente deklarieren, um die Abschnitte eines XML-Dokuments mit chinesischen oder arabischen Zeichen zu deklarieren, und diesen Abschnitten die 16-Bit-UCS-2-Codierung zuordnen. Das restliche Dokument kann die effizientere 8-Bit-Codierung verwenden.

Standardmäßig wird die ISO-10646-UTF-8-Codierung angenommen. Wenn das Text-Entity eine andere Codierung verwendet, müssen Sie diese am Anfang des Entities deklarieren:

```
<?xml encoding="Codierungs.Name" ?>
```

Dabei ist `Codierungs.Name` der Name eines Zeichensatzes, der nur die Zeichen aus dem römischen Alphabet (A bis Z und a bis z), arabische Ziffern, Punkte, Trennstriche und Unterstriche verwendet. Der XML-Interpreter muß die folgenden Zeichensatznamen erkennen:

- UTF-8
- UTF-16
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- ISO-8859-1 bis -9
- ISO-2022-JP
- Shift-JIS
- EUC-JP

Beispiele für Codierungsdeklarationen sehen wie folgt aus:

```
<?xml version="1.0" encoding='UTF-16'?>
<?xml version="1.0" standalone="yes" encoding="EUC-JP"?>
```

Die Vorgabe (UTF-8) wird erkannt, wenn die ersten vier Byte eines XML-Text-Entity die Hexadezimalwerte `3C`, `3F`, `58` und `4D` haben, die ersten vier Zeichen der Codierungsdeklaration. Wenn es keine Deklaration gibt oder wenn keines der anderen Codierungsschemata erkannt wird, wird vorausgesetzt, daß das Entity in UTF-8 vorliegt.

Alle XML-Interpreter können Entities in UTF-8 oder UTF-16 lesen. Das Standard-ASCII-Alphabet ist im ersten Teil des UTF-8-Zeichensatzes enthalten; Sie müssen also den Zeichensatz für das Entity nicht deklarieren, wenn es in reinem ASCII codiert ist.

Wird ein XML-Text-Entity in UCS-2 codiert, muß es mit einer entsprechenden Codierungssignatur beginnen, dem sogenannten Byte Order Mark (Markierung für die Byte-Reihenfolge). Dabei handelt es sich um `FF FF` für die Standard-Byte-Reihenfolge und `FE FF` für die umgekehrte Byte-Reihenfolge. Diese Zeichen werden nicht als Teil des Markups oder der Zeichendaten des XML-Dokuments betrachtet.

Neben der Verwendung der Codierungs-Deklaration darf der XML-Interpreter auch den ersten paar Bytes entnehmen, welche Codierung eingesetzt wird. Darüber hinaus kann auch die MIME-Typ-Identifikation `text/xml` verwendet werden, die von einem Web-Server hinzugefügt wird, um die verwendete Zeichencodierung zu erkennen.

Entities und Entity-Mengen

Der Wechsel zu einer anderen Codierung ist nicht die einzige Methode, Zeichen darzustellen, die nicht im UTF-8-Zeichensatz enthalten sind. Vergessen Sie nicht, daß Sie auf jedes Zeichen verweisen können, indem Sie seine ISO-10646-Zeichennummer in einem Zeichenverweis angeben (wie beispielsweise `&#38;`).

Sie können auch ein Entity deklarieren, das das von Ihnen benötigte Zeichen darstellt, beispielsweise die folgende Deklaration des Gradzeichens (°), das aus dem ISO-8859-1-Zeichensatz stammt:

```
<!ENTITY deg    "&#176;">
```

Sie können in Ihrem XML-Dokument an beliebiger Stelle auf dieses Entity verweisen:

```
<para>Die Temperatur beträgt heute im Süden 38 &deg;C.</para>
```

Nicht alle Computersysteme und Übertragungsmedien verarbeiten die komplexen Zeichensätze, die Sie kennengelernt haben. Immer noch herrscht der 7-Bit-ASCII-Zeichensatz vor. Diese Arten von Zeichen-Entity-Deklarationen gibt es schon seit den ersten Tagen von SGML, und sie wurden in *Entity-Mengen* gesammelt.

Diese Entity-Mengen sind Teil des SGML-Standards (ISO 8879) und laufen unter den etwas kryptischen Namen `ISOlag1` (römisches Alphabet, akzentuierte Zeichen), `ISOnum` (numerische und Sonderzeichen) `ISOcyr1` (kyrillische Zeichen, die im Russischen verwendet werden) usw. Es handelt sich dabei um echte SGML-Funktionen, die in

XML nicht unverändert genutzt werden können. (XML erlaubt die Verwendung der SDATA-Notation (Systemdaten) nicht). Es wurden jedoch bereits XML-Versionen der wichtigsten dieser Entity-Mengen veröffentlicht, wie beispielsweise die XML-Verison der `ISOdia`-Entity-Menge (diakritische Zeichen), die Sie in Listing 8.4 sehen.

Listing 8.4: Die für XML abgeänderte ISOdia-Entity-Menge

```
<!-- (C) International Organization for Standardization 1986
     Permission to copy in any form is granted for use with
     conforming SGML systems and applications as defined in
     ISO 8879, provided this notice is included in all copies.
 -->
<!-- Character entity set. Typical invocation:
     <!ENTITY % ISOdia PUBLIC
            "ISO 8879:1986//ENTITIES Diacritical Marks//EN//XML">
     %ISOdia;

-->
<!-- This version of the entity set can
     be used with any SGML document
     which uses ISO 10646 as its document character set.
     This includes XML documents and ISO HTML documents.
     This entity set uses hexadecimal numeric character references.

     Creator: Rick Jelliffe, Allette Systems

     Version: 1997-07-07          -->

<!ENTITY acute  "&#180;" ><!--=acute accent-->
<!ENTITY breve  "&#x2D8;" ><!--=breve-->
<!ENTITY caron  "&#x2C7;" ><!--=caron-->
<!ENTITY cedil  "&#184;" ><!--=cedilla-->
<!ENTITY circ   "^" ><!--=circumflex accent-->
<!ENTITY dblac  "&#x2DD;" ><!--=double acute accent-->
<!ENTITY die    "&#168;" ><!--=dieresis-->
<!ENTITY dot    "&#x2D9;" ><!--=dot above-->
<!ENTITY grave  "`" ><!--=grave accent-->
<!ENTITY macr   "&#175;" ><!--=macron-->
<!ENTITY ogon   "&#x2DB;" ><!--=ogonek-->
<!ENTITY ring   "&#x2DA;" ><!--=ring-->
<!ENTITY tilde  "&#x2DC;" ><!--=tilde-->
<!ENTITY uml    "&#168;" ><!--=umlaut mark-->
```

Die Entity-Mengen werden als separate Dateien bereitgestellt, eine für jede dieser Mengen. Wenn Sie wissen, daß Sie bestimmte Zeichen für ein XML-Dokument brauchen, nehmen Sie einfach die erforderliche Deklaration in Ihre XML-DTD auf, wie in Listing 8.5 gezeigt.

Listing 8.5: Typische Notationsdeklarationen in einer DTD

```
1:  <?xml version="1.0" standalone="no"?>
2:  <!DOCTYPE chapter SYSTEM "chapter.dtd" [
3:
4:  <!ENTITY % ISOlat1 PUBLIC "ISO 8879-1986//ENTITIES
5:                     Added Latin 1//EN//XML" "isolat1.xml">
6:  %ISOlat1;
7:
8:  <!ENTITY % ISOnum PUBLIC "ISO 8879:1986//ENTITIES
9:             Numeric and Special Graphic//EN//XML" "isonum.xml">
10: %ISOnum;
11:
12: <!ENTITY % ISOpub PUBLIC "ISO 8879:1986//ENTITIES
13:            Publishing//EN//XML" "isopub.xml">
14: %ISOpub;
15: ]>
16: <chapter><number/> ... </chapter>
```

Beachten Sie, daß sich die Form der Entity-Deklaration in der DTD von der in der eigentlichen Entity-Datei verwendeten (Listing 8.4) etwas unterscheidet. Es müssen ein Systembezeichner sowie ein öffentlicher Bezeichner verwendet werden, aber es gibt das Schlüsselwort `SYSTEM` nicht.

Zusammenfassung

Im heutigen Kapitel haben Sie mehrere Methoden kennengelernt, Zeichen in ein XML-Text-Entity einzufügen, die auf Ihrer Tastatur normalerweise nicht zur Verfügung stehen. Sie haben nicht nur die Zeichencodierungsschemata kennengelernt, sondern auch, wie über Entity-Mengen ganze Sätze von Sonderzeichen bereitgestellt werden.

Darüber hinaus haben Sie erfahren, wie man externe Grafikdateien einbindet, indem man sie mit einer speziellen Notation deklariert und im XML-Dokument darauf verweist und wie man eine Hilfsapplikation deklariert, mit der die Daten verarbeitet werden können.

Neben der Tatsache, daß Sie externe Entities verwenden müssen, um binäre Daten in Ihre formatierte XML-Ausgabe einzubinden, wie beispielsweise Grafikdateien (im rohen XML-Code können Sie nur Entity-Verweise angeben), können Entities ein wunderbares Werkzeug sein, eine komplexe Sammlung zahlreicher Dokumente in überschaubare Stücke zu zerlegen. Durch Verwendung interner und externer Text-Entities, kombiniert mit der hierarchischen Reihenfolge bei der Interpretation der beiden DTD-Untermengen, können Sie eine abgestufte Hierarchie mit Schabloneninhalt anlegen. Auf der globalen Ebene können Sie sicher Vorgabeeinstellungen treffen und es den einzelnen XML-Dokumenten überlassen, diese gegebenenfalls zu überschreiben.

Wie Sie in Kapitel 10 erfahren werden, können die extrem leistungsfähigen Linking-Funktionen von XML diese Aufgabe noch einfacher machen. In Kapitel 9 werden Sie alles, was Sie bisher gelernt haben, einsetzen, um XML-Dokumente zu analysieren – die ultimative Prüfung, ob Ihr XML-Code funktioniert oder nicht.

F&A

F Wie können Sie nichtgeparsten (Nicht-XML-)Text in ein externes Entity einfügen?

A *Es gibt mehrere Methoden. Sie können eine TEXT-Notation deklarieren, aber das erlaubt Ihnen nicht, den Text physisch in das XML-Dokument aufzunehmen. (Er würde dann an die Hilfsapplikation weitergereicht, die Sie in der Notationsdeklaration angegeben haben.) Die beste Methode ist wahrscheinlich, die Datei mit dem Text als externes Text-Entity zu deklarieren und den Text aus dieser Datei in einen CDATA-Abschnitt zu stellen.*

F Man kann die Parameter-Entities nicht in einem XML-Dokument verwenden, aber man kann allgemeine Entities in einer DTD verwenden?

A *Nein. Parameter-Entities sind genau dasselbe wie allgemeine Entities. Sie enthalten Markup-Text (den einzigen Text, den Sie außer Kommentaren in einer DTD verwenden können), deshalb haben sie genau dieselbe Aufgabe.*

F Gibt es Beschränkungen hinsichtlich der Größe eines Entity?

A *Nicht durch XML. SGML setzt einige Beschränkungen für die Größe bestimmter Objekte (beispielsweise die Länge von Namen), aber sie wurden in XML alle aufgehoben. Einschränkungen finden nur durch Ihre XML-Applikation statt oder durch den Computer, auf dem Sie arbeiten.*

F Können Sie ANSI-Code für ein Zeichen verwenden?

A *Nein. Die Eingabe des ANSI-Codes für ein Zeichen war eine Funktion einiger Microsoft-Windows-Pakete; sie hat nichts mit XML zu tun. Es gibt immer weniger Pakete, die diese Abkürzungen erlauben, und es handelte sich dabei um nichts anderes als um eine Abkürzung. Es wurde nämlich damit einfach nur eine andere Methode bereitgestellt, im aktuellen Codierungsschema ein Zeichen einzugeben, das auf der Tastatur nicht zur Verfügung stand. Wenn Ihre Bearbeitungs-Software dieses Schema unterstützt und Sie es nutzen wollen, dann tun Sie das. Verwechseln Sie es jedoch nicht mit der Zeichencodierung.*

F Was passiert, wenn die für eine Notation deklarierte Software nicht vorhanden ist oder aus irgendeinem Grund nicht ausgeführt werden kann?

A *Was XML betrifft, nichts. Der XML-Interpreter prüft nur, ob die Deklaration korrekt ist. Es bleibt der XML-Applikation überlassen, den Inhalt eines Entity zu verarbeiten, das die Notation verwendet. Die Applikation ist außerdem dafür verantwortlich, was passiert, wenn die Hilfsapplikation nicht zur Verfügung steht (schließlich kann sie auch einen eigenen internen Viewer haben).*

Übungen

1. Entwickeln Sie eine einfache DTD für einen grundlegenden Geschäftsbrief. Sie muß nicht kompliziert sein – nur genug für einen Briefkopf, Adreßblöcke und einen einfachen Text.
2. Legen Sie einen einfachen Brief in XML an, und benutzen Sie dazu Ihre Brief-DTD. Erweitern Sie den XML-Code im Dokument und in der DTD so, daß ein Standardbriefkopf eingebunden werden kann, der ein Firmenlogo im GIF-Format enthält.
3. Richten Sie Ihre XML-Applikation so ein, daß Sie schnell und einfach Ihren Namen in den Signaturblock einsetzen können.

Gültigkeits-prüfung

Woche 2

In den vergangenen Kapiteln haben Sie gelernt, wie man Dokumenttypdefinitionen (DTDs) entwickelt, die formalen Spezifikationen der Struktur Ihrer XML-Dateien. Jetzt haben Sie wahrscheinlich zwei Fragen:

- Wie kann ich prüfen, ob meine DTD korrekt ist?
- Wie kann ich prüfen, ob meine Dokumente den in der DTD definierten Regeln entsprechen?

Diese Aufgabe können auswertende Parser für Sie übernehmen. In diesem Kapitel lernen Sie die folgenden Dinge:

- DTDs mit Hilfe von DXP prüfen
- DTDs mit Hilfe von XML für Java prüfen
- Mit Hilfe des DXP-Parsers prüfen, ob XML-Dateien den Regeln einer DTD entsprechen
- Mit Hilfe des Parsers XML für Java prüfen, ob XML-Dateien den Regeln einer DTD entsprechen

Einen Überblick über die verfügbaren Parser finden Sie in Kapitel 5.

Prüfen Ihrer DTD mit DXP

Wie Sie den DXP-Parser erhalten und installieren, wurde bereits in Kapitel 5 beschrieben. Hier wollen wir zeigen, wie er für die Überprüfung Ihrer DTDs genutzt wird.

Geben Sie an der DOS-Eingabeaufforderung folgendes ein:

```
jre -cp .;c:\datachannel\dxp\classes dxpcl -s -v c:\xmlex\file.xml
```

Dabei gilt:

- `jre` ruft die Java Laufzeit-Engine auf (Java Runtime Engine).
- `-cp` setzt den Klassenpfad (class path; dort, wo man die verwendeten Klassen findet). In diesem Fall werden zwei Pfade angegeben: Der erste (.) verweist auf das aktuelle Arbeitsverzeichnis, der zweite ist `c:\datachannel\dxp\classes`. Sie werden durch ein Semikolon voneinander abgetrennt.
- `dxpcl` ist der Name des Java-Programms (Klasse).
- `-s` steht für Silent Mode (Stiller Modus).

Hier sind wir nur an den Fehlern in Ihrer Datei interessiert. Im Silent Mode werden nur Fehlermeldungen ausgegeben.

- `-v` steht für `Validation on`, aktiviert also die Auswertung.
- `c:\xmlex\file.xml` ist die zu überprüfende Datei.

Eine DTD-Überprüfung mit DXP

Jetzt werden wir die in Listing 9.1 gezeigten Markup-Deklarationen prüfen.

Listing 9.1: *`dtdv.dtd`* ***– Deklarationen, die jetzt mit DXP geprüft werden.***

```
1: <!ENTITY % admonitions "(tip ¦ warning ¦ note)" >
2: <!ENTITY % paracontent "(#PCDATA ¦ icon ¦ menu ¦ xref ¦
➥iconbmp)*" >
3:
4: <!ELEMENT helptopic (title, rule, procedure, rule?,
➥%admonitions;) >
5: <!ATTLIST helptopic id ID #IMPLIED>
6:
7: <!ELEMENT title (#PCDATA) >
8: <!ATTLIST title keyword CDATA>
9:
10: <!ELEMENT procedure (step+)>
12: <!ELEMENT step (action, (%admonitions;)*) >
13:
14: <!ELEMENT action %paracontent; >
15: <!ELEMENT tip %paracontent; >
16: <!ATTLIST tip targetgroup (beginners ¦ specialists)
➥"beginners" >
17:
18: <!ELEMENT warning %paracontent; >
19: <!ELEMENT note %paracontent; >
20: <!ELEMENT icon (#PCDATA) >
21: <!ELEMENT menu (#PCDATA¦shortcut)+>
22:
23: <!ELEMENT xref (#PCDATA) >
24: <!ATTLIST xref linkend idref #REQUIRED>
```

```
25:
26: <!ELEMENT shortcut (#PCDATA)>
27: <!ELEMENT tip (#PCDATA) >
28:
29: <!ELEMENT iconbmp EMPTY>
30: <!ATTLIST iconbmp src ENTITY #REQUIRED
31:               type NOTATION (bmp ¦ gif) "gif">
```

Hier haben Sie die Gelegenheit, Ihr Wissen zu testen. Bevor Sie dem Parser die Fehlersuche überlassen, versuchen Sie doch einfach, sie selbst zu erkennen! Finden Sie heraus, wie viele Fehler es gibt, beschreiben Sie Sie, und suchen Sie nach einer Lösung.

Markup-Deklarationen können an zwei Stellen erscheinen:

- in der externen oder
- in der internen Untermenge der Dokumenttypdeklaration

Wenn Sie sich in der externen Untermenge befinden, stehen die Markup-Deklarationen in einer externen Datei (einer speziellen Art externem Entity), und die Dokumenttypdeklaration in der XML-Datei muß auf diese externe Datei verweisen:

```
<!DOCTYPE helptopic SYSTEM "http://www.protext.be/help.dtd" []>
```

In unserem Fall sollte die XML wie in Listing 9.2 gezeigt aussehen.

Listing 9.2:** `dtdv.xml` **– einfache XML-Datei zum Parsing unter Verwendung von DXP

```
1:  <?xml version="1.0" ?>
2:  <!DOCTYPE helptopic SYSTEM "dtdv.dtd" [
3:  ]>
```

Sie haben Ihre XML-Datei mit einer Dokumenttypdeklaration begonnen, die auf die externe Untermenge in der Datei `dtdv.dtd` verweist.

Beachten Sie, daß sich Ihre XML-Datei (`dtdv.xml`) im Dateisystem im selben Unterverzeichnis wie `dtdv.dtd` befinden muß.

Jetzt führen wir DXP aus, um Fehler zu suchen:

```
jre -cp .;c:\datachannel\dxp\classes dxpcl -s -v c:\xmlex\dtdv.xml
```

Die erste Fehlermeldung erscheint:


```
FATAL ERROR: encountered ">". Was expecting one of: <EOF> , <S>
Location: file:///c:/xmlex/dtdv.dtd:8:30

Found errors/warnings: 1 fatal error(s), 0 error(s) and 0 warning(s)
```

Die Struktur der Fehlermeldungen ist in Kapitel 5 erklärt.

Das Problem hat mit der folgenden Zeile zu tun:

```
"<!ATTLIST title keyword CDATA>"
```

Hier deklarieren Sie die Attribute für das Element `title`; in diesem Fall gibt es nur eines (`keyword`).

Sie brauchen für jedes Attribut eine Definition, die sich aus den folgenden Komponenten zusammensetzen muß:

- Attributname
- Attributtyp
- Vorgabewert

Alle diese Komponenten werden durch Kommas voneinander getrennt.

In unserem Beispiel wurde nur das folgende definiert:

- Der Name, `keyword`
- Der Typ, CDATA oder `stringtype`, der jeden literalen String als Wert annehmen kann.

Sie haben den Vorgabewert vergessen. Der fehlende Vorgabewert muß einer der folgenden sein:

- #REQUIRED
- #IMPLIED
- Ein Attributwert, dem optional `#FIXED` vorausgehen kann

Und es muß ihm ein Leerzeichen vorausgehen.

Sie entscheiden sich für #IMPLIED; die Zeile muß also wie folgt aussehen:

```
<!ATTLIST title keyword CDATA #IMPLIED>
```

Parsen Sie noch einmal. Jetzt erhalten Sie:


```
FATAL ERROR: encountered "+". Was expecting: "*"
Location: file:///c:/xmlex/dtdv.dtd:21:34

Found errors/warnings: 1 fatal error(s), 0 error(s) and 0 warning(s)
```

Dieser Fehler betrifft

```
<!ELEMENT menu (#PCDATA¦shortcut)+>
```

Wenn Sie gemischten Inhalt haben – Zeichendaten durchsetzt mit untergeordneten Elementen –, muß das Inhaltsmodell einen Stern für beliebiges Vorkommen angeben. Die Zeile muß also wie folgt aussehen:

```
<!ELEMENT menu (#PCDATA¦shortcut)*>
```

Als nächsten Fehler erhalten wir:


```
FATAL ERROR: encountered "idref". Was expecting one of:
➥"ID" , "IDREF" , "IDREFS" , "ENTITY" , "ENTITIES" , "NMTOKEN" ,
➥"NMTOKENS" , "NOTATION" , "CDATA" , "%", "("


Location: file:///c:/xmlex/dtdv.dtd:24:24

Found errors/warnings: 1 fatal error(s), 0 error(s) and 0 warning(s)
```

Alle Schlüsselwörter (auch »idref«) müssen in XML in Großbuchstaben geschrieben werden. Nach der Korrektur erhalten wir:


```
ERROR: element declared twice "tip"
Location: file:///c:/xmlex/dtdv.dtd:27:11

ERROR: notation not declared "bmp"
Location: file:///c:/xmlex/dtdv.dtd:31:41

ERROR: notation not declared "gif"
Location: file:///c:/xmlex/dtdv.dtd:31:44

FATAL ERROR: encountered end of file
Location: :3:4

Found errors/warnings: 1 fatal error(s), 3 error(s) and 0 warning(s)
```

Das ist interessant. Sie haben drei Fehler und einen fatalen Fehler.

Das Element `tip` wurde zweimal deklariert. Die Spezifikation sagt deutlich, daß kein Elementtyp mehrfach deklariert werden darf. Verletzungen solcher Gültigkeitsbeschränkungen werden als Fehler betrachtet. Korrigieren Sie den Fehler, indem Sie die zweite Elementdeklaration von `tip` löschen.

Sie erhalten Fehler zu den Notationen, was verständlich ist, weil gemäß der Spezifikation alle Notationsnamen in der Deklaration deklariert werden müssen.

Fügen Sie die Notationsdeklaration für BMPs und GIFs am Anfang Ihrer Datei ein:

```
1: <!NOTATION bmp SYSTEM "paint.exe">
2: <!NOTATION gif SYSTEM "">
```

Wenn Sie diese beiden Notationsdeklarationen am Ende der Datei einfügen, werden die Probleme dadurch nicht gelöst, weil Notationen (und Entities) deklariert werden müssen, bevor ein Verweis darauf erfolgt.

Anschließend gibt es immer noch einen Fehler:


```
FATAL ERROR: encountered end of file
Location: :3:4

Found errors/warnings: 1 fatal error(s), 0 error(s) and 0 warning(s)
```

Offensichtlich ist hier die Fehlermeldung `encountered end of file`, weil unsere Datei nur einen Prolog enthält und kein Element.

Weil wir nur die DTD prüfen wollen, belastet uns das fehlende Element momentan noch nicht.

Sie haben also keine Fehler mehr in Ihrer DTD, wie in Listing 9.3 gezeigt.

Listing 9.3:** `dtdv.dtd` **– die fehlerfreie DTD

```
1: <!NOTATION bmp SYSTEM "paint.exe">
2: <!NOTATION gif SYSTEM "">
3:
4: <!ENTITY % admonitions "(tip ¦ warning ¦ note)" >
5: <!ENTITY % paracontent "(#PCDATA ¦ icon ¦ menu ¦ xref ¦
➥iconbmp)*" >
6:
7: <!ELEMENT helptopic (title, rule, procedure, rule?,
➥%admonitions;) >
8: <!ATTLIST helptopic id ID #IMPLIED>
9:
10: <!ELEMENT title (#PCDATA) >
11: <!ATTLIST title keyword CDATA #IMPLIED>
12:
13:<!ELEMENT procedure (step+)>
14:<!ELEMENT step (action, (%admonitions;)*) >
15:
16:<!ELEMENT action %paracontent; >
17:<!ELEMENT tip %paracontent; >
18:<!ATTLIST tip targetgroup (beginners ¦ specialists)
➥"beginners" >
19:
20:<!ELEMENT warning %paracontent; >
21:<!ELEMENT note %paracontent; >
22:
```

```
23:<!ELEMENT icon (#PCDATA) >
24:<!ELEMENT menu (#PCDATA¦shortcut)*>
25:
26:<!ELEMENT xref (#PCDATA) >
27:<!ATTLIST xref linkend IDREF #REQUIRED>
28:
29: <!ELEMENT shortcut (#PCDATA)>
30:
31: <!ELEMENT iconbmp EMPTY>
32: <!ATTLIST iconbmp src ENTITY #REQUIRED
33:               type NOTATION (bmp ¦ gif) "gif">
```

Für `internal` erscheinen die Deklarationen zwischen den eckigen Klammern ([und]) der Dokumenttypdeklaration:

```
<!DOCTYPE helptopic [
<!ELEMENT helptopic (title, procedure)>
<!ATTLIST helptopic id ID #REQUIRED>
...
]
<helptopic>...
```

Beachten Sie, daß die DTD eines Dokuments aus der Kombination einer internen und einer externen Untermenge besteht.

Momentan befinden sich alle Ihre Deklarationen in der externen Untermenge Ihrer Dokumenttypdeklaration.

Was passiert, wenn Sie den Inhalt Ihrer Datei in Ihre interne Untermenge kopieren und dann nicht mehr auf die externe Datei mit den Deklarationen verweisen, wie in Listing 9.4 gezeigt?

***Listing 9.4:** `dtdv.xml` – XML-Datei mit Deklarationen in der internen Untermenge*

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE helptopic [
3: <!NOTATION bmp SYSTEM "paint.exe">
4: <!NOTATION gif SYSTEM "">
5:
```

```
6: <!ENTITY % admonitions "(tip ¦ warning ¦ note)" >
7: <!ENTITY % paracontent "(#PCDATA ¦ icon ¦ menu ¦ xref ¦
➥iconbmp)*" >
8:
9: <!ELEMENT helptopic (title, rule, procedure, rule?,
➥%admonitions;) >
10: <!ATTLIST helptopic id ID #IMPLIED>
11:
12: <!ELEMENT title (#PCDATA) >
13: <!ATTLIST title keyword CDATA #IMPLIED>
14:
15: <!ELEMENT procedure (step+)>
16: <!ELEMENT step (action, (%admonitions;)*) >
17:
18: <!ELEMENT action %paracontent; >
19: <!ELEMENT tip %paracontent; >
20: <!ATTLIST tip targetgroup (beginners ¦ specialists)
➥"beginners" >
21:
22: <!ELEMENT warning %paracontent; >
23: <!ELEMENT note %paracontent; >
24:
25: <!ELEMENT icon (#PCDATA) >
26: <!ELEMENT menu (#PCDATA¦shortcut)*>
27:
28: <!ELEMENT xref (#PCDATA) >
29: <!ATTLIST xref linkend IDREF #REQUIRED>
30:
31: <!ELEMENT shortcut (#PCDATA)>
32:
33: <!ELEMENT iconbmp EMPTY>
34: <!ATTLIST iconbmp src ENTITY #REQUIRED
35:               type NOTATION (bmp ¦ gif) "gif">
36:
37: ]>
```

Das Parsen der Datei ergibt folgendes:


```
FATAL ERROR: parameter entity reference in entity value in
➥internal subset for "admonitions"

Location: file:/c:/xmlex/dtdv.xml:9:65
```

In der internen Untermenge können Verweise auf Parameter-Entities (`%admonitions;` und `%paracontent;`) nur dort erfolgen, wo auch Markup-Deklarationen auftreten können, nicht innerhalb von Markup-Deklarationen, wie die Deklaration des Elements `tip`. Das bedeutet, folgendes ist erlaubt:

```
<!DOCTYPE helptopic [
<!ATTLIST helptopic a CDATA "A11">
<!ENTITY % buttons SYSTEM "button.ent">
%buttons;
]>
```

Sie können den Verweise auf das Parameter-Entity, `%buttons;`, an dieser Stelle angeben, weil hier auch Markup-Deklarationen angegeben werden können.

Das folgende ist *nicht* erlaubt:

```
<!DOCTYPE helptopic [
<!ENTITY % paracontent "(#PCDATA ¦ emphasis)*">
<!ELEMENT tip %paracontent; >
]>
```

Das ist nicht möglich, weil der Verweis auf das Parameter-Entity in einer Markup-Deklaration erscheint. Es gibt zwei Lösungsmöglichkeiten:

- Schreiben Sie wieder alles in die externe Untermenge.
- Ersetzen Sie die Parameterverweise durch ihren deklarierten Inhalt, wie in Listing 9.5 gezeigt.

Listing 9.5:** `dtdv.dtd` **– korrigierte Datei mit ersetzten Verweise auf Parameter-Entities

```
1: <?xml version="1.0" ?>
2: <!DOCTYPE helptopic [
3: <!NOTATION bmp SYSTEM "paint.exe">
4: <!NOTATION gif SYSTEM "">
5: <!ELEMENT helptopic (title, rule, procedure, rule?,
➥(tip ¦ warning ¦ note)) >
6: <!ATTLIST helptopic id ID #IMPLIED>
7:
8: <!ELEMENT title (#PCDATA) >
9: <!ATTLIST title keyword CDATA #IMPLIED>
10:
11: <!ELEMENT procedure (step+)>
```

```
12: <!ELEMENT step (action, ((tip ¦ warning ¦ note))*) >
13:
14: <!ELEMENT action (#PCDATA ¦ icon ¦ menu ¦ xref ¦
➥iconbmp)* >
15: <!ELEMENT tip (#PCDATA ¦ icon ¦ menu ¦ xref ¦ iconbmp)* >
16: <!ATTLIST tip targetgroup (beginners ¦ specialists)
➥"beginners" >
17:
18: <!ELEMENT warning (#PCDATA ¦ icon ¦ menu ¦ xref ¦
➥iconbmp)* >
19: <!ELEMENT note (#PCDATA ¦ icon ¦ menu ¦ xref ¦ iconbmp)* >
20:
21: <!ELEMENT icon (#PCDATA) >
22: <!ELEMENT menu (#PCDATA¦shortcut)*>
23:
24: <!ELEMENT xref (#PCDATA) >
25: <!ATTLIST xref linkend IDREF #REQUIRED>
26:
27: <!ELEMENT shortcut (#PCDATA)>
28:
29: <!ELEMENT iconbmp EMPTY>
30: <!ATTLIST iconbmp src ENTITY #REQUIRED
31:               type NOTATION (bmp ¦ gif) "gif">
32:
33: ]>
```

Überprüfung Ihrer DTD mit XML für Java

XML für Java ist ein auswertender XML-Browser in zu 100 % reinem Java. Das Paket (`com.imb.xml.parser`) enthält Klassen und Methoden für das Parsing, das Erzeugen, die Änderung und das Auswerten von XML-Dokumenten. XML für Java ist ein robuster XML-Interpreter und außerdem sehr vollständig.

Installation von XML für Java

- Laden Sie XML für Java unter `http://alphaworks.ibm.com/formula/xml` herunter. Die Datei `xml4j_1_1_9.zip` umfaßt 1288 Kbyte.
- Entzippen Sie die Datei `xml4j_1_1_9.zip` in ein neues Verzeichnis, beispielsweise `c:\xml4j`.

Da es sich auch hierbei um eine Java-Implementierung handelt, sollten Sie darauf achten, daß Ihre virtuelle Java-Maschine in Version 1.x läuft. Weitere Informationen finden Sie in Kapitel 5.

XML für Java – die Anwendung

Geben Sie an der DOS-Eingabeaufforderung folgendes ein:

```
jre -cp c:\xml4j\xml4j.jar trlx c:\xmlex\dtdv.xml
```

Dabei gilt:

- `jre` ruft die Java Laufzeit-Engine auf.
- `-cp` gibt den Klassenpfad an. In diesem Fall verweisen Sie auf die `jar` (Java-Archivdatei) mit dem Namen `xml4j.jar`.
- `trlx` ist die Java-Klasse, die das Parsing übernimmt.
- `c:\xmlex\dtdv.xml` ist die zu prüfende Datei.

Schrittweise DTD-Auswertung mit XML für Java

Verwenden Sie dieselbe Datei wie in Listing 9.1. Hier die empfangenen Fehlermeldungen:


```
dtdv.dtd: 8, 30: Spaces are expected.
dtdv.dtd: 8, 30: '#REQUIRED' or '#IMPLIED' or '#FIXED' or
➥attribute value is expected.
dtdv.dtd: 21, 35: This content model is not matched with
➥the mixed model '(#PCDATA¦FOO¦BAR¦...¦BAZ)*': '(#PCDATA¦shortcut)+'
dtdv.dtd: 24, 29: 'CDATA' or 'ID' or 'IDREF' or 'IDREFS'or
➥'ENTITY' or 'ENTITIES' or 'NMTOKEN' or 'NMTOKENS' or 'NOTATION'
➥or '(' is expected.
dtdv.dtd: 27, 24: Element 'tip' is already declared.
dtdv.dtd: 31, 27: NOTATION 'bmp' is not declared.
dtdv.dtd: 31, 33: NOTATION 'gif' is not declared.
c:\xmlex\dtdv.xml: 3, 3: The document has no element.
```

Jede Fehlermeldung besteht aus den folgenden Komponenten:

- Die Datei, in der der Fehler aufgetreten ist
- Die Zeilennummer
- Die Zeichenposition, an der der Fehler erkannt wurde
- Eine Beschreibung

Mit XML für Java erhalten Sie alle Fehler in einem Durchlauf mit klaren, verständlichen Fehlermeldungen.

Die letzte Fehlermeldung hat damit zu tun, daß kein vollständiges Dokument vorliegt, sondern nur ein Zeiger auf die DTD.

Mit XML für Java ist es möglich, die DTD direkt zu überprüfen. Dazu fügen Sie in der Befehlszeile den Parameter `-dtd` ein und lesen die Datei mit der externen Untermenge direkt:

```
jre -cp c:\xml4j\xml4j.jar trlx -dt c:\xmlex\dtdv.dtd
```

Eine viel sauberere Vorgehensweise! Natürlich erhalten Sie dieselben Ergebnisse, außer für den letzten Fehler, der darauf hinweist, daß das Dokument kein Element enthält.

Auswertung Ihrer XML-Dateien mit DXP

Geben Sie an der DOS-Eingabeaufforderung folgendes ein:

```
jre -c .;c:\datachannel\dxp\classes dxpcl -s -v c:\xmlex\wfq.mxl
```

Dabei gilt:

- `jre` ruft die Java Laufzeit-Engine auf.
- `-cp` setzt den Klassenpfad (wo die verwendeten Klassen abgelegt sind). Hier wurden zwei Pfade angegeben: Der erste (.) bezieht sich auf das aktuelle Arbeitsverzeichnis, der zweite ist `c:\datachannel\dxp\classes`;. Sie werden durch ein Semikolon (;) voneinander abgetrennt.
- `dxpcl` ist der Name des Java-Programms (Klasse).
- `-s` steht für Silent Mode.

- `-v` steht für `Validation on`, also die Aktivierung der Auswertung.
- `c:\xmlex\wfq.xml` ist die zu überprüfende Datei

Schrittweises Durchlaufen einer XML-Datei mit DXP

In Kapitel 5 haben Sie die Datei mit Hilfe des Parsers wohlgeformt gestaltet. Diese Datei (siehe Listing 9.6) enthält eine Beschreibung eines Hilfethemas.

***Listing 9.6:** `wfq.xml` – die wohlgeformte Hilfedatei*

```
1: <?xml version="1.0" ?>
2: <?protext objid="I5678" ?>
3: <!DOCTYPE helptopic [
4: <!ENTITY doubleclick "Double-click">
5: ]>
6: <helptopic>
7: <title keyword="printing,network;printing,shared printer">How to
➥use a shared network printer?</title>
8: <procedure>
    <step><action>In <icon>Network Neighborhood</icon>, locate
➥and double-click the computer where the printer you want to
➥use is located. </action>
9: <tip targetgroup="beginners">To see which computers have shared
➥printers attached, click the <menu>View</menu> menu,
10: click <menu>Details</menu>, &amp; look for printer names or
➥descriptions in the Comment column of the Network
➥Neighborhood window.</tip>
11: </step>
12: <step>
13: <action>&doubleclick; the printer icon in the window
➥that appears.</action>
14: </step>
15: <step>
16: <action>
17: To set up the printer, <xref linkend="id45">follow the instructions
➥</xref> on the screen.
18: </action></step>
19: </procedure>
20: <rule form="double"/>
21: <tip>
22: <p>After you have set up a network printer, you can use it as
➥if it were attached to your computer.
➥For related topics, look up &quot;printing&quot; in the Help Index.
```

```
23: </p>
24: </tip>
25: </helptopic>
```

Früher in diesem Kapitel haben Sie eine DTD korrigiert, die die Struktur eines Hilfethemas beschrieb (siehe Listing 9.3). Jetzt wollen Sie beides verbinden. Dazu verweisen Sie in der DOCTYPE-Deklaration von `wfq.xml` auf `dtdv.dtd`:

```
<!DOCTYPE helptopic SYSTEM "dtdv.dtd" []>
```

Die Änderungen sehen Sie in Listing 9.7.

***Listing 9.7:** `wfq.xml` – jetzt wird in der DOCTYPE-Deklaration auf die DTD verwiesen*

```
1: <?xml version="1.0" ?>
2: <?protext objid="I5678" ?>
3: <!DOCTYPE helptopic SYSTEM "dtdv.dtd" [
4: <!ENTITY doubleclick "Double-click">
5: ]>
6: <helptopic>
7: <title keyword="printing,network;printing,shared printer">How
➥to use a shared network printer?</title>
8: <procedure>
9: <step><action>In <icon>Network Neighborhood</icon>, locate and
➥double-click the computer where the printer you want to use is
➥located. </action>
10: <tip targetgroup="beginners">To see which computers have shared
➥printers attached, click the <menu>View</menu> menu,
11: click <menu>Details</menu>, &amp;amp; look for printer names or
➥descriptions in the Comment column of the Network Neighborhood
➥window.</tip>
12: </step>
13: <step>
14: <action>&doubleclick; the printer icon in the window that
➥appears.</action>
15: </step>
16: <step>
17: <action>
18: To set up the printer, <xref linkend="id45">follow the
➥instructions</xref>   on the screen.
19: </action></step>
```

```
20: </procedure>
21: <rule form="double"/>
22: <tip>
23: <p>After you have set up a network printer, you can use it as
➥if it were attached to your computer. For related topics,
➥look up &quot;printing&quot; in the Help Index.
24: </p>
25: </tip>
26: </helptopic>
```

Jetzt beginnen wir mit der Überprüfung:

```
jre -cp .;c:\datachannel\dxp\classes dxpcl -s -v c:\xmlex\wfq.xml
```

Wir erhalten das folgende Ergebnis:


```
ERROR: Invalid content : procedure
Possible: rule
Location: file:/c:/xmlex/wfq.xml:8:2

ERROR: element not declared in DTD "rule"
Location: file:/c:/xmlex/wfq.xml:21:2

ERROR: attribute hasn't been declared in the DTD "form"
Location: file:/c:/xmlex/wfq.xml:21:7

FATAL ERROR: java.lang.NullPointerException:
Location: :0:0
```

Welchen definierten Inhalt hat Ihr Hilfethema? Ihr Hilfethema muß mit einer Überschrift beginnen, gefolgt von einer Linie und dann der Prozedur. Das ist in Ihrem Dokument nicht der Fall.

Als erstes fügen wir Ihrem Dokument die Linie hinzu:

```
7: <title keyword="printing,network;printing,shared printer">How to
➥use a shared network printer?</title><rule/>
```

`rule` ist ein leeres Element. Beachten Sie dazu die folgende spezielle Syntax:

```
<rule></rule>
```

oder

```
<rule/>
```

Jetzt führen Sie Ihren Parser noch einmal aus:


```
ERROR: element not declared in DTD "rule"
Location: file:/c:/xmlex/wfq.xml:8:2

ERROR: element not declared in DTD "rule"
Location: file:/c:/xmlex/wfq.xml:21:2

ERROR: attribute hasn't been declared in the DTD "form"
Location: file:/c:/xmlex/wfq.xml:21:7

FATAL ERROR: java.lang.NullPointerException:
Location: :0:0
```

Die Spezifikation definiert die folgenden Gültigkeitsbeschränkungen:

- Ein Element ist gültig, wenn es eine Deklaration dafür gibt.
- Ein Attribut muß deklariert worden sein.

Ist das nicht der Fall, liegen Fehler vor. Fügen Sie also Deklarationen für die fehlenden Elemente und Attribute ein:

```
<!ELEMENT rule EMPTY>
<!ATTLIST rule form (single ¦ double ¦ dotted) "single">
```

Wenn Sie jetzt den Parser noch einmal ausführen, erhalten Sie folgendes:


```
ERROR: Invalid content : p
Possible: iconbmp, icon, menu, #PCDATA, , xref
Location: file:/c:/xmlex/wfq.xml:23:2

ERROR: element not declared in DTD "p"
Location: file:/c:/xmlex/wfq.xml:23:2

ERROR: unknown ID referred "id45"
Location: file:/c:/xmlex/wfq.xml:26:14

Found errors/warnings: 0 fatal error(s), 3 error(s) and 0 warning(s)
```

Der Tip hinter der zweiten Linie enthält ein `p`, das in der DTD nicht erlaubt ist. Wir entfernen das Start- und Ende-Tag für `p` innerhalb des Elements `tip` aus `wfq.xml`.


```
ERROR: unknown ID referred "id45"
Location: file:/c:/xmlex/wfq.xml:25:14

Found errors/warnings: 0 fatal error(s), 1 error(s) and 0 warning(s)
```

`IDREF`-Werte müssen mit dem Wert eines `ID`-Attributs im XML-Dokument übereinstimmen, was hier nicht der Fall ist. Es gibt in Ihrem Dokument kein Element mit einem Attribut der Typ-ID mit dem Wert `id45`.

Entfernen Sie das Element `xref`, so daß Sie ein gültiges Dokument erhalten.

Auswerten Ihrer XML-Dateien mit XML für Java

Geben Sie an der DOS-Eingabeaufforderung folgendes ein:

```
jre -xp x:\xml4j\xml4j.jar trlx c:\xmlex\wfq.xml
```

Dabei gilt:

- `jre` ruft die Java Laufzeit-Engine auf.
- `-cp` setzt den Klassenpfad. In diesem Fall verweisen Sie auf die `jar`-Datei (Java-Archivdatei) mit dem Namen `xml4j.jar`.
- `trlx` ist die Java-Klasse, die das Parsing übernimmt.
- `c:\xmlex\wfq.xml` ist die zu überprüfende Datei.

Schrittweise Prüfung einer XML-Datei mit XML für Java

Verwenden Sie die Datei aus Listing 9.5. Mit XML für Java erhalten Sie die folgenden Fehlermeldungen:


```
c:\xmlex\wfq.xml: 21, 22: Attribute 'form' of element 'rule'
➥is not declared.
c:\xmlex\wfq.xml: 21, 22: Can't find content model of '<rule>'.
c:\xmlex\wfq.xml: 24, 5: Can't find content model of '<p>'.
c:\xmlex\wfq.xml: 25, 7: Content mismatch in '<tip>'.  Content model is
''(#PCDATA¦icon¦menu¦xref¦iconbmp)*'.
c:\xmlex\wfq.xml: 26, 13: Content mismatch in '<helptopic>'.
➥Content model is '(title,rule,procedure,rule?,(tip¦warning¦note))'.
c:\xmlex\wfq.xml: 18, 45: ID 'id45' is not defined in the document.
```

Die folgenden Probleme wurden erkannt:

- Für das Element `rule` gibt es keine Element- und Attributlistendeklarationen.
- Im Element `tip` wurde ein `p` verwendet, das gemäß der DTD nicht erlaubt ist.
- Dem `title` muß gemäß der DTD ein `rule`-Element folgen.
- Das `xref`-Element verweist über sein Attribut `linkend` des Typs `IDREF` auf eine eindeutige Bezeichner-ID, die in dem XML-Dokument nicht existiert.

Zusammenfassung

In diesem Kapitel haben Sie zwei in Java geschriebene Parser kennengelernt und eingesetzt, DXP von DataChannel und XML für Java von IBM. Sie konnten damit die folgenden Aufgaben erledigen:

- Überprüfung der Deklarationen in der DTD.

Beachten Sie, daß die Behandlung der Deklarationen in der externen Untermenge im Vergleich zur internen Untermenge unterschiedlich ist.

- Prüfen, ob XML-Dokumente den in der DTD (Dokumenttypdefinition) definierten Regeln entsprechen.

F&A

F Warum sollte ich mich überhaupt anstrengen, gültige XML-Dateien zu produzieren?

A *Es ist eben viel einfacher, gültige XML-Dateien zu verwenden. Wenn Sie wissen, daß das Element* `tip` *nur in diesem oder jenen Kontext auftreten kann, brauchen Sie sich nur darum zu kümmern, was mit dem Element* `Tip` *in Ihrem Programm oder in Ihren Stylesheets passieren soll, weil der Kontext bekannt ist. Kann dagegen ein Tip an jeder beliebigen Stelle auftreten, müssen Ihre Programme oder Stylesheets wesentlich komplizierter und umfangreicher werden.*

F Achtet ein Browser darauf?

A *Nein. Ein Browser kann mit einer wohlgeformten XML-Datei arbeiten. Eine wohlgeformte Datei beinhaltet genügend Informationen für den Browser, um eine Baumstruktur anzulegen und die Datei darzustellen.*

Die Qualität der Darstellung hängt natürlich von den bereitgestellten und unterstützten Stilen ab.

F Gibt es andere Software, die das überprüfen könnte?

A *Authoring-Tools achten leider nicht immer auf die Gültigkeit. Wenn Sie eine neue XML-Datei bearbeiten, werden Sie von der Software gefragt, welche DTD Sie verwenden wollen. Anschließend führt Sie die Software durch den Bearbeitungsprozeß, indem nur die Tags bereitgestellt werden, die das Inhaltsmodell erlaubt. Bei den meisten Authoring-Tools können Sie auf diese Weise sichergehen, daß Sie wohlgeformte und gültige XML-Dateien haben.*

F Gibt es noch andere auswertende Parser?

A *Ja. Es gibt MSXML von Microsoft und Larval von Tim Bray bei Textuality, beide in Java geschrieben.*

Übungen

1. Für diese Übung brauchen Sie die folgende XML-Datei, `valq.xml`:

```
1: <!DOCTYPE references SYSTEM "docb.dtd" [
2: ]>
3: <refentry id="refentry.xref">
4:     <refmeta>
5:         <refentrytitle>delete_mark</refentrytitle>
6:     </refmeta>
7:     <refnamediv>
8:         <refname>delete_mark</refname>
9:         <refpurpose>Deletes the currently
➥selected region and puts it in the specified paste buffer
➥</refpurpose>
10:     </refnamediv>
11:     <refsynopsisdiv>
12:         <title>Synopsis</title>
13:         <cmdsynopsis>
14:             <command>delete_mark</command>
15:             <arg><option>-append</option></arg>
16:             <arg><replaceable>buffername</replaceable></arg>
17:         </cmdsynopsis>
18:         <cmdsynopsis>
19:             <command>delete_mark</command>
20:             <arg>null</arg>
21:        </cmdsynopsis>
22:    </refsynopsisdiv>
23:    <refsect>
24:        <title>Description</title>
25:            <para>This command deletes the currently selected
```

```
➥region and puts it in the paste buffer specified.
➥If no buffer name is supplied, the current paste buffer is
➥used (see <citerefentry><refentrytitle>set paste=buffername
➥</refentrytitle></citerefentry>).
26: If the <literal><replaceable>buffername</replaceable>
➥</literal>is <literal>null</literal>, the selection is
➥deleted without copying it to a paste buffer.
27: This command corresponds to the <interface>Delete</interface>
➥menu item on the default <interface>Edit</interface> menu.
28: If the <option>-append</option> option is selected, text
➥is inserted at the end of the buffer rather than
➥completely replacing its contents.</para>29 <command>dm
➥</command> is a synonym for <command>delete_mark</command>.
30: </para>
31: </refsect>
32: <refsect>
33:    <title>Examples</title>
34:    <screen>Command: <userinput>dm</userinput></screen>
35:    <screen>Command: <userinput>dm bufA</userinput></screen>
36:    <screen>Command: <userinput>dm -append buf2</userinput>
➥</screen>
37:</refsect>
38:</refentry>
```

Außerdem brauchen Sie die folgende Datei, `docb.dtd`, auf die in `valq.xml` verwiesen wird:

```
1: <!ENTITY % in-line "option ¦ replaceable ¦ citerefentry
➥¦ refentrytitle ¦ literal ¦ userinput ¦ interface ¦
➥command ¦ arg" >
2: <!ENTITY % paracontent "(#PCDATA ¦ %in-line;)*">
3: <!ELEMENT refentry (refmeta?, refnamediv, refsynopsisdiv,
➥refsect+)>
4: <!ATTLIST  refentry id ID #REQUIRED>
5: <!ELEMENT refmeta (refentrytitle, refmiscinfo*) >
6:    <!ELEMENT refentrytitle (#PCDATA)>
7:    <!ELEMENT refmiscinfo (#PCDATA)>
8: <!ELEMENT refnamediv (refname, refpurpose)>
9:    <!ELEMENT refname (#PCDATA)>
10:    <!ELEMENT refpurpose %paracontent;>
11: <!ELEMENT refsynopsisdiv (title, cmdsynopsis+)>
12:     <!ELEMENT cmdsynopsis (command, arg+)>
13: <!ELEMENT refsect  (title, (para ¦ screen)+ )>
14:    <!ELEMENT title (#PCDATA)>
15:    <!ELEMENT para %paracontent; >
16:    <!ELEMENT screen %paracontent; >
17: <!ELEMENT option (#PCDATA)>
```

```
18: <!ELEMENT replaceable (#PCDATA)>
19: <!ELEMENT citerefentry (#PCDATA ¦ refentrytitle)*>
20: <!ELEMENT literal (#PCDATA)>
21: <!ELEMENT userinput (#PCDATA)>
22: <!ELEMENT interface (#PCDATA)>
23: <!ELEMENT command (#PCDATA)>
24: <!ELEMENT arg (#PCDATA ¦ option ¦ replaceable)*>
```

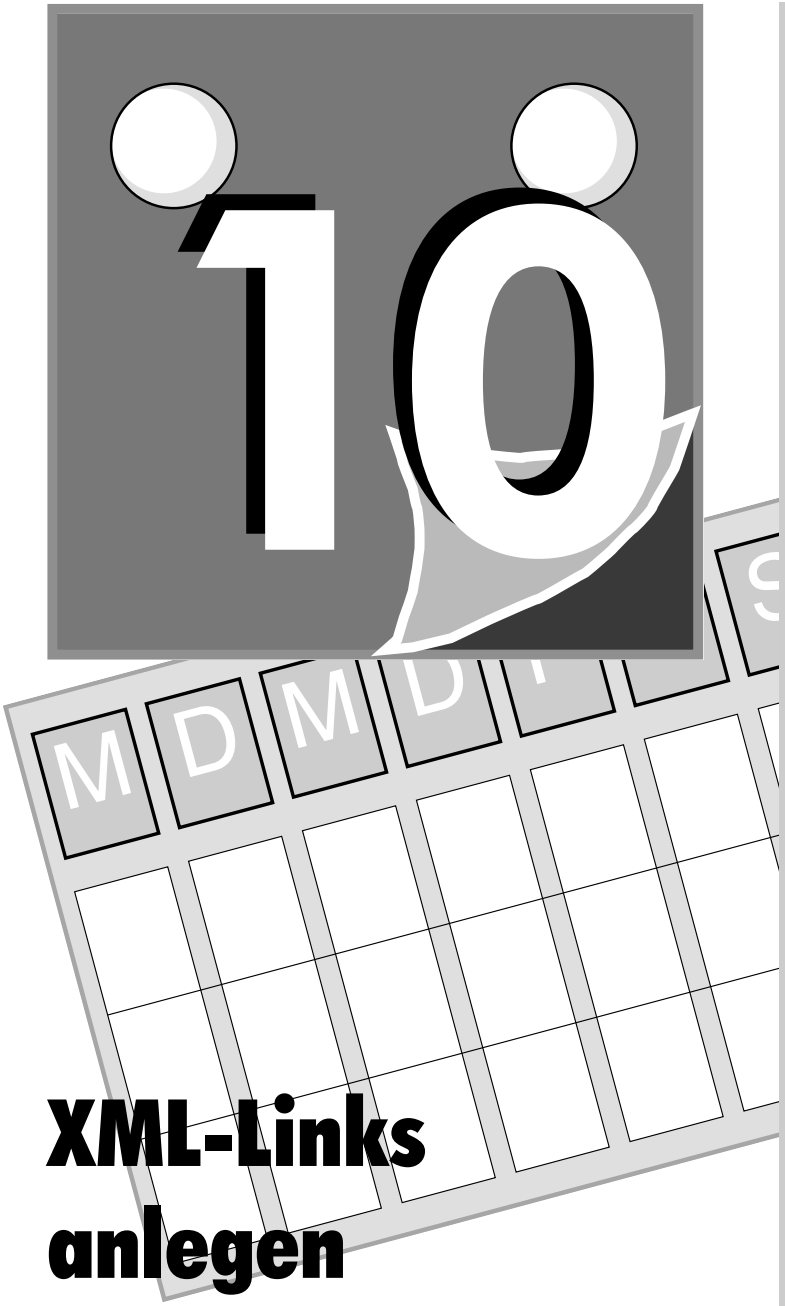
- Ermitteln Sie die Fehler und die Fehlermeldungen, die der Parser erzeugt.
- Prüfen Sie die Datei mit einem Parser Ihrer Wahl.
- Korrigieren Sie alle aufgetretenen Probleme.

2. Führen Sie anhand der XML-Datei, die Sie in den vergangenen Kapiteln angelegt haben, die folgenden Schritte aus:

- Führen Sie Fehler an.

Verwenden Sie als Referenz die W3C-Empfehlungen.

- Überprüfen Sie die Datei mit unterschiedlichen Parsern.
- Werten Sie die unterschiedlichen Ergebnisse der Parser aus.



XML-Links anlegen

Woche 2

Neben all den Spekulationen um XML als Sprache hat man zwei verwandten Vorschlägen recht wenig Achtung entgegengebracht, XLink und XPointer, die vielleicht sehr viel aufregender sind als die eigentliche XML-Sprache. Diese beiden Vorschläge, zusammengefaßt unter dem generischen Namen XML-Linking, stellen eine Revolution der Dokumentverknüpfung dar. (Im ersten Vorschlag wurden sie als gemeinsames XLL-Dokument (*XML Linking Language*) kombiniert).

Heute werden Sie die folgenden Dinge kennenlernen:

- Unterschiede zwischen HTML- und XML-Hyperlinks
- Verschiedenen Hyperlink-Typen von XML und wie sie für die Verknüpfung mehrerer Punkte, nur lesbarer Dokumente und zur Beschreibung von Verknüpfungen verwendet werden
- Einsatz der verschiedenen Attribute von Link-Elementen, um zu steuern, wie und wann Links aktiviert werden
- Zuordnung der Attribute von Elementen an existierenden XML-Dokumente, so daß ein Hyperlink-Verhalten eingeführt wird, das zuvor nicht vorhanden war

XLink und HTML werden in diesem Kapitel ständig miteinander verglichen, aber es ist nicht unbedingt erforderlich, daß Sie alle Details des HTML-Hyperlinkings verstehen. Wenn Sie jedoch zumindest eine Ahnung von HTML haben, werden Sie besser verstehen, wie aufregend die Möglichkeiten sind, die XLink uns eröffnet.

Hyperlinks

Jeder, der in seinem Leben eine Webseite geöffnet hat, weiß, wie Hypertext aussieht. Was den Hypertext von normalem Text unterscheidet, sind die Hyperlinks, die typischen blau unterstrichenen Textabschnitte, die Stellen kennzeichnen, die Sie durch einen Mausklick zu anderen Seiten bringen.

Gerade die Möglichkeit, Dokumente zu verknüpfen, brachte den großen Erfolg für das World Wide Web. Und sie war sicherlich eines der Entwurfsziele von HTML. (Bei CERN – dem Europäischen Atomforschungszentrum – in der Schweiz inspirierten die weitläufige Verteilung der Projekte und die gemeinsam genutzten Dokumente Tim Berners-Lee, nach einer Lösung zu suchen, die Dokumente miteinander zu verknüpfen, unabhängig von ihrer physischen Position im Computernetzwerk.)

Ein Hyperlink (kurz: Link) ist eine Zuordnung zwischen zwei Textpassagen, zwischen einem Text und einem Objekt (wie beispielsweise Grafiken, die in Webseiten eingebunden werden, indem eine Verknüpfung mit der Bilddatei angelegt wird) oder sogar

zwischen einem Objekt und einem Textabschnitt (wie beispielsweise dem Bildzuordnungsmechanismus in HTML, mit dem es möglich ist, daß der Link abhängig davon, wo auf dem Bild Sie klicken, auf unterschiedliche Dokumente verweist).

In der offiziellen Terminologie des XLink-Vorschlags ist ein Link »eine explizite Beziehung zwischen zwei oder mehr 10Datenobjekten oder Teilen von Datenobjekten«. Der Einfachheit halber geht es in diesem Kapitel nur um Links zwischen XML-Elementen, unabhängig von ihrem Inhalt.

Die Standards sprechen auch über Link-Ressourcen, Objekte, die an den Links beteiligt sind. Ich bevorzuge das weniger exakte, aber überschaubarere Konzept eines Ziels, auch wenn die Einführung eines Links in XML viel mehr als die einfache Link-Verfolgung von HTML bedeuten kann.

In den einfachen Begriffen des HTML-Linkings ausgedrückt, ist ein Link die Zuordnung zwischen einer Quelle und einem Ziel. Das Ziel kann eine vollständige HTML-Seite sein, so daß die Beschreibung des Ziels (der »Locator«) eine URI (*Universal Resource Identifier*) ist, oder ein benanntes Element innerhalb einer HTML-Seite, die durch das Symbol # gekennzeichnet wird, einen Fragmentbezeichner, gefolgt von dem Wert des `NAME`-Attributs des Zielelements. In der morgigen Beschreibung der XPointer-Sprache erhalten Sie detailliertere Informationen zur Verwendung von Fragmentbezeichnern in XML-Dokumenten.

Der HTML-Code für die Quelle des Links würde irgendwie wie folgt aussehen:

```
<H3>Einfaches Linking</H3>
<P>Dies <A HREF="http://writer.xs4all.nl/simplelink.html#S15">link</A>
zeigt auf einen bestimmten Abschnitt im Dokument. </P>
```

HTML kann ohne weiteres über die Quellen und Ziele von Links sprechen, weil seine Linking-Mechanismen so einfach sind. Die Linking-Sprache von XML, XLink, ist so viel leistungsfähiger als die von HTML, daß man hier nicht mehr einfach von Quellen und Zielen sprechen kann. XLink spricht statt von Quellen von *Linking-Elementen* – Links können in zwei Richtungen erfolgen, das Ende eines Links könnte also sowohl Quelle als auch Ziel sein. Und statt von Zielen spricht man von *Ressourcen* – eine Ressource könnte ein Datenabschnitt sein, das Ergebnis einer Datenbankabfrage oder ein externer Link, der als Zwischenglied zu dem endgültigen Ziel dient.

Wenn Sie in einer HTML-Seite auf einen Link klicken, wird eine neue Seite geöffnet (obwohl es sich auch um ein neues Fenster oder einen neuen Frame handeln könnte). In diesem einfachen Szenario könnte man sagen, ein Link wird *verfolgt*, weil Sie den Eindruck erhalten, von einem Dokument zu einem anderen zu reisen. Wie Sie später erfahren werden, erlaubt die XLink-Syntax die Angabe mehrerer oder gruppierter

Link-Lokatoren, und die Methode, wie sie verarbeitet werden, kann mehr ein Implementierungsproblem (Browser-Software) denn ein linguistischer Aspekt sein. Aus diesem Grund spricht XLink vom *Traversieren* eines Links.

Nachdem wir die Terminologiefragen geklärt haben, wollen wir die Praxis betrachten.

Lokatoren (Locators)

XML-Links arbeiten mit Link-Elementen. Die Link-Elemente wiederum enthalten Lokatoren in Form von Attributen oder anderen Elementen, die auf bestimmte Positionen innerhalb einer Ressource zeigen.

Im allgemeinen ist ein Lokator eine URI, ein Fragmentbezeichner, oder eine URI kombiniert mit einem Fragmentbezeichner. Lokatoren für XML-Dokumente sind erweiterte Zeiger (XPointer), die Sie in Kapitel 11 kennenlernen werden.

Die Syntax der Lokatoren erlaubt Ihnen die folgenden Variationen:

- `URI#fragment` – Bezieht sich auf die ganze durch die URI bezeichnete Ressource und extrahiert dann den Teil, der durch den Fragmentbezeichner angesprochen wird.
- `URI¦fragment` - Die Anwendung kann entscheiden, wie sie die URI verarbeitet, um die Ressource zu extrahieren. Das könnte beispielsweise genutzt werden, um nur einen bestimmten Teil eines Dokuments zu laden statt das vollständige Dokument.

Handelt es sich bei dem Fragmentbezeichner um eine Zeichenkette, die den XML-Regeln für einen Namen entspricht, wird die Zeichenkette als Wert des `id`-Attributs eines XML-Elements behandelt. Der Lokator `afile.html#ht7` würde also auf ein Element in der Datei `afile.html` verweisen, deren `id`-Attributwert gleich `ht7` ist (beispielsweise `<para id="ht7">`). Diese eingebaute Interpretation soll Sie ermutigen, die ID-Adressierung zu verwenden, die sinnvoll, weil völlig eindeutig, ist. Bei dieser Adreßform besteht aber auch das Risiko einer Verwirrung, falls ihre Software nicht sicherstellt, daß alle Element-ID-Werte innerhalb eines Dokuments eindeutig sind.

Link-Elemente

In HTML können nur zwei Elemente als Links dienen: `A` (Anker) und `IMG` (Bild). In XML ist ein Link ein Link, wenn das Link-Element sagt, daß das so sei. Die Existenz des Links wird durch die Elementattribute gekennzeichnet. Das bedeutet, jedes XML-Element kann als Linking-Element agieren, wenn Sie nur sicherstellen, daß es die rich-

tigen Attribute hat. Später in diesem Kapitel werden Sie erfahren, daß Sie Attribute auch neu zuordnen können. Hier konzentrieren wir uns auf das Konzept, daß eine bestimmte Menge von XLink-Attributen einen Link kennzeichnet und beschreibt.

Das erste Attribut, das ein Element als Link-Element kennzeichnet, dessen Deklaration in einer XML-DTD etwa wie folgt aussehen würde (vergessen Sie nicht, daß ein Element, das als Link-Element deklariert ist, auch der Struktur entsprechen muß, die Sie in der DTD festlegen wollen):

```
<!ELEMENT CORRELATION ANY>
<!ATTLIST CORRELATION
      xlink:form CDATA #FIXED wert>
```

Hier ist `wert` entweder `simple` oder `extended`. (Der Wert könnte auch `locator`, `group` oder `document` sein, aber dabei handelt es sich nicht um eigenständige Linking-Elemente, sondern um verwandte Elemente.) Die Bedeutung dieser Werte wird in den folgenden Abschnitten beschrieben.

Die aktuelle XLink-Arbeitsskizze (3. März 1998) beschreibt die Verwendung des Attributs `xml:link`. Seit der Veröffentlichung dieses Vorschlags hat der XML-Namensraumvorschlag die Verwendung des Präfix `xml:` als nicht empfehlenswert beschlossen. In einer E-Mail vom 5. Mai 1998 hat Eve Mailer, eine der Verfasserinnen der XLink- und XPointer-Vorschläge, darauf hingewiesen, daß dieses Attribut in der nächsten Version der Skizze in `xlink:form` umbenannt wird, um Probleme mit dem Namensraum zu vermeiden. Das Veröffentlichungsdatum der nächsten Version steht noch nicht fest, aber ich habe die geänderte Syntax in dieser Beschreibung bereits berücksichtigt.

Einfache Links

Einfache Links erinnern stark an den einfachen HTML-Linking-Mechanismus, den Sie heute bereits kennengelernt haben. Einfache Links verwenden nur einen Lokator und arbeiten nur in einer Richtung:

```
<simple.link xlink:form="simple"
href="http://me.com/title.xml">siehe auch</simple.link>
```

Dieses einfache Link-Element enthält einen Textabschnitt, der als Ressource dient, ein Ende des Links. Diese Art Link wird auch als Inline-Link bezeichnet. Morgen lernen Sie mehr über Inline-Links und Out-of-Line-Links.

Verwechseln Sie den Begriff Inline-Link nicht mit Links, die innerhalb des aktuellen Dokuments enthalten sind – internen Links –, und denen, die auf andere Dokumente verweisen – externen Links. Ein Inline-Link ist einfach ein Link mit einem Textabschnitt, der als ein Link-Ende agiert (wie die A-Elemente in HTML).

Erweiterte Links

In einfachen Links bietet XLink nicht mehr als das grundlegende Linking, wie Sie es von HTML-Dokumenten her gewohnt sind (obwohl es auch hier einige wichtige Verbesserungen gab). In erweiterten Links erscheint die wahre Stärke von XLink und damit XML. Selbst wenn nichts anderes darüber gesagt werden könnte, ist es noch aufregend genug, daß sie nicht physisch in einer der XML-Dateien enthalten sein müssen, von denen der Link kommt oder wohin er geht. Erweiterte Links erlauben XML-Dokumenten folgendes:

- Verknüpfung von beliebig vielen Ressourcen, was zu mehreren Zielen führt, anstelle einer einfachen 1:1-Beziehung, wie wir sie aus HTML kennen.
- Verknüpfung mit Ressourcen, die die Links nicht selbst enthalten können. Unter anderem handelt es sich dabei um Grafikdateien, Sounddateien, nur lesbare Dokumente usw., deren Codierung nicht erlaubt, daß Sie sie so abändern, daß Links eingebettet werden können.
- Aktivierung der (dynamischen) Filterung, das Hinzufügen und das Abändern von Links. Hier könnte man sich vorstellen, die Links an einer bestimmten Stelle anzupassen, so daß etwa erfahrene Leser eines technischen Handbuchs einen anderen Pfad einschlagen als Anfänger.

Beachten Sie, daß die Entwicklung von XLink und XPointer weitergeht und es sich dabei immer noch um Skizzen handelt. Leider bedeutet das, daß der Vorschlag in den wichtigsten Web-Browsern noch nicht implementiert wurde (außer der sehr einfachen Unterstützung von XLink im Mozilla-Quellcode von Netscape) und daß es höchst unwahrscheinlich ist, daß er implementiert wird, bevor er nicht stabiler ist.

In anderen Paketen gibt es jedoch schon erste Ansätze der Unterstützung. Im Softwarepaket HyBrick (`http://collie.fujitsu.com/hybrick`) gibt es eine experimentelle Implementierung von Teilen von XPointer im Public-Domain-Jade-DSSSL-Prozessor (`http://www.jclark.com`), und Teile davon werden auch bereits im Viewer/Editor-Paket IndelvIT implementiert (`http://www.indelv.com`).

Ein erweiterter Link zeigt eigentlich auf nichts und verknüpft auch nichts. Ein erweitertes Link-Element wird durch den Attributwert `xlink:form` gekennzeichnet und enthält eine Menge von Lokator-Elementen, die in ihrer Gesamtheit den erweiterten Link ausmachen:

```
<comment xlink:form="extended">
      <opinion xlink:form="locator" href="note4"/>
      <reference href="#section2.1"/>
      <reference href="http://here.com/appB.html"/>
      <reference href="references.htm"/>
</comment>
```

Wie bei den anderen Link-Elementen wird die Art des erweiterten Link-Elements durch das Attribut `xlink:form` angegeben. Hier deklariert sich ein `comment`-Element selbst als erweiterter Link, und ein `opinion`-Element deklariert sich als Lokator-Element. Wie Sie sehen, kann das im Start-Tag des Elements erfolgen, ohne daß dafür eine DTD erforderlich ist. Die Deklarationen für diese Elemente in einer DTD könnten wie folgt aussehen:

```
<!ELEMENT comment ANY>
<!ATTLIST comment
      xlink:form  CDATA #FIXED "extended">

<!ELEMENT opinion ANY>
<!ATTLIST opinion
      xlink:form  CDATA #FIXED "locator">

<!ELEMENT reference ANY>
<!ATTLIST reference
      xlink:form  CDATA #FIXED "locator">
```

Beachten Sie, daß ich für diese Elemente `ANY` als Inhaltsmodell verwendet habe, so daß innerhalb der Elemente beliebige Elemente enthalten sein können.

Vergessen Sie nicht, daß, wenn Sie das XML-Dokument auswerten, die Elemente für das Linking mit den in der DTD definierten strukturellen Regeln übereinstimmen müssen.

Wo erweiterte Link-Elemente Lokator-Elemente enthalten, die wiederum selbst Elemente enthalten, werden nur die eigentlichen Lokator-Elemente als Link-Elemente betrachtet, nicht die Elemente.

Wie würde das in der Praxis aussehen? Wir betrachten zwei Abschnitte in einem XML-Dokument, die gegenseitig aufeinander verweisen. Die einfachste Methode ist es, einen einfachen Link in jeden dieser Abschnitte einzufügen, der auf den anderen Abschnitt zeigt, wie in Listing 10.1 gezeigt.

Listing 10.1: Kreuzverweise mit Hilfe einfacher Links

```
1: <para id="idea1"><xref href="idea2">A first
2: impression of the subject matter
3: would inevitably lead to the
4: conclusion that ... </xref></para>
5: <para id="idea2"><xref href="idea1">Contrary
6: to our earlier conclusion, it
7: would appear that the evidence points
8: to a different ... </xref></para>
```

Listing 10.1 zeigt eine direkte Lösung für Kreuzverweise zwischen den beiden Abschnitten; aber was passiert, wenn Sie das Ganze erweitern und einen dritten Punkt ins Spiel bringen wollen? Im besten Fall müßten Sie nur ein paar müßige Wiederholungen bei der Link-Bearbeitung ausführen; im schlimmsten Fall müßten Sie den gesamten Markup-Code völlig neu überdenken.

Erweiterte Links und eine große Anzahl erweiterter Links werden als Out-of-Line-Links bezeichnet (was später in diesem Kapitel noch erklärt wird), erlauben assoziative Links wie diese, die sehr viel einfacher verwaltet werden können. Plazieren Sie diese Links an irgendeiner anderen Stelle im Dokument, vielleicht am Anfang, damit sie leichter zu finden und zu aktualisieren sind, wie in Listing 10.2 gezeigt.

In einem früheren Kapitel haben Sie gelernt, wie man die DTD in einer externen Datei ablegt. Später haben Sie erfahren, wie man ein XML-Dokument ganz einfach in Teile zerlegt, die man mit Hilfe von Entities kombiniert. Später in diesem Kapitel lernen Sie, wie man Textabschnitte physisch in ein Dokument einbindet. Der Vorschlag der modularen Dokumenterstellung sollte Ihnen bekannt vorkommen. Diese XML-Funktionen eröffnen uns Möglichkeiten der einfachen Dokumentverwaltung und der Konstruktion, die weit über alles nur entfernt mögliche in HTML hinausragen.

Listing 10.2: Kreuzverweise mit erweiterten Links

```
1: <argument xlink:form="extended">
2:    <theme xlink:form="locator« href="idea1">
3:    <theme xlink:form="locator« href="idea2">
4:  </argument>
.
.
5:  <para id="idea1">A first impression of the subject matter 6: would inevitably
lead to the conclusion that ... </para>
.
.
```

```
7:  <para id="idea2">Contrary to our earlier conclusion, it 8:would appear that
the evidence points to a different ... </para>
```

Wie sich ein Viewer verhält, wenn Sie auf eine der in Listing 10.2 gezeigten erweiterten Link-Ressourcen klicken, oder wie er überhaupt die Existenz erweiterter Links erkennt, ist ein Problem der Anwendung und wird von der XLink-Spezifikation nicht berücksichtigt. Man könnte jedoch ein spezielles Symbol anzeigen, das darauf hinweist, daß es noch weitere Materialien gibt, oder es könnte ein Menü aufgeklappt werden, von dem Sie eines der Link-Enden auswählen. (Es gibt noch weitere deskriptive Attribute für Linking-Elemente, die diese Auswahl sinnvoller machen; mehr darüber später.)

In Abbildung 10.1 sehen Sie die Menüliste, die im Panorama SGML-Browser von SoftQuad angezeigt wird.

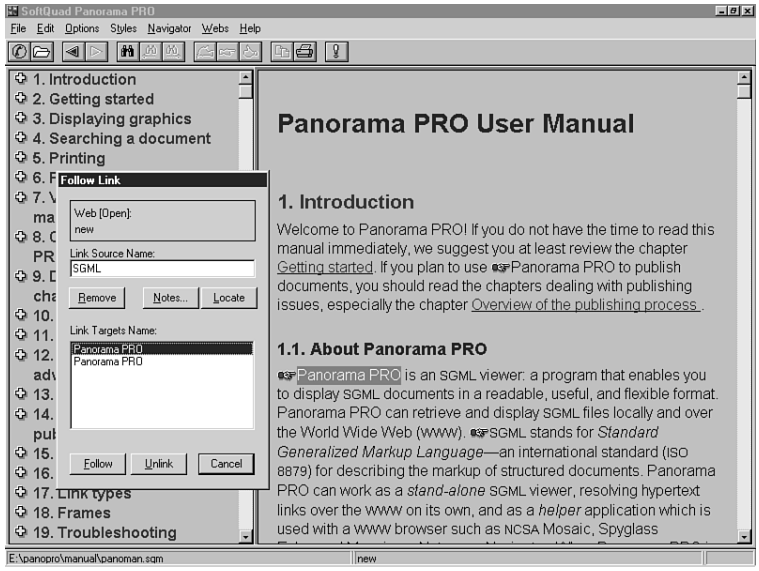


Abbildung 10.1: Auswahl eines Ziels für einen Link mit mehreren Zielen.

In Abbildung 10.1 verweisen die beiden Link-Enden neben dem Satz »Panorama Pro« (gekennzeichnet durch das Icon mit einer zeigenden Hand) auf das Link-Ende neben dem Wort »SGML«. Wenn Sie auf eines der Link-Enden von Panorama Pro klicken, gelangen Sie einfach zum Link-Ende SGML. Wenn Sie auf das SGML-Link-Ende klicken (in Panorama sind alle Links bidirektional), gibt es zwei Möglichkeiten. Panorama zeigt also eine Liste möglicher Link-Enden an, wo Sie das gewünschte auswählen können.

Es wäre interessanter, Ihnen Mehrfach-Links in einem echten XML-Paket zeigen zu können, aber es gibt leider noch keine allgemeine Unterstützung für XML-Links (neben der einfachen Unterstützung in Netscape Mozilla).

Panorama von SoftQuad ist ein SGML-Browser mit eingebetteter HyTime-Engine, die zahlreiche der komplexen HyTime-Linking-Funktionen unterstützt – einige derselben Funktionen, die XML von HyTime geerbt hat. Trotz der Tatsache, daß Panorama für die Verwendung mit SGML vorgesehen war, erledigt Panorama weiterhin seinen Job gut, XML-Code zu rendern, vorausgesetzt, der Code, den Sie laden, ist gültig und Sie driften nicht zu weit vom reinen SGML-Code ab. (Eine Testversion finden Sie unter `http://www.sq.com`.)

Interessanterweise wäre es ganz einfach, Links zwischen Elementen einzufügen, lange nachdem das Dokument fertiggestellt wurde, ohne irgendeinen Inhalt ändern zu müssen (insbesondere, wenn die Links nicht in dem beteiligten Dokument enthalten sind). Das wird Ihnen morgen sehr viel sinnvoller vorkommen, wenn Sie lernen, wie man mit Hilfe erweiterter Zeiger auf ganze Elementbereiche verweist.

Die Tatsache, daß Sie Links in ein Dokument einfügen können, nachdem es fertiggestellt wurde, bedeutet, daß Sie auch Links auf und von Teile(n) nur lesbarer Dokumente anlegen können, wie beispielsweise für Dokumente auf einer CD-ROM.

Erweiterte Link-Gruppen

Sie haben bereits erfahren, daß sich Links in externen Dokumenten befinden können; das wird mit Hilfe erweiterter Link-Gruppen realisiert. Wie ein erweiterter Link zeigt ein erweitertes Link-Gruppen-Element auf nichts und verknüpft auch nichts. Statt dessen enthält es mehrere Dokumentelemente, in denen jedes (durch eine URI gekennzeichnete) Dokument die Link-Ressourcen enthält:

```
<xternal.refs>
      <ref.doch href="http://here.com/biblio.html"/>
      <ref.doch href="lists.htm"/>
</xternal.refs>
```

Diese Element- und Attributdeklarationen könnten in der XML-DTD etwa wie folgt aussehen:

```
<!ELEMENT xternal.refs (ref.doc*)>
<!ATTLIST xternal.refs
      xlink:form   CDATA #FIXED "group"
      steps     CDATA #IMPLIED>
```

```
<!ELEMENT ref.doc EMPTY>
<!ATTLIST ref.doc
      xlink:form  CDATA #FIXED "document">
```

Wie bei so vielen der anderen XLink-Elemente sind auch hier keine Regeln formuliert, wie ein Browser oder ein Viewer sich verhalten soll, wenn er auf Link-Gruppen wie diese trifft. Mit den zusätzlichen deskriptiven Attributen jedoch, die später beschrieben werden, könnte ein Viewer eine Popup-Liste aller Zuordnungen eines bestimmten Typs anzeigen und Ihnen erlauben, eine oder mehrere davon auszuwählen.

Wenn Sie erweiterte Link-Gruppen verwenden, tragen Sie das Risiko, daß eines der Link-Dokument-Elemente auch eine erweiterte Link-Gruppe enthält. Was passiert, wenn diese erweiterte Link-Gruppe zurück auf das ursprüngliche Dokument zeigt oder auf ein Dokument, das wiederum eine neue erweiterte Link-Gruppe enthält? Es würde viele verknüpfte erweiterte Link-Gruppen brauchen, bis Sie in einer Situation angekommen wären, in der es unendliche Links und Link-Schleifen gibt. Um das zu verhindern, können Sie einen Wert für das `steps`-Attribut des Gruppenelements deklarieren. Dabei handelt es sich um einen numerischen Wert, der angibt, wie viele Schichten Verschachtelung Sie, der Autor, zulassen wollen.

Inline- und Out-of-Line-Links

In XLink kann ein Link inline oder out-of-line sein. Standardmäßig sind alle Link-Elemente inline (der Wert des `inline`-Attributs ist `true`). Sie brauchen also das Attribut nur dann zu deklarieren, wenn ein Link-Element out-of-line ist (`inline` ist `false`).

Es herrscht größere Verwirrung bezüglich der Bedeutung dieser Begriffe, insbesondere wenn das komplexere Konzept von Links innerhalb oder außerhalb eines Dokuments dabei in Betracht gezogen wird. Ob ein Link inline oder out-of-line ist, hat absolut nichts damit zu tun, ob er sich innerhalb des Dokuments befindet oder nicht.

Einfache Links sind im wesentlichen immer inline. Ein einfacher Out-of-line-Link ist denkbar, aber man kann sich kaum vorstellen, welchen Nutzen er haben sollte. Es wäre ein Link mit einem einfachen Ende, also ein Zeiger auf eine Position ohne jede weitere Assoziation ..., also niemandem von gesteigertem Nutzen.

Wichtig wird der Unterschied zwischen Inline- und Out-of-line-Links bei erweiterten Links. Listing 10.3 zeigt ein Beispiel für einen erweiterten Inline-Link in einem XML-Dokument.

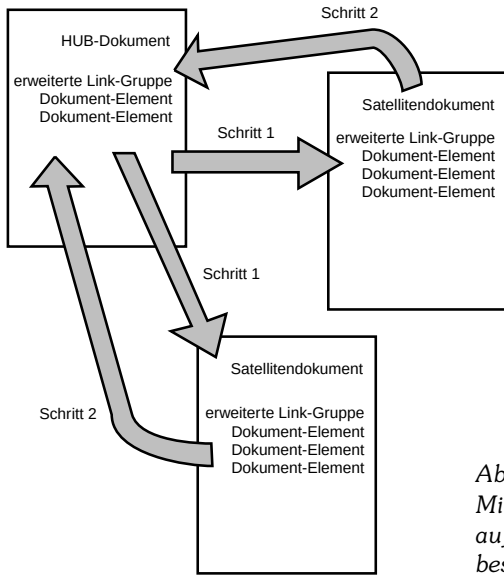


Abbildung 10.2:
Mit Hilfe von `steps` wird das Linking auf ein übergeordnetes Dokument beschränkt.

Listing 10.3: Ein Beispiel für einen erweiterten Inline-Link

```
1:  <para id="para4">    This is a
2:       <xref.list>psychosomatic
3:         <see.also href="med1.xml"/>
4:         <see.also href="http://www.medical.com/defs.htm"/>
5:         <see.also href="#para6">
6:       </xref.list>     condition, brought on by ...
7:  </para>
```

Hier enthält das Element `xref.list` Zeichendaten, die als Teil der durch die Links gebildeten assoziativen Spuren dienen. Das Link-Element enthält also, um in der offiziellen Terminologie der XLink-Spezifikation zu bleiben, eine lokale Ressource des Links.

Wenn Sie dieselbe Menge erweiterter Links noch einmal betrachten, diesmal als Out-of-line-Link formuliert, sollten Sie den Unterschied sofort erkennen, wie in Listing 10.4 gezeigt.

Listing 10.4: Verwendung eines erweiterten Out-of-line-Links

```
1:  <xref.list>
2:    <see.also href="#para4">
3:    <see.also href="med1.xml"/>
4:    <see.also href="http://www.medical.com/defs.htm"/>
```

```
5:    <see.also href="#para6">
6:  </xref.list>
.
.
.
7:  <para id="para4">This is a psychosomatic condition,
8: brought on by ... </para>
```

Wird der erweiterte Link als Out-of-line-Link formuliert, befindet er sich nicht mehr an derselben Stelle wie die Link-Ressourcen und enthält einen Zeiger auf die lokale Link-Ressource, nicht mehr die Link-Ressource selbst.

Link-Verhalten

Links in HTML-Dokument sind entweder passiv, wie `A`-Element-Links, die nichts tun, bis Sie darauf klicken, oder aktiv, wie das `IMG`-Element (das ein Bild hervorruft), das automatisch die Grafikdatei einfügt, auf die es verweist (normalerweise können Sie dieses Verhalten unterdrücken, indem Sie die Optionen im Web-Browser so setzen, daß Bilder nicht automatisch geladen werden). Das Link-Verhalten ist dem jeweiligen Element zugeordnet, und Sie können von der Anwenderseite aus nichts unternehmen, es zu ändern.

In XLink können Sie nicht nur steuern, wann ein Link aktiviert oder überquert wird, sondern auch, was passiert, wenn ein Link überquert wird. Dieses Link-Verhalten wird über die Attribute `show` und `actuate` geregelt.

Link-Effekte

In HTML wird ein Bild immer an der Stelle eingebettet, wo sein Ankerelement (`IMG`) auftritt. Andere Links sind einfach Sprünge zu neuen Dokumenten, entweder im selben Fenster, in einem neuen Fenster oder in einem Frame im aktuellen oder in einem neuen Fenster.

In XLink erlaubt Ihnen das `show`-Attribut, dieses Verhalten für alle Links zu formalisieren – statt sich auf einen Link mit einem bestimmten Namen zu verlassen –, und sogar noch ein bißchen mehr:

- `show="embed"` – Wenn der Link überquert wird, wird die vorgesehene Ressource an der Stelle eingebettet, wo das Überqueren begonnen hat. Damit können Sie beispielsweise virtuelle Dokumente anlegen, die sich aus Abschnitten aus anderen Dokumenten zusammensetzen. Oder, praktischer, Sie können Dinge wie dynamische Inhaltsverzeichnisse und Indizes anlegen, indem Sie Teile anderer Elemente aufnehmen und sie (scheinbar) physisch an die Link-Position kopieren.

- `show="new"` – Wird der Link überquert, wird die entsprechende Ressource in einem neuen Kontext angezeigt (oder verarbeitet). Das bedeutet normalerweise, daß die Ressource in einem neuen Fenster angezeigt wird. Wichtig dabei ist jedoch, daß das, was mit der Ziel-Ressource geschieht, die Anzeige der Ausgangs-Ressource nicht beeinflußt. Das könnte genutzt werden, um einen Textabschnitt aus einem externen Dokument einzubinden, ohne seine Stildeklarationen auf das aktuelle Dokument anzuwenden.
- `show="replace"` – Wenn der Link überquert wird, ersetzt die dafür bereitgestellte Ressource die Ressource, mit der das Überqueren begann. Das ist das Äquivalent zu einem HTML-Link, in dem die aktuelle HTML-Seite durch eine neue ersetzt wird.

Bei der Verwendung erweiterter Links ist es möglich, daß Sie einen Standardwert für das `show`-Attribut des erweiterten Link-Elements vorgeben, das jeden für die Lokator-Elemente in der DTD deklarierten Standardwert überschreibt. In dem folgenden Codeausschnitt beispielsweise hat das erste `location`-Element den Standardwert `replace` für das `show`-Attribut, aber das zweite `location`-Element den Wert `new` für das `show`-Attribut, der die Standarddefinition überschreibt:

```
<extended show="replace">
      <location href="mypage.tml"/>
      <location href="hispage.htm" show="new"/>
</extended>
```

Link-Timing

In HTML wird ein Bild immer automatisch an der Stelle eingebunden, wo sein Ankerelement (`IMG`) auftritt. Sie müssen nichts dazu tun, es sei denn, Sie haben im Web-Browser das automatische Laden von Bildern deaktiviert. Andere Links sind passiv, das heißt, es passiert überhaupt nichts, bis Sie darauf klicken.

In XLink erlaubt das Attribut `actuate`, diese Verhalten für alle Links zu formalisieren, statt sich dabei auf Links mit einem bestimmten Namen verlassen zu müssen, und Sie können damit festlegen, was passiert, wenn ein Link-Element angetroffen wird:

- `actuate="auto"` – Wenn eine der anderen Ressourcen desselben Links erkannt wird, wird diese Ressource automatisch geladen. Zusammen mit der Attributdeklaration `show="replace"` kann dieses Link-Verhalten beispielsweise genutzt werden, um den Viewer automatisch auf eine andere Seite weiterzuleiten.
- `actuate="user"` – Die Ressource wird erst geladen, wenn eine explizite Aufforderung erfolgt. Dies ist das Äquivalent zu einem HTML-Link, wobei nichts passiert, bis Sie auf die Quelle eines Links klicken.

Bei der Verwendung erweiterter Links ist es möglich, für das erweiterte Link-Element einen Standardwert für das `actuate`-Attribut zu deklarieren, der den für die Lokator-Elemente in der DTD deklarierten Standardwert überschreibt. In dem folgenden Codeausschnitt beispielsweise hat das erste `location`-Element den Standardwert `user` für das `actuate`-Attribut und das zweite `location`-Element den `actuate`-Attributwert `auto`, der den Standardwert überschreibt:

```
<extended actuate="user">
      <location href="mypage.html" show="embed"/>
      <location hrfe="hispage.htm" actuate="auto" show="embed"/>
</extended>
```

Mit diesem Code erzielen Sie womöglich nicht die Ergebnisse, die Sie erwartet hatten. Die erste Ressource (`mypage.html`) wird wie gewünscht geladen. Währenddessen wird jedoch auch die zweite Ressource (`hispage.html`, für die `actuate=auto` gesetzt ist) geladen! Da sie beide eingebettet sind, sehen Sie sie beide, aber mit unterschiedlichen Werten für das `show`-Attribut könnten Sie am Ende etwas ganz anderes sehen, als Sie zu Beginn angenommen hatten.

Wenn es mehrere eingebettete Links gibt und `actuation` gleich `auto` ist, erwarten Sie möglicherweise, daß eine Ressource aktiviert wird. Das passiert, aber die Aktivierung könnte durch eine zweite Aktivierung überdeckt werden. Das ist nicht weiter schlimm, aber Sie sollten sich auf die Probleme gefaßt machen, die durch die Deklaration dieses Link-Verhaltens verursacht werden können.

Das behavior-Attribut

Eine Anwendung kann mit Links alles mögliche tun. Sie kann spezielle Symbole anzeigen, Sounds abspielen, Anzeigeeigenschaften ändern ...; es gibt unendlich viele Möglichkeiten.

Im Internet Explorer 5 hat Microsoft eine Erweiterung der CSS-Stylesheet-Syntax implementiert, die dem `behavior`-Attribut sehr nahe kommt. Diese Erweiterung, als binäres Verhalten (Binary Behaviors) bezeichnet, erlaubt, einem HTML- oder XML-Element einen Abschnitt erweiterbaren Code anzufügen.

XLink beinhaltet ein Verhaltensattribut, das Ihnen erlaubt, das Verhalten für einen Link zu deklarieren. Der Inhalt und die Bedeutung jedes Werts, den Sie diesem Attribut zuordnen, und wie die Anwendung auf diesen Attributwert reagiert, bleibt dem Anwendungsentwickler überlassen.

Link-Beschreibungen

Eines der Mankos beim Linking in HTML-Dokumenten ist, daß es sich dabei um einen Alles-oder-nichts-Ansatz handelt. Entweder gibt es einen Link – oder es gibt keinen. Und alle Links sind genau gleich. Es gibt keine Möglichkeit, die Art der Beziehung zwischen zwei Link-Ressourcen zu beschreiben. Sie folgen einem HTML-Link, um plötzlich festzustellen, daß die Verknüpfung zwischen der Quelle und dem Ziel mehr in den Köpfen des Erzeugers vorhanden war als im Web-Browser.

Diese fehlende Möglichkeit, eine Link-Beschreibung bereitzustellen, wird noch kritischer, wenn es mehrere Ressourcen gibt, die an dem Link beteiligt sind, wie beispielsweise in erweiterten Links. Hier ist es sinnlos, mehrere Lokatoren anzubieten, wenn es keine wirkliche Methode gibt, die Beziehungen zu beschreiben oder zumindestens zwischen zwei parallelen Ressourcen zu unterscheiden.

XLink bietet mehrere Attribute an, mit deren Hilfe Sie Links und Link-Ressourcen beschreiben können:

- Ein Link kann ein `role`-Attribut haben, das die Bedeutung des Links für die Anwendung beschreibt. Ein Beispiel:

```
<extended role="bibliographies">
              <location href="myfile.htm"/>
              <location href="hisfile.html"/>
         </extended>
```

- Ist ein Link inline, kann eine lokale Ressource ein `content-role`-Attribut und ein `content-title`-Attribut haben. Das `content-role`-Attribut gibt an, welche Rolle diese Ressource im Link spielt (neben dem eigenen `role`-Attribut des Links). Das `content-title`-Attribut dient als Überschrift, die dem Benutzer die Rolle erklärt, die die Ressource im Link spielt. Ein Beispiel:

```
<mylink content-role="siehe auch" content-title="Referenz">
wie in meinem anderen Buch erklärt.</mylink>
```

- Eine entfernt gelegene Ressource kann ein `role`-Attribut und ein `title`-Attribut haben. Das `role`-Attribut kennzeichnet die Rolle, die die Ressource im Link spielt (neben dem eigenen `role`-Attribut des Links). Das `title`-Attribut dient als Überschrift, das den Benutzern die Rolle erklärt, die die Ressource im Link spielt. Ein Beispiel:

```
<extended role="bibliographies">
   <location href="myfile.htm"
     role="reference" title="This Chapter"/>
   <location href="hisfile.html"
     role="reference" title="Whole Book"/>
   </extended>
```

Die Deklaration von Rollen und Titeln für Links kann den scheinbar unlogischen Verknüpfungen, die man so oft in HTML-Dokumenten findet, Sinn geben. Sie könnten sich dazu vorstellen, Sie lesen einen Auszug aus einem bestimmten Buch und wollen nur die Links auf Dokumente über denselben Autor anzeigen oder die, die zwischen bestimmten Datumswerten geschrieben wurden!

Mozilla und das role-Attribut

Ein Beispiel für die Verwendung des `role`-Attributs finden Sie im aktuellen Release im Mozilla-Code von Netscape (der irgendwann zu Netscape Communicator 5 werden soll). Mozilla verwendet `role="HTML"`, um verknüpfte HTML-Dokumente in das aktuelle HTML-Dokument einzubetten, wie in Listing 10.5 (dem Basisdokument) und Listing 10.6 (dem »eingebundenen« Dokument) gezeigt. Technisch ausgedrückt, handelt es sich beim Importieren durch Linking um eine sogenannte *Transklusion*.

Listing 10.5: Mozilla-Code, um ein HTML-Dokument einzubetten

```
1: <?xml version="1.0"?>
2:   <page>
3:     <section>
4:       This text comes from the original (source) file.
5:     </section>
6:     <Foot XML-Link="LINK" Role="HTML"
7:        Show="EMBED" href="include.htm" />
8:     <section>
9:       This text also comes from the original file.
10:    </section>
11:  </page>
```

Listing 10.6: HTML-Code in dem Dokument, das eingebettet werden soll

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2:  <HTML>
3:    <HEAD>
```

```
4:      <META HTTP-EQUIV="Content-Type"
5:         CONTENT="text/html; charset=iso-8859-1">
6:      <META NAME="GENERATOR" CONTENT="Mozilla/5.0 [en]
7:        (WinNT; N ;Nav) [Mozilla]">
8:     <TITLE>Included Text</TITLE>
9:    </HEAD>
10:   <BODY>
11:     <HR>
12:     <H1>
13:     <FONT COLOR="#FF6666">Included Text</FONT></H1>
14:     <DIV color="red">
15:       <FONT COLOR="#FF6666">
16:         This text comes from a second file and is
17:         automatically embedded via the XLink
18:         mechanism.
19:       </FONT>
20:     </DIV>
21:     <HR>
22:   </BODY>
23: </HTML>
```

Die resultierende Ausgabe zeigt die nahtlose Kombination der beiden Dokumente (siehe Abbildung 10.3).

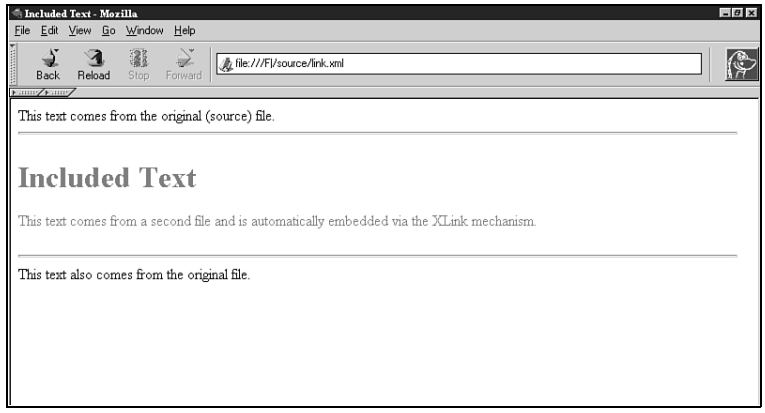


Abbildung 10.3: Texteinbindungen »by reference« in Mozilla.

Beachten Sie, daß die aktuelle Mozilla-Verison eine ältere Version des XLink-Vorschlags unterstützt, der damals noch als XML-Link oder XLL (*XML Link Language*) bezeichnet wurde. Die Kombination aus Groß- und Kleinbuchstaben (beispielsweise »Role« und »Show«), die ebenfalls aus einer früheren Version des Vorschlags stammt, muß genau wie in Listing 10.5 gezeigt verwendet werden, sonst funktioniert das Ganze nicht.

Neuzuordnung von Attributen

Früher in diesem Kapitel habe ich Ihnen bereits gesagt, daß es ganz einfach sei, Links in ein XML-Dokument einzufügen, nachdem Sie es fertiggestellt haben.

Wie Sie wissen, kann jedes Element einen Link darstellen. Es wird über das Attribut `xlink:form` als Link eines bestimmten Typs deklariert. Die Namen der Link-Elemente können zwar frei gewählt werden, aber die Namen ihrer Attribute sind vordefiniert – `actuate`, `bahavior`, `content-role`, `content-title`, `href`, `inline`, `role`, `show`, `steps` und `title`.

Angenommen, Sie wollen in einem XML-Dokument existierende Elemente als Links verwenden. Das ist ganz einfach, indem man erweiterte Links verwendet, die sich in einem externen Dokument befinden. Die Probleme beginnen, wenn die Elemente in diesem Dokument bereits Attribute mit denselben Namen wie die XLink-Attribute haben. Sie umgehen sie, indem Sie das Attribut `xml:attributes` verwenden, um existierenden Attributnamen neue Attributnamen zuzuordnen.

Betrachten Sie die folgende Elementdeklaration:

```
<!ELEMENT book (front, body, back) >
<!ATTLIST book
     title     CDATA               #REQUIRED
     role     (single¦volume)     #IMPLIED>
```

Wollten Sie dieses Element als Link-Element verwenden, müßten die Attribute `title` und `role` neu zugeordnet werden. Dazu könnten Sie eine Deklaration in eine interne DTD-Untermenge stellen. (Sie wissen, daß eine DTD in eine interne und eine externe DTD-Untermenge unterteilt werden kann und daß die interne DTD-Untermenge Priorität besitzt.) Die Deklaration, die Sie in der internen DTD-Untermenge anlegen, könnte wie folgt aussehen:

```
<!ATTLIST book
   xlink:form     CDATA  #FIXED "simple"
   xml:attributes CDATA  #FIXED "title mytitle role myrole">
```

Dieser XML-Code im Dokument würde korrekt als einfacher Link erkannt:

```
<book title="Presenting XML" role="main" myrole="Reference"
mytitle="Other Books" href="http://www.amazon.com/search?Light+North"/>
```

Zusammenfassung

Dieses Kapitel hatte XLink zum Thema, das Bindeglied zu den Linking-Funktionen von XML. Sie haben erfahren, welche Link-Typen es gibt, wie man sie verwendet und welche Attribute sie haben können. In Kapitel 11 werden Sie mehr über erweiterte Zeiger (XPointer) erfahren und wie Sie sie einsetzen, um Textblöcke, Elemente, Elementgruppen und viele andere Abschnitte eines XML-Dokuments zu finden.

F&A

F Wie würden Sie einen Link in XML definieren, um das HTML-Verhalten nachzubilden?

A *Ein Link Element mit* `show="replace"` *und* `actuate="user"` *würde das HTML-Verhalten für einen normalen Link auf eine andere HTML-Seite nachbilden.*

F Was ist der Unterschied zwischen einem Inline-Link und einem Out-of-Line-Link?

A *Ein Inline-Link-Element enthält eine seiner Link-Ressourcen. Handelt es sich um einen einfachen Link, könnten Sie mehr oder weniger sagen, daß das Link-Element ein Ende des Links enthält.*

F Kann ein einfacher Link ein Out-of-Line-Link sein?

A *Theoretisch ja. In der Praxis wäre er jedoch nicht von größerem Nutzen, weil ein solcher Link mit nur einem Ende wenig mehr erledigen könnte, als einer Position eine bestimmte Semantik zuzuordnen.*

Übungen

1. Sie haben ein XML-Dokument mit einem `chapter`-Element, das in `section`-Elemente unterteilt ist. Sie wissen, daß Sie irgendwann ein Inhaltsverzeichnis anlegen wollen, deshalb wurde jedem `section`-Element ein `id`-Attribut zugewiesen, dessen Wert gleich der Abschnittsnummer ist. Skizzieren Sie den XML-Code (unter Verwendung eines erweiterten Out-of-Line-Links), und zeigen Sie, wie Sie ein eingebettetes Inhaltsverzeichnis anlegen würden.
2. Sie haben mehrere XML-Dokumente, die alle ein Kapitel eines Buches enthalten. Jetzt wollen Sie ein kurzes Inhaltsverzeichnis anlegen, das die `title`-Elemente jedes Kapitels aus diesen Dokumenten enthält. Skizzieren Sie den XML-Code (unter Verwendung einer erweiterten Out-of-Link-Link-Gruppe), die diese Aufgabe erledigt.

Die erweiterte Adressierung in XML

Woche 2

In Kapitel 10 haben Sie das Linking in XML kennengelernt, aber das ist nur ein Teil der gesamten Funktionalität. Heute lernen Sie die *Lokatoren* kennen – eine Methode, bestimmten Teilen von XML-Dateien Links zuzuordnen.

Sie kennen bereits eine Methode, Positionen zu bezeichnen, nämlich die URIs (*Universal Resource Identifier*). In XML können URIs ähnlich wie in HTML verwendet werden – zur Navigation im World Wide Web. In diesem Kapitel werde ich mich auf andere Lokatoren konzentrieren, die erweiterten Zeiger (XPointer). Hierbei geht es um folgendes:

- Wir sprechen ein bißchen über Wälder und wie XML Elementbäume sieht.
- Sie erfahren, wie man mit Hilfe absoluter Referenzen (`root()`, `origin()`, `id`, `html`) auf Ressourcen verweist.
- Sie lernen, wie Ressourcen unter Verwendung relativer Begriffe (`child`, `descendant`, `ancestor`, `preceding`, `psibling`, `following`, `fsibling`) angesprochen werden.
- Sie lernen, wie Link-Ressourcen, die aus mehreren Elementen und Elementbereichen (`spans`) bestehen, angesprochen werden.
- Sie erfahren, wie man Link-Ressourcen nach Instanz, Element, Attribut und Elementinhalt unterscheidet.

Erweiterte Zeiger

Wie Sie aus Kapitel 10 bereits wissen, werden beim Linking in XML Link-Ressourcen einander zugeordnet. (Eine Link-Ressource ist eigentlich ein Zeiger auf eine oder mehrere Positionen oder eine Methode, einen Zeiger in eine Position aufzulösen.) Sie können fast alles benutzen, um eine Position aufzulösen, beispielsweise ein Web-Server-CGI-Skript oder eine Datenbankabfrage. Die erweiterten Zeiger von XML (XPointers) ermöglichen Ihnen, auf die unterschiedlichsten Arten auf XML-Ressourcen zu zeigen. Wie Sie sehen werden, können Sie die Dokumentstruktur, die durch XML-Elemente und -Attribute festgelegt ist, dabei voll nutzen.

Erweiterte Zeiger sind in einem separaten Vorschlag beschrieben, *XML Pointer Language* XPointer), der Teil des allgemeinen XLL-Vorschlags (*XML Link Language*) ist.

Sie verwenden einen erweiterten Zeiger in Kombination mit einer URI, um noch genauer auf eine Link-Ressource zu verweisen. Wenn Sie in ein XML-Dokument zeigen, müssen Sie einen erweiterten Zeiger verwenden. Wenn Sie in etwas anderes als ein XML-Dokument zeigen, schreibt die XPointer-Empfehlung nichts über die Syntax des Lokators vor und schränkt sie auch nicht ein, aber Sie können dennoch erweiterte Zeiger benutzen.

Dokumente als Bäume

Bevor Sie verstehen können, wie erweiterte Zeiger arbeiten und wie man sie korrekt einsetzt, sollten Sie verstehen, wie ein XML-Link-Prozessor (nämlich ein Softwarepaket, das einen Link auflöst, um die eigentlichen Enden zu finden) die Struktur eines XML-Dokuments »sieht«.

Bei Ihrer bisherigen Arbeit mit diesem Buch haben Sie sich zweifellos an den Gedanken gewöhnt, daß sich ein XML-Dokument aus Elementbäumen zusammensetzt. Das »Wurzel«-Element befindet sich ganz oben oder ganz unten (abhängig von der Perspektive), und alle anderen Elemente verzweigen sich von dort aus.

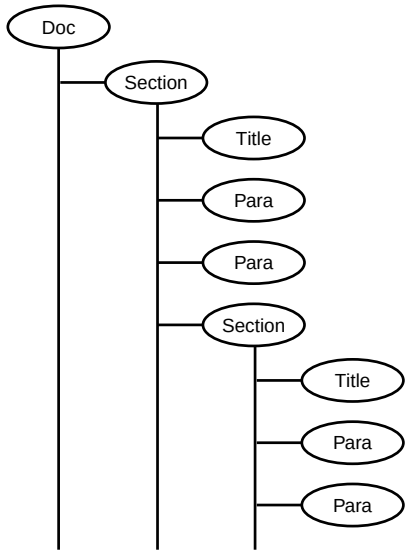


Abbildung 11.1:
Ein typischer XML-Elementbaum

Eine gewöhnliche Baumstruktur wäre jedoch zu einfach für die Beschreibung erweiterter Zeiger. Wenn ich eine absolute Position mit einem relativen Link kombiniere, der irgendwo tief im Dokument beginnt, muß der Auflösungsmechanismus den Baum, der mit dem Wurzelelement beginnt, mit dem Baum, der mit dem als Startpunkt für die relative Position dienenden Element beginnt, kombinieren. Aus der Sicht eines erweiterten Zeigers handelt es sich bei einem XML-Dokument also nicht um einen einzigen Elementbaum – sondern um Hunderte.

Für jeden möglichen Eintrittspunkt oder Referenzpunkt kann ein Baum angelegt werden, statt also über einen Baum zu sprechen, verwenden wir den Begriff Hain. Ein Hain besteht aus mehreren Bäumen. Er ist zu klein, um ein Wald zu sein, aber auch mehr als nur ein einziger riesiger Baum.

Jeder Punkt in dem Baum wird als Knoten bezeichnet. Wir haben bereits über Elementbäume gesprochen, deshalb nehmen Sie sicher an, die Knoten sind einfach Elemente. Das stimmt so nicht. Ein Knoten kann ein Element sein, aber auch einfach ein Textabschnitt (Zeichendaten) innerhalb eines Elements.

Wenn Sie mehr über DSSL erfahren haben, werden Sie sehen, daß ein solcher Hain mit Hilfe von Elementen und Attributen konstruiert werden kann. Das ist für das Linking nicht wichtig, weil Attribute nicht als Link-Ressourcen genutzt werden können.

Betrachten Sie die wie folgt deklarierte Elementstruktur:

```
<!ELEMENT section (#PCDATA ¦ title ¦ para )>
```

Dabei handelt es sich um ein typisches Element mit gemischtem Inhalt. Es kann sowohl Rohzeichendaten (ohne Tags) als auch Elemente (`title` und `para`) enthalten. Betrachten Sie den folgenden Codeabschnitt:

```
<section><title>Bert geht zu Bett</title>  <para>Hallo, Bert.</para>
<para>Heute ist ein besonderer Tag.</para></section>
```

Jetzt wollen wir versuchen, diesen Abschnitt als Baum darzustellen. Es scheint ganz einfach zu sein. Wir haben ein `section`-Element, das ein `title`-Element und zwei `para`-Elemente umfaßt. Wirklich? Betrachten Sie den Abschnitt noch einmal, und sehen Sie sich dann an, was ich in Abbildung 11.2 daraus gemacht habe.

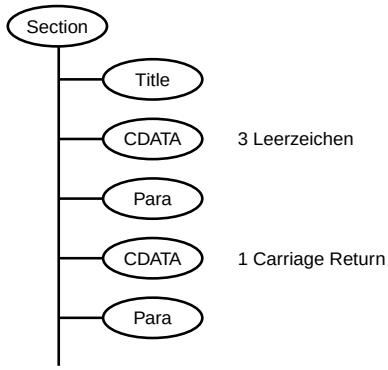


Abbildung 11.2:
Ein Elementbaum mit verborgenen Knoten.

In Abbildung 11.2 habe ich ein `section`-Element mit einem `title`-Element, einen Knoten mit Zeichendaten (drei Leerzeichen), ein `para`-Element, einen dazwischenliegenden Zeichendaten-Knoten (ein CR-Zeichen oder sogar ein CR- und ein LF-Zeichen, abhängig vom verwendeten Betriebssystem) und ein letztes `para`-Element erkannt.

Dies ist ein sorgfältig gewähltes Beispiel, das zeigt, was schieflaufen kann, aber diese verborgenen Knoten (weniger verborgen als vergessen) können zu sehr unerwarteten Ergebnissen führen, wenn Sie versuchen, erweiterte Zeiger zu verwenden, die Knoten zählen, um die Link-Ressource zu finden.

Positionsbezeichnungen

Es gibt zwei Arten von Positionsbezeichnungen: absolute Bezeichnungen und relative Bezeichnungen.

Absolute Bezeichnungen sind ähnlich den Positionszeigern, die Sie aus HTML kennen. Diese Begriffe ignorieren die strukturellen Beziehungen mehr oder weniger und zeigen direkt auf bestimmte Elemente.

Relative Bezeichnungen sind einfacher nachzuvollziehen und leistungsfähiger als absolute Bezeichnungen, weil sie beispielsweise die Wegbeschreibung nachbilden, die Sie einem Freund mitteilen, damit dieser Ihr Büro findet.

Absolute Bezeichnungen

Ein erweiterter Zeiger kann eine der folgenden absoluten Positionsbezeichnungen verwenden:

- `root()` – Gibt an, daß die Positionsquelle für die Positionsbezeichnung das Wurzelelement des aktuellen Dokuments ist. Beachten Sie, daß dem Schlüsselwort root ein Klammernpaar folgt (technisch ausgedrückt, ein *leeres Argument*), um zu verhindern, daß es mit IDs verwechselt wird.
- `origin()` – Gibt an, daß die Positionsquelle für die Positionsbezeichnung die Link-Ressource ist, an der das Durchqueren begonnen hat, und nicht das aktuelle Wurzelelement. Beachten Sie, daß dem Schlüsselwort origin ein Klammernpaar folgt (technisch ausgedrückt, ein *leeres Argument*), um zu verhindern, daß es mit IDs verwechselt wird.
- `id(name) -` Gibt an, daß die Positionsquelle für die Positionsbezeichnung das Element (es muß ein eindeutiges Element sein) ist, für das ein Attribut als ID-Klasse deklariert wurde. Der Wert dieses Attributs ist `name`. Der Name dieses Attributs könnte völlig beliebig sein, deshalb die Bedeutung, Attribute explizit zu deklarieren, die Sie als IDs verwenden wollen.
- `html(name-wert)` – Gibt an, daß die Positionsquelle für die erste Positionsbezeichnung das erste Element des Typs A ist, das ein name-Attribut mit dem Wert *name-wert* hat (das ist dasselbe wie die Auswahl eines Ziels in HTML unter Verwendung des Symbols #).

Wenn Sie die Positionsquelle der ersten Positionsbezeichnung bei einer Suche nicht angeben, wird standardmäßig das Wurzelelement des XML-Dokuments, das die Ressource enthält, verwendet. Das bedeutet, wenigstens offiziell, daß Sie `root()` eigentlich nie anzugeben brauchen. Es schadet aber auch nicht, wenn Sie `root()` angeben, wodurch einem menschlichen Betrachter die Aufgabe Ihres Zeigers sehr viel deutlicher wird.

Relative Bezeichnungen

Für eine relative Positionsbezeichnung muß es einen Ausgangspunkt geben (eine Positionsquelle). Sie verwenden relative Positionsbezeichnungen, um zum Ziel-XML-Dokument zu gelangen, indem Sie die Position in Hinblick auf Ihren aktuellen Standpunkt beschreiben.

Das Ziel eines erweiterten Zeigers ist letztlich, Sie an die korrekte Position innerhalb eines XML-Dokuments zu führen. Das ist ein bißchen so, als würde man versuchen, jemandem eine Wegbeschreibung zu einem bestimmten Platz in einer Stadt zu geben. Es ist schwierig, beim ersten Versuch gleich die richtige Richtung vorzugeben. Möglicherweise fangen Sie mit einem bekannten Ausgangspunkt (einer absoluten Position) an, zählen Blöcke oder Straßen und beschreiben dann die Gebäude:

```
Geh zum Hauptbahnhof
Geh nach Norden weiter zur Beethovenbrücke
Nimm die dritte Straße links
Geh in das zweite Haus links
Frag nach Alfons
```

Jeder Schritt der Anweisung geht davon aus, daß der vorhergehende Schritt korrekt ausgeführt wurde. Wird dabei ein Fehler gemacht, führen alle anderen Anweisungen Sie unweigerlich in die Irre.

Erweiterte Zeiger arbeiten ganz ähnlich. Sie bilden eine Abfolge von Positionsbezeichnungen, die entweder absolute Positionen darstellen oder eine Folge von Positionen, wo sich jede auf die jeweils vorhergehende bezieht. Erweiterte Zeiger beschreiben Elemente und Knoten in einem XML-Dokument hinsichtlich verschiedener Eigenschaften, beispielsweise ihrer Typ- oder Attributwerte, oder einfach durch Zählen.

Jede relative Positionsbezeichnung besteht aus einem *Schlüsselwort* gefolgt von einem oder mehreren *Schritten*. Jede Bezeichnung ist ein Sprung auf dem Pfad zur endgültigen Position der Link-Ressource. Auf dieselbe Weise, wie Sie jemandem sagen: »Nimm die dritte Straße rechts, geh drei Blöcke weiter, biege nach links ab und geh in das zweite Haus rechts«, könnten Sie jemandem mit Hilfe erweiterter Zeiger beispielsweise sagen: »Nimm das dritte Kind dieses Elements, suche seinen ältesten Bruder und nimm dann dessen zweites Kindelement.« Eine solche Wegbeschreibung könnte unter Verwendung erweiterter Zeiger etwa wie folgt geschrieben werden:

```
child(3,DIV).psibling(-2).descendant(4.para).string(5,"bottle",1,1)
```

Sie können die folgenden Schlüsselwörter für relative Bezeichnungen verwenden:

- `child` – Wählt die Kindelemente der Positionsquelle aus.
- `descendant` – Wählt die Elemente im Inhalt der Positionsquelle aus.
- `ancestor` – Wählt die Elemente aus, in deren Inhalt die Positionsquelle gefunden wird.
- `preceding` – Wählt Elemente aus, die vor der Positionsquelle erscheinen.
- `following` – Wählt Elemente aus, die nach der Positionsquelle erscheinen.
- `psibling` – Wählt die vorhergehenden Geschwisterelemente der Positionsquelle aus.
- `fsibling` – Wählt die nachfolgenden Geschwisterelemente der Positionsquelle aus.

Abbildung 11.3 zeigt ein Beispiel für solch einen Hain. Ich will versuchen, Ihnen diese Schlüsselwörter in Hinblick auf einen bestimmten Punkt in diesem Hain zu erklären – Element Nummer 4.

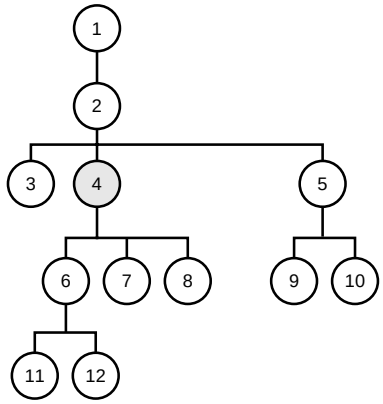


Abbildung 11.3:
Relative Positionen in einem Elementbaum.

Wir verwenden Element Nummer 4 als Positionsquelle und betrachten, was die absoluten Bezeichnerschlüsselwörter dafür bedeuten.

- `child -` Wählt die Elemente 6, 7 und 8 aus (die unmittelbaren Kinder, nicht aber 11 und 12).
- `descendant` – Wählt die Elemente 6, 7, 8, 11 und 12 aus (die unmittelbaren Kinder und deren Kinder).
- `ancestor` – Wählt die Elemente 1 und 2 aus (die Elemente, die ElemeNT 4 enthalten).

- `preceding` – Wählt die Elemente 1, 2 und 3 aus (alle Elemente, die vor der Positionsquelle erscheinen).
- `following` – Wählt die Elemente 6, 7, 8, 11, 12, 5, 9 und 10 aus (alle Elemente, die nach der Positionsquelle erscheinen).
- `psibling` – Wählt das Element 3 aus (die Elemente auf derselben Hierarchiestufe oder in derselben Generation, die vor der Positionsquelle stehen).
- `fsibling` – Wählt Element 5 aus (die Elemente auf derselben Hierarchiestufe oder in derselben Generation, die nach der Positionsquelle stehen).

Auswahl

Bisher haben Sie gelernt, einen Positionstyp mit Hilfe absoluter oder relativer Bezeichnungen anzugeben. Diese Bezeichnungen teilen dem Prozessor mit, welches Element Sie ansprechen wollen, aber Sie sollten außerdem in der Lage sein, anzugeben, an welchen dieser Elemente (oder vielleicht allen davon) Sie interessiert sind. Erweiterte Zeiger bieten mehrere Methoden, Elemente auszuwählen, wie Sie noch sehen werden.

Auswahl nach der Instanznummer

Durch die Angabe einer Nummer können Sie eine bestimmte Elementinstanz oder einen Bereich von Elementinstanzen ansprechen, beispielsweise wie folgt:

```
child (2)
```

Damit wird das zweite Kindelement angesprochen.

Bei der Angabe einer negativen Zahl wird von der letzten Instanz aus rückwärts gezählt, beispielsweise:

```
(-1, PARA)
```

Damit wird das letzte <PARA>-Element angesprochen.

Auswahl nach dem Knotentyp

Neben der Auswahl einer Ressource nach einer Nummer können Sie auch einen bestimmten Knotentyp angeben. Um einen bestimmten Knotentyp auszuwählen, folgen der Instanz ein Komma und ein `node type`, wobei es sich um eines der folgenden Dinge handeln kann:

- Ein `name`-Wert wählt Elemente mit einem bestimmten Namen aus.
- Beispielsweise spricht `child(2,para)` das zweite `para`-Element an, das Kind des aktuellen Elements ist.
- Das Schlüsselwort `#element` wählt XML-Elemente aus. Das ist der Standardtyp, falls Sie keinen Typ angeben, deshalb werden Sie dieses Schlüsselwort auch vermutlich nie brauchen.
- Das Schlüsselwort `#pi` wählt XML-Verarbeitungsanweisungen (Processing Instructions) aus. Offensichtlich können Sie nicht zu Attributen einer Verarbeitungsanweisung gehen oder zu darin enthaltenen Elementen (beides ist unmöglich), aber Sie können (mit `string`) Text darin ansprechen.
- Das Schlüsselwort `#comment` wählt XML-Kommentare aus. Offensichtlich können Sie nicht zu Attributen eines Kommentars gehen oder zu darin enthaltenen Elementen (beides ist unmöglich), aber Sie können (mit `string`) Text darin ansprechen.
- Das Schlüsselwort `#text` wählt Textbereich direkt innerhalb der Zeichendatenabschnitte (CDATA) innerhalb eines Elements aus. Sie können dafür keine Attribute angeben, aber Sie können bestimmte Textabschnitte innerhalb eines Textbereichs ansprechen (mit `string`).
- Das Schlüsselwort `#cdata` wählt Textbereiche innerhalb von Zeichendatenabschnitten (CDATA) aus. Es ist nicht möglich, Attribute anzugeben, aber Sie können bestimmte Textabschnitte innerhalb eines CDATA-Abschnitts ansprechen (mit `string`).
- Das Schlüsselwort `#all` wählt alle Knotentypen aus.

Auswahl nach einem Attribut

Der Elementtyp kann, falls er angegeben wurde, mit einem Attributnamen genauer spezifiziert werden. Dazu verwenden Sie das Schlüsselwort `attr`. Sie erhalten den Wert des angegebenen Attributs.

Auswahl von Text

Alle bisher vorgestellten absoluten und relativen Positionsbeschreibungen wählen vollständige Elemente aus. Die Positionsbezeichnung `string` erlaubt Ihnen, in ein Element zu blicken und den darin befindlichen Text auszuwählen. Diese String-Auswahl ist besonders praktisch, wenn man Links zu Nicht-XML-Daten (wie etwa Textdateien) vornimmt oder zu XML-Daten, die sehr große Textblöcke enthalten.

Die `string`-Positionsbezeichnung nimmt die folgenden vier Argumente entgegen:

- Eine Zahl, die festlegt, welches Vorkommen des angegebenen Strings gesucht werden soll. Um alle Vorkommen auszuwählen, verwenden Sie das Schlüsselwort `all`.
- Den Zeichenstring, in einfache oder doppelte Anführungszeichen eingeschlossen (beispielsweise »das war's nicht«), mit dem eine Übereinstimmung gesucht wird. Statt eines Strings können Sie auch eine Zahl angeben, um ein bestimmtes Zeichen darzustellen.
- Eine Positionsnummer, die angibt, wie viele Zeichen vom Anfang des übereinstimmenden Strings gezählt werden sollen, um den gesuchten String zu finden. Dabei darf als Wert nicht 0 angegeben werden. Wenn Sie keinen Wert angeben, wird 1 angenommen. Eine positive Zahl (beispielsweise +2) zählt vom linken Ende des Strings nach rechts, eine negative Zahl (beispielsweise -4) zählt vom rechten Ende des Strings an rückwärts (nach links).
- Eine Nummer, die die Länge des gesuchten Strings angibt (die Anzahl der Zeichen im String). Wenn Sie 0 oder überhaupt keinen Wert angeben, wird damit die Position unmittelbar vor dem Zeichen an der Positionsnummer angenommen. Mit einem leeren String (einem Paar Anführungszeichen: `""`) geben Sie den Punkt unmittelbar nach dem gesuchten Zeichenstring an.

Hier einige Beispiele für die Verwendung von `string`:

- `string(3,"das",1,2)` wählt die Zeichen »as« vom dritten Vorkommen des Worts »das« an aus.
- Vorausgesetzt, Sie haben das Wort bereits ausgewählt, gibt `string(2,"")` die Positon zwischen dem »d« und dem »a« im Wort »das« an.
- `string(all,"(",-2,1)` wählt das letzte Zeichen jedes Worts aus, das vor einer öffnenden Klammer erscheint.
- `string(1,"badesee",-6,3)` wählt die Zeichen »ade« vom ersten Vorkommen des Worts »badesee« aus.

Auswahl von Gruppen und Bereichen (Spannen)

Eine Ressourcenposition kann einen einzelnen erweiterten Zeiger oder zwei durch das Schlüsselwort `span` voneinander getrennte erweiterte Zeiger enthalten. Enthält sie zwei erweiterte Zeiger, umfaßt die Position alles zwischen dem Anfang des Ziels des ersten erweiterten Zeigers bis zum Ende des zweiten.

Spannen sind eine große Ausnahme zu der restlichen Sprache der erweiterten Zeiger, weil es sich bei den Spannenpositionen *nicht* um Bäume handelt. Sie können nicht voraussetzen, daß eine Spanne einen sinnvollen Abschnitt eines XML-Dokuments enthält. Es ist sehr unwahrscheinlich, daß es sich dabei um ein einzelnes oder auch mehrere Elemente handelt. Damit sind die Verwendungsmöglichkeiten der Spannen in XML-Anwendungen beschränkt, aber es bleibt der XML-Software überlassen, damit zurechtzukommen.

Zusammenfassung

In diesem Kapitel haben wir ein XML-Dokument als Objekt betrachtet, das sich aus mehreren Komponenten zusammensetzt. Sie haben Markup und die Unterscheidung zwischen Markup und Zeichendaten in XML kennengelernt. Außerdem haben Sie erfahren, wie XML-Dokumente gleichzeitig eine logische und eine physische Struktur haben, und die Verbindung zwischen diesen und den Elementen und Entities von XML kennengelernt.

F&A

F Was ist ein Hain?

A *Ein XML-Dokument mit einer hierarchischen, baumähnlichen Struktur. Die Ansicht eines XML-Dokuments, das für das Linking verwendet wird, ist eine Menge von Bäumen, ein sogenannter Hain. Ein Blatt wird als Knoten bezeichnet, und die Zweige heißen Kinder.*

F Warum müssen Sie im Umgang mit `string` und `span` so vorsichtig sein?

A *Anders als die anderen erweiterten Zeiger, die ganze Elemente oder Elementgruppen ansprechen, können `string` und `span` Teile von Elementen auswählen. Das könnte dazu führen, daß ein XML-Dokument nicht mehr gültig oder nicht mehr wohlgeformt ist.*

F Hat der erweiterte Zeiger `root()` irgendeine sinnvolle Aufgabe?

A *Nicht wirklich, aber er macht das Linking für menschliche Leser besser nachvollziehbar. (Sie wissen, daß es eines der Entwurfsziele von XML und XLL ist, daß sie vom Menschen möglichst gut lesbar sind.)*

F Warum werden `root()` und `origin()` immer mit nachfolgendem Klammerpaar geschrieben?

A *Die Klammern stellen sogenannte leere Elemente dar, die verhindern, daß eine Verwechslung mit IDs auftritt.*

Übungen

1. Die folgenden Daten stammen aus einem einfachen XML-Buch-Katalog:

```
<catalog>
<title>Simon's XML Book Catalog</title>
<book>
<title>Presenting XML</title>
<author>Richard Light</author>
<author>Simon North</author>
<publisher>Sams.Net</publisher>
<date>1997</date>
<ISBN></ISBN>
<cd>no</cd>
</book>
<book>
<title>XML Complete</title>
<author>Steven Holzner</author>
<publisher>McGraw-Hill</publisher>
<date>1998</date>
<ISBN>0-07-913702-4</ISBN>
<cd>yes</cd>
</book>
<book>
<title>Designing XML Internet Applications</title>
<author>Michael Leventhal</author>
<author>David Lewis</author>
<author>Mathew Fuchs</author>
<publisher>Prentice Hall PTR</publisher>
<date>1998</date>
<ISBN>0-13-616882-1</ISBN>
<cd>yes</cd>
</book>
<book>
<title>Structuring XML Documents</title>
<author>David Megginson</author>
<publisher>Prentice Hall PTR</publisher>
<date>1998</date>
<ISBN>0-13-642299-3</ISBN>
```

```
<cd>yes</cd>
</book>
<book>
<title>The XML Companion</title>
<author>Neil Bradley</author>
<publisher>Addison-Wesley</publisher>
<date>1998</date>
<ISBN>0-201-34285-5</ISBN>
<cd>no</cd>
</book>
<book>
<title>XML: Extensible Markup Language</title>
<author>Elliotte Rusty Harold</author>
<publisher>IDG Books</publisher>
<date>1998</date>
<ISBN>0-7645-3199-9</ISBN>
<cd>yes</cd>
</book>
</catalog>
```

Welchen erweiterten Zeiger könnten Sie zur Auswahl eines Buchtitels einsetzen?

2. Welchen erweiterten Zeiger könnten Sie zur Auswahl des dritten Buchtitels einsetzen?
3. Welchen erweiterten Zeiger könnten Sie zur Auswahl der vollständigen Einträge für das zweite und das dritte Buch einsetzen?

Die Anzeige von XML im Internet Explorer

Woche 2

Heute werden wir über die folgenden Dinge sprechen:

- Microsofts XML-Vision
- Anzeige von XML-Daten im Microsoft Internet Explorer 4
- Anzeige von XML-Daten im Microsoft Internet Explorer 5

Microsofts XML-Vision

Wir beginnen mit dem Wesentlichen. XML kann zur Kennzeichnung drei verschiedener Informationsarten verwendet werden:

- Daten – Strukturierte Informationsmengen, die normalerweise von relationalen Datenbanken verarbeitet werden.
- Dokumente – Weniger strukturierte Informationsmengen mit einer bestimmten Eigenschaft, nämlich daß die Reihenfolge der darin enthaltenen Information von Bedeutung ist.
- Meta-Daten – Daten über andere Informationen, beispielsweise wer der Autor ist, Schlüsselwörter, die Zielgruppe usw.

Bis vor einiger Zeit dachte man bei Microsoft, wenn man Leuten Dokumente zeigen will, sei HTML die Markup-Sprache der Wahl. Sie betrachteten XML als die beste Methode, strukturierte Daten zwischen Applikationen auszutauschen.

Hier ein Beispiel dafür, was man sich vorstellte: Eine relationale Datenbank wird abgefragt. Als Ergebnis dieser Abfrage sollten mit XML-Tags ausgezeichnete, strukturierte Daten über das Netzwerk an den Client geschickt werden, wo die Daten abgefragt, verändert, angezeigt, verarbeitet usw. werden können.

Internet Explorer 4 bietet bereits Funktionen zum Lesen und Anzeigen von Daten, die mit XML-Tags ausgezeichnet sind.

Microsoft verwendet XML auch für Meta-Daten. Ihr .CDF-Format (*Channel Definition Format*) ist eine XML-Anwendung, die Sites beschreibt, also Meta-Information (Information über Sites) bereitstellt.

Die Anzeige von XML im Internet Explorer 4

Es gibt unterschiedliche Methoden, XML im IE4 zu nutzen. Wir werden sie hier genauer betrachten.

Überblick über die XML-Unterstützung im Internet Explorer 4

Der Internet Explorer 4

- führt eine CDF-Verarbeitung durch.
- CDF steht für Channel Definition Format, eine XML-Anwendung.
- beinhaltet zwei XML-Parser: einen nichtanalysierenden Parser in C und einen analysierenden Parser in Java.

Diese Parser wurden in den Kapiteln 5 und 9 nicht erwähnt, weil sie zum Zeitpunkt der Drucklegung dieses Buchs noch nicht vollständig mit der W3C-Empfehlung konform waren.

- stellt ein XML-Data-Source-Objekt bereit.
- stellt ein XML Object Model bereit, ebenso wie eine Schnittstelle (API), über die die Entwickler mit den XML-Daten im Browser arbeiten können.

Anzeige von XML unter Verwendung des XML Data Source Object

Data-Source-Objekte werden für das verwendet, was Microsoft als *Datenbindung* bezeichnet. Die Datenbindung ist die Methode von Microsoft, Datenmanipulationen dem Browser (Client) zu überlassen und damit vom Server zu entfernen. Wenn Sie eine neue Sicht auf die Daten brauchen, setzen Sie normalerweise eine erneute Abfrage beim Server ab. Der Server führt die erforderlichen Berechnungen aus und sendet eine neue HTML-Seite an den Client. Das passiert bei der Datenbindung nicht. Der Server sendet eine HTML-Seite zusammen mit den Daten zum Client. Die Daten werden lokal abgelegt und können lokal verarbeitet werden, ohne daß eine weitere Verbindung zum Server erforderlich ist.

Die Auslagerung der Verarbeitung vom Server auf den Client ist keine schlechte Idee – in einer Welt, wo es ständig zu wenig Bandbreite gibt.

Um diese Datenbindung zu implementieren, müssen Sie zuerst ein Data-Source-Objekt in Ihre Seite einbinden. Das XML-Data-Source-Objekt ist nur eines der Data-Source-Objekte, die der Internet Explorer 4.0 bietet.

Einige andere Data Source-Objekte:

- Das TDC (Tabular Data Control)
- Der Remote Data Service (mit OLE-DB oder ODBC)
- Das JDBC-Applet

Weitere Informationen über Data-Source-Objekte finden Sie auf der folgenden Web-Site: `http://www.microsoft.com/workshop/c-frame.htm#/workshop/author/default.asp`.

Nachdem Sie das Data-Source-Objekt eingefügt haben, definieren Sie HTML-Elemente, die in der Lage sind, die Daten aus dieser Datenquelle zu lesen: die sogenannten »Datenverbraucher« oder »Datenkonsumenten«.

Wir betrachten ein konkretes Beispiel. Als erstes brauchen wir eine XML-Datei mit strukturierten Daten, wie in Listing 12.1 gezeigt.

Listing 12.1:** `musicans.xml` **– eine XML-Datei mit strukturierten Daten

```
1:  <?xml version="1.0" ?>
2:  <musicians>
3:  <musician>
4:  <name>Joey Baron
5:  </name>
6:  <instrument>drums
7:  </instrument>
8:  <NrOfRecordings>1
9:  </NrOfRecordings>
10: </musician>
11: <musician>
12: <name>Bill Frisell
13: </name>
14: <instrument>guitar
15: </instrument>
16: <NrOfRecordings>3
17: </NrOfRecordings>
18: </musician>
19: <musician>
20: <name>Don Byron
21: </name>
22: <instrument>clarinet
23: </instrument>
24: <NrOfRecordings>2
25: </NrOfRecordings>
26: </musician>
```

```
27:  <musician>
28:  <name>Dave Douglas
29:  </name>
30:  <instrument>trumpet
31:  </instrument>
32:  <NrOfRecordings>1
33:  </NrOfRecordings>
34:  </musician>
35:  </musicians>
```

Außerdem brauchen wir eine HTML-Datei, wie in Listing 12.2 gezeigt.

Listing 12.2: HTML-Datei, der ein XML-Data-Source-Objekt hinzugefügt wird

```
1:   <HTML>
2:   <HEAD>
3:   <TITLE>Overview of musicians</TITLE>
4:   </HEAD>
5:   <BODY>
6:   <H1>An overview of my favorite musicians</H1>
7:   <HR>
8:   <P>My favorite musicians are:</P>
9:   <!-- Things to come -- >
10:  <HR>
11:  <!-- some other stuff -- >
12:  </BODY>
13:  </HTML>
```

Anschließend wird das XML-Data-Source-Objekt eingefügt, wie in Listing 12.3 gezeigt.

Listing 12.3: Ein XML-Data-Source-Objekt

```
1:  <APPLET
2:      code="com.ms.xml.dso.XMLDSO.class"
3:      id="xmldso"
4:      width="0"
5:      height="0"
6:      mayscript="true">
7:  <PARAM NAME="URL" VALUE="...">
8:  </APPLET>
```

Die drei Punkte (...) stehen für die URL, die auf eine XML-Datei verweist.

Normalerweise kommen die XML-Daten von derselben Stelle wie die HTML-Datei, die das `APPLET`-Tag enthält.

Listing 12.4 plaziert das Applet in unserer HTML-Datei.

Listing 12.4: HTML-Datei mit eingefügtem XML-Data-Source-Objekt

```
<HTML>
<HEAD>
<TITLE>Overview of musicians</TITLE>
</HEAD>
<BODY>
<H1>An overview of my favorite musicians</H1>
<HR>
<P> My favorite musicians are: </P>
<APPLET
    code="com.ms.xml.dso.XMLDSO.class"
    id="xmldso"
    width="0"
    height="0"
    mayscript="true">
<PARAM NAME="URL" VALUE="musicians.xml">
</APPLET>
<!-- More things to come -- >
<HR>
<!-- some other stuff -- >
</BODY>
</HTML>
```

Im nächsten Schritt werden wir HTML-Elemente einbinden, die in der Lage sind, die von unserem XML-Data-Source-Objekt bereitgestellten Daten darzustellen.

Nicht alle HTML-Elemente unterstützen die Datenbindung. Die folgende Liste zeigt die wichtigsten Elemente für die Datenbindung:

- DIV
- SPAN
- TABLE

Um Informationen aus der externen Datenquelle an diese HTML-Elemente zu binden, hat Microsoft ihnen einige Attribute zugeordnet. Die wichtigsten davon sind in Tabelle 12.1 aufgelistet.

Attribut	Aufgabe
DATASRC	Verweist auf den Namen des Data-Source-Objekts.
DATAFLD	Gibt die Daten an, an die das Element gebunden wird.

Tabelle 12.1: Attribute für die Verbindung des Data-Source-Objekts mit dem Datenverbraucher

Jetzt vervollständigen wir unsere Beispiel:

```
<P><SPAN DATASRC="#xmldso" DATAFLD="name"></SPAN></P>
```

Dabei verweist der Wert des DATASRC-Attributs auf die ID unseres Applets, also xmldso.

Das Zeichen # wird verwendet, weil es sich um einen Verweis auf einen Namen innerhalb desselben Dokuments handelt.

Der Wert des DATAFLD-Attributs bezieht sich auf den Namen eines in unserer XML-Datei verwendeten Elements.

Listing 12.5 zeigt, wie diese Datenverbraucher mit dem Data-Source-Objekt verbunden werden.

Listing 12.5: Die Verbindung des Data-Source-Objekts mit dem Datenverbraucher

```
1:   <HTML>
2:   <HEAD>
3:   <TITLE>Overview of musicians</TITLE>
4:   </HEAD>
5:   <BODY>
6:   <H1>An overview of my favorite musicians</H1>
7:   <HR>
8:   <P>My favorite musicians are:</P>
9:   <APPLET
10:      code="com.ms.xml.dso.XMLDSO.class"
11:      id="xmldso"
12:      width="0"
```

```
13:      height="0"
14:      mayscript="true">
15:  <PARAM NAME="URL" VALUE="musicians.xml">
16:  </APPLET>
17:  <DL>
18:  <DT><B><SPAN DATASRC="#xmldso" DATAFLD="name"></SPAN></B></DT>
19:  <DD>on <SPAN DATASRC="#xmldso" DATAFLD="instrument"></SPAN></DD>
20:  </DL>
21:  <HR>
22:  <!-- some other stuff -- >
23:  </BODY>
24:  </HTML>
```

Abbildung 12.1 zeigt diese HTML-Datei im Internet Explorer 4.0.

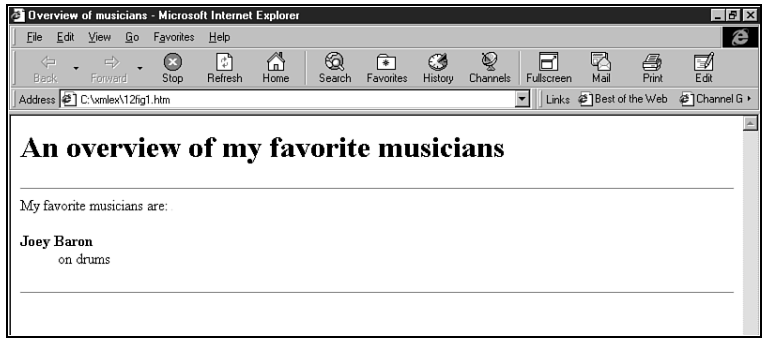


Abbildung 12.1: Das Ergebnis einer Datenbindung im Internet Explorer 4.0.

Damit haben wir die grundsätzliche Vorgehensweise. Wir können Informationen aus unserer XML-Datei lesen – aber das Ergebnis ist noch nicht berauschend.

Wir lesen einfach Informationen über den ersten Musiker, der in unserer XML-Datei erscheint.

Beim Laden der Daten aus der XML-Datei wird die Information in unterschiedliche Datensätze aufgesplittet, wovon nur einer zur Verfügung steht. Er wird als der aktuelle Datensatz bezeichnet.

Nach dem Laden der XML-Datei ist normalerweise der erste Datensatz der aktuelle.

Natürlich wollen wir auch Informationen über alle anderen Musiker haben. Es gibt zwei Techniken, um diese Informationen zu erhalten:

- Wir könnten eine Funktion einfügen, die die Daten der einzelnen Musiker sequentiell ermittelt – mit anderen Worten, mit der wir uns durch die Daten bewegen können.
- Wir könnten ein Element verwenden, das in der Lage ist, mehrere Werte aufzunehmen – das HTML-Element `TABLE`.

Zuerst wollen wir betrachten, wie wir uns durch die Daten bewegen.

Dazu verwenden wir den Code in Listing 12.6.

Listing 12.6: Schaltflächen für die Navigation in die Daten einfügen

```
<input type=button value="<<"
    onclick='xmldso.recordset.movefirst();'>
<input type=button value="<"
    onclick='xmldso.recordset.moveprevious();>
<input type=button value=">"
    onclick='xmldso.recordset.movenext();'>
<input type=button value=">>"
    onclick='xmldso.recordset.movelast();'>
```

Es gibt damit vier neue Schaltflächen (`INPUT type="button"`) mit Semantik:

- Gehe zum ersten (<<)
- Gehe zum letzten (>>)
- Gehe zum vorhergehenden (<)
- Gehe zum nächsten (>)

Die Schaltflächen entsprechen den vier Methoden zur Navigation in dem Recordset, das unser XML-Data-Source-Objekt erzeugt hat.

Wenn Sie dies in Ihrem Browser öffnen, sehen Sie jedoch, daß es nicht so funktioniert wie vorgesehen.

Für den `next`-Fall muß eine Prüfung auf das Dateiende (EOF) eingefügt werden. Dasselbe gilt für `previous` und den Dateianfang (BOF), wie in Listing 12.7 gezeigt.

Listing 12.7: Positionsprüfung innerhalb der Daten

```
 <input type=button value="<<"
     onclick='xmldso.recordset.movefirst();'>
 <input type=button value="<"
```

```
4:       onclick='xmldso.recordset.moveprevious();
5:           if (xmldso.recordset.BOF)
6:           xmldso.recordset.movefirst();'>
7:   <input type=button value=">"
8:       onclick='xmldso.recordset.movenext();
9:           if (xmldso.recordset.EOF)
10:          xmldso.recordset.movelast();'>
11:  <input type=button value=">>"
12:      onclick='xmldso.recordset.movelast();'>
```

Abbildung 12.2 zeigt unsere HTML-Datei mit den Navigationsschaltflächen im Internet Explorer 4.0.

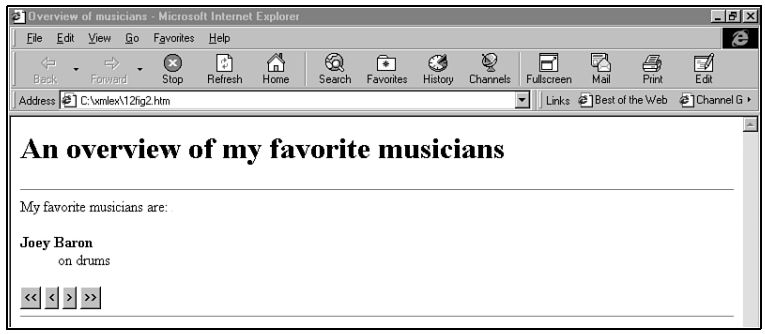


Abbildung 12.2: Die neuen Navigationsschaltflächen.

Die zweite Lösung wäre ein Überblick im Tabellenformat. Wir werden also ein `TABLE`-Element an die Daten binden, wie in Listing 12.8 gezeigt.

Listing 12.8: Verwendung einer Tabelle

```
1:  <TABLE DATASRC="#xmldso">
2:  <THEAD>
3:  <TH>Musician</TH>
4:  <TH>Instrument</TH>
5:  </THEAD>
6:  <TR>
7:      <TD><SPAN DATAFLD="name"></SPAN></TD>
8:      <TD><SPAN DATAFLD="instrument"></SPAN></TD>
9:  </TR>
10: </TABLE>
```

Die Tabelle wird für die Datenquelle auf `bound` gesetzt. Innerhalb der Tabelle gibt es einwertige Elemente (`SPAN`), die auf die verschiedenen XML-Elemente verweisen, wie in Listing 12.9 gezeigt.

Listing 12.9: Die vollständige HTML-Datei

```
1:   <HTML>
2:   <HEAD>
3:   <TITLE>Overview of musicians</TITLE>
4:   </HEAD>
5:   <BODY>
6:   <H1>An overview of my favorite musicians</H1>
7:   <HR>
8:   <P>My favorite musicians are:</P>
9:   <APPLET
10:      code="com.ms.xml.dso.XMLDSO.class"
11:      id="xmldso"
12:      width="0"
13:      height="0"
14:      mayscript="true">
15:  <PARAM NAME="URL" VALUE="musicians.xml">
16:  </APPLET>
17:  <TABLE BORDER="2" CELLPADDING="3" CELLSPACING="2" width="40%"
➥DATASRC="#xmldso">
18:  <THEAD>
19:  <TH>Musician</TH>
20:  <TH>Instrument</TH>
21:  </THEAD>
22:  <TR>
23:      <TD><SPAN DATAFLD="name"></SPAN></TD>
24:      <TD><SPAN DATAFLD="instrument"></SPAN></TD>
25:  </TR>
26:  </TABLE>
27:  <HR>
28:  <!-- some other stuff -- >
29:  </BODY>
30:  </HTML>
```

Abbildung 12.3 zeigt unsere Musikerdaten in tabellarischem Format im Internet Explorer 4.0.

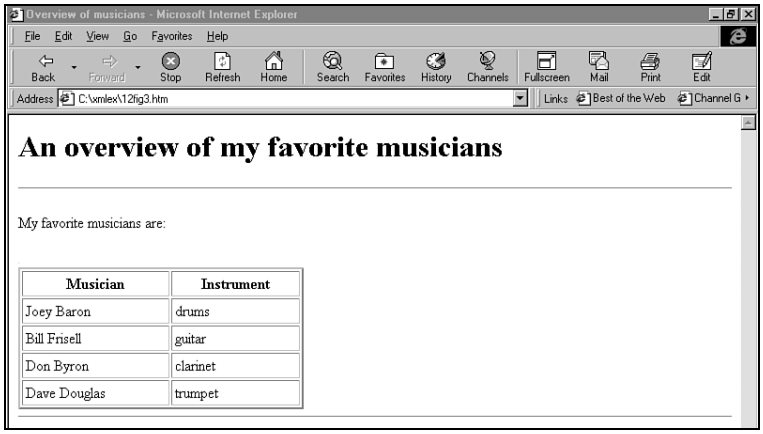


Abbildung 12.3: Unsere Daten im tabellarischen Format im Internet Explorer 4.0.

Sie haben damit das leistungsfähige Konzept kennengelernt, Daten vom Server auf den Client zu verlagern. Das Filtern, Sortieren und Berechnen kann damit auf der Client-Seite erfolgen, so daß Bandbreite gespart wird.

Seien Sie jedoch vorsichtig. Diese Funktion der Datenbindung ist Teil der Microsoft-Implementation von Dynamic HTML, das höchst inkompatibel mit den Implementierungen von Netscape ist.

Mit anderen Worten: Dieser Mechanismus funktioniert nur im Internet Explorer.

Anzeige von XML mit dem XML Object API

Dieser Abschnitt zeigt die Verwendung von JavaScript für den Zugriff auf das XML Object Model (API) für die Umwandlung von XML in HTML.

Das im Internet Explorer 4 verwendete Objektmodell wurde durch das von der W3C definierte DOM (*Document Object Model*) abgelöst, das in der Version 5 implementiert ist.

Ein XML-Dokument wird in Baumstruktur modelliert. Diese Struktur beginnt mit einem Element der obersten Ebene (der Wurzel) und verzweigt sich in Richtung der untergeordneten Elemente.

Es verwendet drei Objektklassen:

- XML-Document-Objekt
- XML-Element-Objekt
- XLM-`Element Collection`-Objekt

Abbildung 12.4 zeigt eine grafische Darstellung dieses Modells.

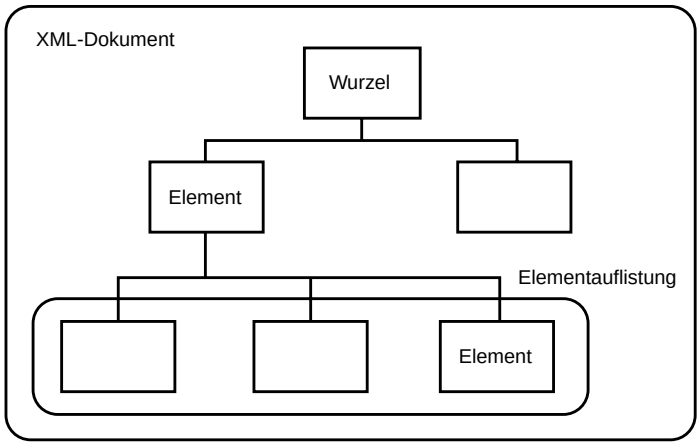


Abbildung 12.4: Das XML-Objektmodell im Internet Explorer 4.

Das `Document`-Objekt stellt das XML-Quelldokument dar. Es beinhaltet den Elementbaum sowie Dokumentinformationen, wie beispielsweise den verwendeten Zeichensatz, den Dokumenttyp, die Dateigröße usw. Nachdem dieses Objekt angelegt wurde, können Sie auf seine Informationen zugreifen, indem Sie seine Eigenschaften mit Hilfe seiner Methoden manipulieren.

Beispiele für XML-`Document`-Objekteigenschaften finden Sie in Tabelle 12.2.

Name	Rückgabe
`Root`	Das Wurzelelement des Dokuments.
`Charset`	Ein String, der den vom Eingabedokument verwendeten Zeichensatz beschreibt.
`Version`	Die Version der verwendeten XML-Spezifikation.
`FileSize`	Die Dateigröße des XML-Dokuments.

Tabelle 12.2: XML-Dokument-Objekteigenschaften

Die vollständige Liste verfügbarer Eigenschaften und Methoden finden Sie unter der URL `http://www.microsoft.com/xml/articles/xmlmodel.asp`.

Ein Dokument wird als Baum mit Knoten modelliert, wobei das Wurzelelement alle anderen Elemente (Knoten) aufnimmt, die wiederum andere Knoten enthalten usw., bis Sie schließlich die Blätter des Baums erreichen, wobei es sich um leere Elemente, Text, Kommentare und Verarbeitungsanweisungen handeln kann.

Im Internet Explorer 4 unterscheidet das XML-Objektmodell fünf Arten von Elementobjekten. Die drei wichtigsten sind:

- `ELEMENT`-Typ – Bei dem Knoten handelt es sich um einen Container oder ein leeres Element.
- `TEXT`-Typ – Der Blattknoten enthält PCDATA oder CDATA.
- `COMMENT`-Typ – Für Kommentare.

Nachdem ein Elementobjekt angelegt wurde, können Sie auf seine Informationen zugreifen, indem Sie seine Eigenschaften anzeigen bzw. diese mit seinen Methoden manipulieren.

Die wichtigsten Eigenschaften der XML-Elementobjekte sehen Sie in Tabelle 12.3.

Name	Ergebnis
Type	0 für `ELEMENT`-Typ, 1 für `TEXT`-Typ usw. für die weiteren Typen.
Tag	Eine Zeichenkette aus Großbuchstaben, die den Namen des Tags enthält.
Text	Der Textinhalt eines Elements oder Kommentars.
Children	Eine Auflistung der Kindelemente des betreffenden Elements.

Tabelle 12.3: XML-Element-Objekteigenschaften

Tabelle 12.4 listet praktische Methoden für XML-Elementobjekte auf.

Name	Funktion
`setAttribute()`	Setzt einen Attributwert.
`removeAttribute()`	Entfernt ein Attribut.
`addChild()`	Fügt ein Kindelement hinzu.
`removeChild()`	Entfernt ein Kindelement.

Tabelle 12.4: XML-Element-Objektmethoden

Das `Element Collection`-Objekt wird für die Manipulation der Kinder eines Elements verwendet. Dabei besteht seine Hauptaufgabe, ein Durchlaufen der Kindelemente zu ermöglichen.

Die einzige Eigenschaft für das XML-`Element Collection`-Objekt ist in Tabelle 12.5 erklärt.

Name	Rückgabe
`length`	Die Länge einer Auflistung (die Anzahl der Knoten)

Tabelle 12.5: XML-Element-Collection-Objekt Eigenschaften

Die einzige Methode des XML-`Element Collection`-Objekts ist in Tabelle 12.6 erklärt.

Name	Funktion
`item`(index)	Das angeforderte Element an der durch den Indexparameter angegebenen Position (Nummer). Mit anderen Worten, `item(0)` gibt das erste Kindelement zurück.
`item`(elementname)	Eine Auflistung aller Elemente mit diesem Namen.
`item`(elementname, index)	Das Element an der durch den Index angegebenen Position (Nummer) in der Elementauflistung mit diesem bestimmten Namen.

Tabelle 12.6: XML-Element-Collection-Objektmethoden

Wir werden viele dieser Eigenschaften und Methoden gleich in einem Beispiel wiedersehen.

Dieses Objektmodell ist sprachneutral und kann für JavaScript, VBScript, C++ oder Java verwendet werden.

Anhand eines JavaScript-Beispiels lesen wir unsere `musicians.xml`-Datei aus und wandeln sie in HTML um, indem wir die für die verschiedenen Objekte (`Document`-Objekt, `Element` -Objekt, `Element Collection`-Objekt) definierten Eigenschaften und Methoden aufrufen.

Das Ergebnis ist eine Tabelle für alle Musiker. Für jeden Musiker gibt es eine Tabellenzeile, die die Werte für den Namen, das Instrument und die Anzahl der Aufnahmen in separaten Zellen enthält.

Die Datei `musicians.xml` sehen Sie in Listing 12.1. Wir haben eine Datei mit strukturierten Daten. Als nächstes legen wir ein Dokumentobjekt an:

```
var xml = new ActiveXObject("msxml");
```

Anschließend geben Sie die XML-Datendatei an, die von dem XML-Dokumentobjekt geladen werden soll:

```
xml.URL = "file:///c¦/xmlex/musicians.xml";
```

Überlegen Sie, was das restliche Programm erledigen soll. Jeder Knoten, der durchlaufen wird, muß daraufhin geprüft werden, ob

- es sich dabei um ein Element handelt
- oder ob es sich um Text handelt.

Handelt es sich um Text, geben wir ihn einfach aus. Handelt es sich um ein Element, müssen wir wissen, wie der Tag-Name aussieht:

- `musicians` – Wir müssen die Tabelle anlegen.
- `musician` – Für jeden Musiker brauchen wir eine neue Zeile.
- Irgend etwas anderes – Für alles andere (`name`, `instrument`, `NrOfRecordings`) benötigen wir jeweils eine Tabellenzelle.

Wir haben diese Logik in JavaScript implementiert, wie in Listing 12.10 gezeigt.

Listing 12.10: Die JavaScript-Funktion *output_doc(elem)*

```
1:   function output_doc(elem)
2:   {
3:   if (elem.type == 0) //Prüfen, ob der Knoten ein Element ist
4:           {
5:               if (elem.tagName == "MUSICIANS") //Tagnamen Großbuchstaben
6:               {
7:                   document.write("<TABLE BORDER='1' CELLPADDING='5'>");
➥//Tabelle anlegen
8:                   traverse(elem); //noch zu definieren; zum Durchlaufen der
Kinder
9:                   document.write("</TABLE>");
10:                      }
11:              else if (elem.tagName == "MUSICIAN")
12:              {
13:                  document.write("<TR>"); //Zeile beginnen
14:                  traverse(elem); //noch zu definieren; zum Durchlaufen der
Kinder
15:                  document.write("</TR>");
16:                      }
17:              else
18:              {
19:                  document.write("<TD>"); //Zelle beginnen
20:                  traverse(elem); //noch zu definieren; zum Durchlaufen der
Kinder
21:                  document.write("</TD>");
22:              }
23:
```

```
24:            }
25:  else if (elem.type==1) //Prüfen, ob der Knotentyp Text ist
26:          document.write(elem.text);
27:      else
28:          alert("Unknown type encountered");
29:    }
```

Die Kinder der einzelnen Elemente werden alle auf dieselbe Weise verarbeitet: dieselben Überprüfungen, wie in Listing 12.10 gezeigt, wurden ausgeführt – ein klarer Fall für eine Rekursion (ein Programm/eine Funktion, die sich selbst aufruft).

In Listing 12.11 deklarieren wir eine Funktion, die alle Kindelemente durchläuft und dabei jeweils die bereits definierte Funktion `output_doc()` aufruft.

Listing 12.11: JavaScript-Funktion *traverse(elem)*

```
1:   function traverse(elem)
2:      {var i;
3:       if (elem.children != null)//falls es Kindelemente gibt
4:          {
5:          for (i=0; i < elem.children.length; i++) //für alle Kinder
6:              output_doc(elem.children.item(i)); //Prüfungen und Ausgaben für
alle Kinder
7:          }
8:          }
```

Jetzt wollen wir den Baum durchlaufen. Dazu müssen wir das Wurzelelement definieren:

```
var docroot = xml.root;
```

Für dieses Wurzelelement rufen wir unsere Funktion `output_doc` auf:

```
ouptput_doc(docroot);
```

Das vollständige Bild sehen Sie in Listing 12.12.

Listing 12.12: HTML-Datei, die die beiden JavaScript-Funktionen benutzt

```
1:   <HTML>
2:   <HEAD><TITLE>XML Object Model in Explorer 4.0</TITLE>
3:   </HEAD>
```

```
4:   <BODY>
5:   <SCRIPT LANGUAGE="JScript" FOR="window" EVENT="onload">
6:     document.write("<HTML><HEAD><TITLE>My favorite
➥musicians</TITLE></HEAD>\n");
7:     document.write("<BODY><H2>My favorite musicians</H2><HR>\n")    ;
8:     var xml = new ActiveXObject("msxml");
9:     xml.URL = "file:///c¦/xmlex/musicians.xml";
10:    var docroot = xml.root;
11:    output_doc(docroot);
12:
13:    function traverse(elem)
14:      {var i;
15:          if (elem.children != null)
16:              {
17:              for (i=0; i < elem.children.length; i++)
18:              output_doc(elem.children.item(i));
19:              }
20:       }
21:
22:    function output_doc(elem)
23:    {
24:          if (elem.type == 0)
25:              {
26:                  if (elem.tagName == "MUSICIANS")
27:                      {
28:                          document.write("<TABLE BORDER='1'
➥CELLPADDING='5'>");
29:                          traverse(elem);
30:                          document.write("</TABLE>");
31:                      }
32:                  else if (elem.tagName == "MUSICIAN")
33:                      {
34:                          document.write("<TR>");
35:                          traverse(elem);
36:                          document.write("</TR>");
37:                      }
38:                  else
39:                      {
40:                          document.write("<TD>");
41:                          traverse(elem);
42:                          document.write("</TD>");
43:                      }
44:
45:               }
46:            else if (elem.type==1)
47:                  document.write(elem.text);
```

```
48:            else
49:                  alert("Unknown type encountered");
50:              }
51:  </SCRIPT>
52:  </BODY>
53:  </HTML>
```

Abbildung 12.5 zeigt, wie es aussieht, wenn diese Datei in den Internet Explorer 4 geladen wird.

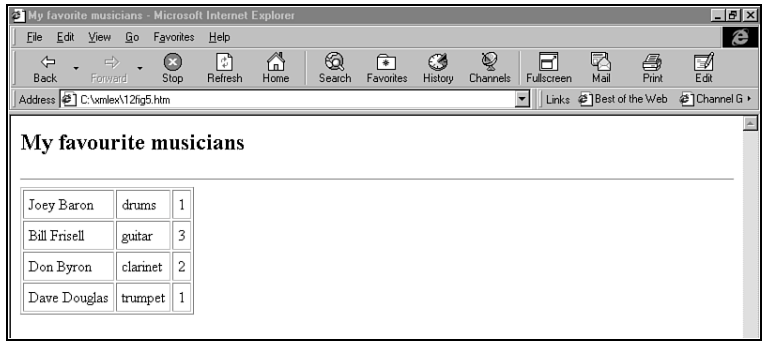


Abbildung 12.5: Das Ergebnis unseres JavaScript im IE4.

Sie sehen, daß dieser Ansatz recht aufwendig ist, um XML im Internet Explorer 4.0 anzuzeigen. Wenn Sie jedoch wirklich eine Programmierung für XML-Daten vornehmen wollen oder Umwandlungen für XML-Daten benötigen, ist diese API einfach und leistungsfähig.

Anzeige von XML mit Hilfe des Microsoft-XSL-Prozessors

XSL steht für Extensible Style Sheet Language. Der erste XSL-Vorschlag wurde zusammen von Microsoft, Inso Corporation und ArborText vorgelegt. Microsoft hat außerdem einen XSL-Prozessor für das Prototyping und zum Testen bereitgestellt. Dieser Prozessor erlaubt, XML-Daten in HTML umzuwandeln.

Beachten Sie bitte, daß eine Umwandlung in HTML stattfindet. Das XML wird nicht unmittelbar dargestellt.

Der MS-XSL-Prozessor stand in zwei Paketen zur Verfügung:

- Im Microsoft-XSL-Befehlszeilen-Utility
- Als Microsoft-XSL-ActiveX-Steuerelement

Die XSL-Syntax, die im MS-XSL-Prozessor verwendet wird, ist völlig veraltet und wurde durch eine neue XSL-Skizze ersetzt.

Aus diesem Grund wird das XSL-Befehlszeilen-Utility nicht mehr auf der Web-Site von Microsoft zur Verfügung gestellt.

Jetzt wollen wir die Arbeitsweise eines XSL-Stylesheet erklären.

Auf der einen Seite brauchen wir eine XML-Datei, beispielsweise `musicians.xml`, die Sie in Listing 12.1 sehen.

Auf der anderen Seite brauchen wir ein ein XSL-Stylesheet, nennen wir es `musicians.xsl`. Seinen Inhalt sehen Sie in Listing 12.13.

Versuchen Sie nicht unbedingt, die Syntax zu verstehen, weil die XSL-Entwicklungsarbeit der W3C noch längst nicht abgeschlossen ist und die Syntax noch wesentlichen Änderungen unterliegt.

Listing 12.13: musicians.xsl

```
<xsl>
<rule>
  <root/>
  <HTML>
    <HEAD>
        <TITLE>My favorite musicians</TITLE>
    </HEAD>
    <BODY>
        <H1>My favorite musicians</H1><HR/>
        <TABLE BORDER="2" CELLPADDING="5">
        <children/>
        </TABLE>
    </BODY>
  </HTML>
</rule>
<rule>
    <target-element type="musician"/>
    <TR>
    <children/>
```

```
20:      </TR>
21:  </rule>
22:  <rule>
23:      <target-element type="name"/>
24:      <TD>
25:      <children/>
26:      </TD>
27:  </rule>
28:  <rule>
29:      <target-element type="instrument"/>
30:      <TD>
31:      <children/>
32:      </TD>
33:  </rule>
34:  <rule>
35:      <target-element type="NrOfRecordings"/>
36:      <TD>
37:      <children/>
38:      </TD>
39:  </rule>
40:  </xsl>
```

Um das XSL-ActiveX-Steuerelement nutzen zu können, legen Sie das folgende `OBJECT`-Element wie in Listing 12.14 gezeigt auf Ihrer HTML-Seite ab.

***Listing 12.14:** `OBJECT`-Element, das auf das XSL-Steuerelement verweist*

```
1:  <OBJECT ID="XSLControl"
2:  CLASSID="CLSID:2BD0D2F2-52EC-11D1-8C69-0E16BC000000"
3:  codebase="http://www.microsoft.com/xml/xsl/msxsl.cab"
4:  style="display:none">
5:  </OBJECT>
```

Dieses Objekt lädt das XSL-ActiveX-Steuerelement automatisch herunter und installiert es.

Als nächstes lassen Sie das Steuerelement auf das XML-Dokument und das XSL-Stylesheet zeigen, wie in Listing 12.15 gezeigt.

***Listing 12.15:** `OBJECT`-Element, das auf das XML-Dokument und das XSL-Stylesheet verweist*

```
1:  <OBJECT ID="XSLControl"
2:  CLASSID="CLSID:2BD0D2F2-52EC-11D1-8C69-0E16BC000000"
3:  codebase="http://www.microsoft.com/xml/xsl/msxsl.cab"
4:  style="display:none">
```

```
5:    <PARAM NAME="documentURL" VALUE="musicians.xml">
6:    <PARAM NAME="styleURL" VALUE="musicians.xsl">
7:  </OBJECT>
```

Dieses Objekt erzeugt aus dem XML-Dokument einen HMTL-String sowie das zugehörige Stylesheet.

Dieser HTML-String muß irgendwo auf der Seite angelegt werden, siehe Listing 12.16.

Listing 12.16: Skript zum Laden des erzeugten HTML

```
1:  <SCRIPT FOR=window EVENT=onload>
2:    var xslHTML = XSLControl.htmlText;
3:    document.all.item("xslTarget").innerHTML = xslHTML;
4:  </SCRIPT>
```

Dieses Skript hält das HTML zunächst in einer Variablen fest und plaziert sie in dem Element mit der ID `xslTarget`.

Wir müssen also ein Element mit dieser ID auf unserer Seite anlegen:

```
<DIV id=xslTarget></DIV>
```

Das vollständige Bild sehen Sie in Listing 12.17.

Listing 12.17: Laden des erzeugten HTML

```
1:  <HTML>
2:    <HEAD>
3:      <TITLE>My favorite musicians</TITLE>
4:        <SCRIPT FOR=window EVENT=onload>
5:        var xslHTML = XSLControl.htmlText;
6:        document.all.item("xslTarget").innerHTML = xslHTML;
7:      </SCRIPT>
8:    </HEAD>
9:    <BODY>
10:         <OBJECT ID="XSLControl"
11:                 CLASSID="CLSID:2BD0D2F2-52EC-11D1-8C69-0E16BC000000"
12:                 CODEBASE="http://www.microsoft.com/xml/xsl/msxsl.cab"
13:                 STYLE="display:none">
14:            <PARAM NAME="documentURL" VALUE="musicians.xml">
```

```
15:               <PARAM NAME="styleURL" VALUE="musicians.xsl">
16:       </OBJECT>
17:         <DIV id="xslTarget"></DIV>
18:   </BODY>
19:   </HTML>
```

Anzeige von XML im Internet Explorer 5

XML kann im Internet Explorer 5 auf unterschiedliche Weise genutzt werden. Diese verschiedenen Möglichkeiten werden wir jetzt genauer betrachten.

Überblick über die XML-Unterstützung im Internet Explorer 5

Neben der bereits im Internet Explorer 4 existierenden XML-Unterstützung können Sie im Release 5 die folgenden Dinge erwarten:

- Vollständige Konformität mit der XML-1.0-Spezifikation
- Direkte Anzeige von XML unter Verwendung von XSL oder CSS (Cascading Style Sheets)
- Einen Mechanismus zum Einbetten von XML in HTML, die sogenannten »Dateninseln«
- Unterstützung von Namensräumen
- Bessere Performance
- Verbesserte Robustheit

Anzeige von XML mit Hilfe des XML-Data-Source-Objekts

Die Internet Explorer 5 wird mit einem neuen C++-Data-Source-Objekt ausgeliefert, das eine bessere Leistung bietet ebenso wie die Möglichkeit, eine direkte Bindung an eine XML-Dateninsel (»Data Island«) vorzunehmen, ohne daß Sie ein `APPLET`- oder `OBJECT`-Tag benötigen, wie in Listing 12.18 gezeigt.

Listing 12.18: Verwendung des XML-Data Source-Objekts im Internet Explorer 5 Beta 2

```
1:   <HTML>
2:   <HEAD>
3:   <TITLE>Overview of musicians</TITLE>
```

```
4:   </HEAD>
5:   <BODY>
6:   <H1>An overview of my favorite musicians</H1>
7:   <HR>
8:   <P>My favorite musicians are:</P>
9:   <XML ID="xmldso" src="musicians.xml"></XML>
10:  <TABLE BORDER="2" CELLPADDING="3" CELLSPACING="2" width="40%"
➥DATASRC="#xmldso">
11:  <THEAD>
12:  <TH>Musician</TH>
13:  <TH>Instrument</TH>
14:  </THEAD>
15:  <TR>
16:      <TD><SPAN DATAFLD="name"></SPAN></TD>
17:      <TD><SPAN DATAFLD="instrument"></SPAN></TD>
18:  </TR>
19:  </TABLE>
20:  <HR>
21:  <!-- some other stuff -- >
22:  </BODY>
23:  </HTML>
```

Anzeige von XML mit Hilfe des XML Object API

Im W3C denkt man viel über die Definition eines standardisierten Dokumentobjektmodells nach.

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

Microsoft versucht, diese neuen Entwicklungen stets nachzuvollziehen. Deshalb wurde das im Internet Explorer 5 verwendete Dokumentobjektmodell auch geändert und erweitert.

Das Objektmodell, das Microsoft im IE5 einsetzt, verwendet fünf Basisklassen aus Objekten:

- `Document`-Objekt
- `Node`-Objekt
- `NodeList`-Objekt
- `NameNodeMap`-Objekt

Abbildung 12.6 zeigt eine grafische Darstellung dieses Modells.

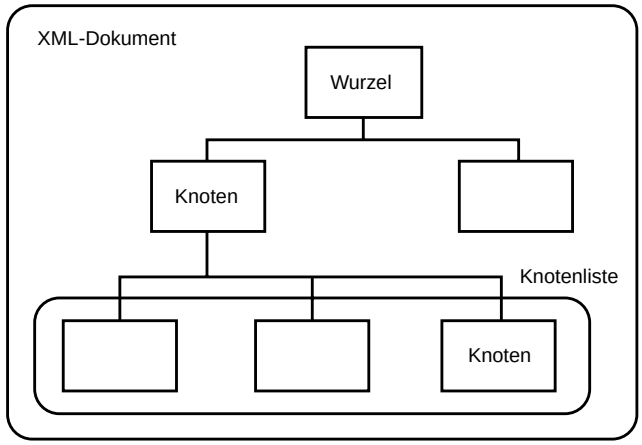


Abbildung 12.6: Das XML-Objektmodell im Internet Explorer 5.

Beachten Sie die Unterschiede zu den Namen der vorherigen Version: `Node` statt `Element`, `NodeList` statt `Element Collection`.

Jedes dieser Objekte besitzt mehrere Eigenschaften und Methoden, die Ihnen ermöglichen, auf Informationen über diese Objekte zuzugreifen und sie zu ändern.

Die Liste der Eigenschaften und Methoden wurde gegenüber der vorherigen Version erweitert.

Das `Document`-Objekt stellt den obersten Knoten im Baum dar.

Nachdem dieses Objekt angelegt wurde, können Sie auf seine Informationen zugreifen und es durch Aufruf seiner Eigenschaften und Methoden manipulieren.

Ein Beispiel für eine `Document`-Objekteigenschaft sehen Sie in Tabelle 12.7.

Name	Rückgaben
`DocumentElement`	Das Wurzelelement der XML-Datei.

Tabelle 12.7: `IDOMDocument`-Objekteigenschaften

Ein Beispiel für eine `Document`-Objektmethode sehen Sie in Tabelle 12.8.

Name	Funktion
`createElement(tagName)`	Gibt ein Element mit dem Namen `tagName` zurück.

Tabelle 12.8: `IDOMDocument`-Objektmethode

Die vollständige Beschreibung aller verfügbaren Eigenschaften und Methoden finden Sie unter `http://www.microsoft.com/workshop/c-frame.htm?913723996859#/xml/default.asp`.

Das `Node`-Objekt stellt einen Knoten in einem XML-Dokument dar.

Das XML-Objektmodell im Internet Explorer 5 unterscheidet 12 Knotentypen:

- ELEMENT-Typ
- ATTRIBUTE-Typ
- TEXT-Typ
- CDATA_SECTION-Typ
- ENTITY_REFERENCE-Typ
- ENTITY-Typ
- PROCESSING_INSTRUCTION-Typ
- COMMENT-Typ
- DOCUMENT-Typ
- DOCUMENT_TYPE-Typ
- DOCUMENT_FRAGMENT-Typ
- NOTATION-Typ

Nachdem dieses `Node`-Objekt angelegt wurde, können Sie durch Aufruf seiner Eigenschaften und Methoden auf seine Informationen zugreifen und sie manipulieren.

Die wichtigsten `Node`-Objekteigenschaften sind in Tabelle 12.9 aufgelistet.

Name	Funktion
`NodeType`	Gibt 0 für den Typ `element` zurück, 1 für den Typ `attribute` usw. für die anderen Typen.
`NodeName`	Gibt den Namen eines Knotens zurück oder setzt ihn.
`NodeValue`	Gibt den Wert (Text) eines Knotens zurück oder setzt ihn.
`ChildNode`	Gibt eine Auflistung der Kindknoten des angegebenen Knotens zurück.

Tabelle 12.9: `IDOMNode`-Objekteigenschaften

Tabelle 12.10 listet einige `Node`-Objektmethoden auf.

Name	Funktion
`insertBefore(newChild, refChild)`	Fügt einen durch den Parameter `newChild` definierten Kindknoten links von dem angegebenen Knoten `refChild` oder am Ende ein.
`removeChild(oldChild)`	Entfernt den angegebenen Kindknoten `oldChild`.

Tabelle 12.10: IDOMNode-Objektmethoden

Das `NodeList`-Objekt wird für die Manipulation der Kinder eines Knotens verwendet. Hauptsächlich wird es genutzt, um die Kindknoten zu durchlaufen.

Die einzige `NodeList`-Objekteigenschaft sehen Sie in Tabelle 12.11.

Name	Rückgabe
`length`	Die Länge einer Knotenliste (also die Anzahl der Knoten).

Tabelle 12.11: IDOMNodeList-Objekteigenschaft

Ein Beispiel für eine `NodeList`-Objektmethode finden Sie in Tabelle 12.12.

Name	Funktion
`item()`	Gibt das angeforderte Element aus den aufgelisteten Knoten zurück.

Tabelle 12.12: IDOMNOdeList-Objektmethoden

Wir werden später in einem Beispiel viele dieser Eigenschaften und Methoden einsetzen.

Dieses Objektmodell ist sprachneutral und kann in JavaScript, VBScript, C++ oder Java verwendet werden.

Aufgrund der Änderungen im Objektmodell zwischen den Versionen 4.0 und 5.0 müssen wir das in Listing 12.12 gezeigte Programm umschreiben. Die neue Version sehen Sie in Listing 12.19.

Listing 12.19: Verwendung des Dokumentobjektmodells im Internet Explorer 5 Beta 2

```
1:   <HTML>
2:   <HEAD><TITLE>XML Object Model in Explorer 5.0</TITLE>
3:   </HEAD>
4:   <BODY>
5:   <SCRIPT LANGUAGE="JScript" FOR="window" EVENT="onload">
6:     document.write("<HTML><HEAD><TITLE>My favorite
```

```
➥musicians</TITLE></HEAD>\n");
7:     document.write("<BODY><H2>My favorite musicians</H2><HR>\n")    ;
8:     var xml = new ActiveXObject("microsoft.xmldom");
9:     xml.load(musicians.xml);
10:    var docroot = xml.documentElement;
11:    output_doc(docroot);
12:
13:    function traverse(node)
14:      {var i;
15:          if (node.childNodes != null)
16:              {
17:              for (i=0; i < node.childNodes.length; i++)
18:              output_doc(node.childNodes.item(i));
19:              }
20:       }
21:
22:    function output_doc(node)
23:    {
24:          if (node.nodeType == 0)
25:              {
26:                  if (node.nodeName == "musicians")
27:                      {
28:                          document.write("<TABLE BORDER='1'
➥CELLPADDING='5'>");
29:                          traverse(node);
30:                          document.write("</TABLE>");
31:                      }
32:                  else if (node.nodeName == "musician")
33:                      {
34:                          document.write("<TR>");
35:                          traverse(node);
36:                          document.write("</TR>");
37:                      }
38:                  else
39:                      {
40:                          document.write("<TD>");
41:                          traverse(node);
42:                          document.write("</TD>");
43:                      }
44:
45:               }
46:           else if (node.nodeType==1)
47:                 document.write(node.nodeValue);
48:           else
49:                 alert("Unknown type encountered");
50:             }
```

```
51:  </SCRIPT>
52:  </BODY>
53:  </HTML>
```

Jetzt betrachten wir die Codeunterschiede zwischen der neuen (für IE5) und der älteren Version (für IE4):

- Das ActiveX-Objekt, das das XML-Dokumentobjekt erzeugt, hat die `progID` `microsoft.xmldom`.
- Um die XML-Datendatei anzugeben, die von dem XML-Dokumentobjekt geladen wird, müssen wir die Lademethode aufrufen und ihr die URL der zu ladenden XML-Datei übergeben.
- Zur Kennzeichnung des Wurzelelements verwenden wir die `documentElement`-Eigenschaft.
- Im eigentlichen Skript verwenden wir die Eigenschaften und Methoden der `Node`- und `Nodelist`-Objekte.
- Die für `musicians` und `musician` ausgewerteten Tag-Namen werden jetzt in Kleinbuchstaben dargestellt.

Die Implementierung von Version 4 der `tagName`-Eigenschaft des `Element`-Objekts gab den Namen in Großbuchstaben zurück. Bei der `nodeName`-Eigenschaft eines `Node`-Objekts (Version 5) ist das nicht mehr der Fall.

Anzeige von eingebettetem XML

Mit dem neuesten Release des Internet Explorer ist es möglich, sogenannte Dateninseln (*Data Islands*) in HTML-Seiten einzubetten. Diese Dateninseln werden mit XML gekennzeichnet.

Dazu verwenden Sie das XML-Tag auf Ihrer HTML-Seite. Um die eigentlichen XML-Daten zu erhalten, gibt es zwei Möglichkeiten:

- Sie nehmen die XML-Daten in das XML-Element auf.
- Sie verwenden eine URL, die auf die XML-Daten verweist.

Listing 12.20 zeigt ein Beispiel für die im XML-Element enthaltenen XML-Daten.

Listing 12.20: Einbetten von XML in eine HTML-Seite

```
1:   <HTML>
2:   <HEAD><TITLE>XML Object Model in Explorer 5.0</TITLE>
3:   </HEAD>
```

```
4:   <BODY>
5:   <H2>My favorite musicians</H2>
6:   <HR>
7:   <XML id="myfavs">
8:   <musicians>
9:   <musician>
10:  <name>Joey Baron
11:  </name>
12:  <instrument>drums
13:  </instrument>
14:  <NrOfRecordings>1
15:  </NrOfRecordings>
16:  </musician>
17:  <musician>
18:  <name>Bill Frisell
19:  </name>
20:  <instrument>guitar
21:  </instrument>
22:  <NrOfRecordings>3
23:  </NrOfRecordings>
24:  </musician>
25:  <musician>
26:  <name>Don Byron
27:  </name>
28:  <instrument>clarinet
29:  </instrument>
30:  <NrOfRecordings>2
31:  </NrOfRecordings>
32:  </musician>
33:  <musician>
34:  <name>Dave Douglas
35:  </name>
36:  <instrument>trumpet
37:  </instrument>
38:  <NrOfRecordings>1
39:  </NrOfRecordings>
40:  </musician>
41:  </musicians>
42:  </XML>
43:  </BODY>
44:  </HTML>
```

Listing 12.21 zeigt ein Beispiel dafür, wie man auf einer HTML-Seite auf eine XML-Datei verweist.

Listing 12.21: Auf einer HTML-Seite kann auf eine XML-Datei verwiesen werden

```
1:   <HTML>
2:   <HEAD><TITLE>XML Object Model in Explorer 5.0</TITLE>
3:   </HEAD>
4:   <BODY>
5:   <H2>My favorite musicians</H2>
6:   <HR>
7:   <XML ID="myfavs" SRC="file:///C¦xmlex/musicians.xml">
8:   </XML>
9:   </BODY>
10:  </HTML>
```

Wenn wir diese Dateien in unserem Browser öffnen, passiert nichts.

Auch hier brauchen Sie das Document Object API, um auf die XML-Daten zuzugreifen. Zunächst müssen Sie jedoch mit dem HTML-Dokumentobjektmodell arbeiten, weil das XML-Tag letztlich ein HTML-Tag ist. Sie können ganz einfach auf das XML-Element verweisen, indem Sie seine ID angeben, `myfavs`.

Nachdem wir das XML-Tag gefunden haben, müssen wir das Wurzelelement unseres XML-Datenbaums finden:

```
myfavs.documentElement
```

Von hier aus können wir den bereits verwendeten Code einsetzen, wie in Listing 12.22 gezeigt.

Listing 12.22: Anzeige eines »Verweises auf« XML auf einer HTML-Seite

```
1:   <HTML>
2:   <HEAD><TITLE>XML Object Model in Explorer 5.0</TITLE>
3:   </HEAD>
4:   <SCRIPT>
5:    function traverse(node)
6:      {var i;
7:          if (node.childNodes != null)
8:              {
9:              for (i=0; i < node.childNodes.length; i++)
10:              output_doc(node.childNodes.item(i));
11:              }
12:       }
13:
14:    function output_doc(node)
15:    {
```

```
16:          if (node.nodeType == 0)
17:              {
18:                  if (node.nodeName == "musicians")
19:                      {
20:                          document.write("<TABLE BORDER='1'
➥CELLPADDING='5'>");
21:                          traverse(node);
22:                          document.write("</TABLE>");
23:                      }
24:                  else if (node.nodeName == "musician")
25:                      {
26:                          document.write("<TR>");
27:                          traverse(node);
28:                          document.write("</TR>");
29:                      }
30:                  else
31:                      {
32:                          document.write("<TD>");
33:                          traverse(node);
34:                          document.write("</TD>");
35:                      }
36:
37:                }
38:            else if (node.nodeType==1)
39:                  document.write(node.nodeValue);
40:            else
41:                  alert("Unknown type encountered");
42:              }
43:
44:  </SCRIPT>
45:  <BODY>
46:  <H2>My favorite musicians</H2>
47: <XML ID="myfavs" src="musicians.xml"></xml>
48:  <SCRIPT>
49:  var root = myfavs.documentElement;
50:  output_doc(root);
51:  </SCRIPT>
52:  <HR>
53:  </BODY>
54:  </HTML>
```

Die beiden folgenden Zeilen sind äquivalent:

```
var root = myfavs.documentElement;
var root = document.all("myfavs").XMLDocument;
```

Direkte Anzeige von XML

Im IE5 kann eine XML-Datei auch direkt geöffnet werden. In diesem Fall wird ein Standard-XSL-Stylesheet angewendet.

Betrachten Sie beispielsweise die in Listing 12.23 gezeigte Datei.

Listing 12.23: helptopic.xml

```
1:  <?xml version="1.0" ?>
2:  <helptopic>
3:   <title keyword="printing,network;printing,shared printer">
4:   How to use a shared network printer?</title>
5:  <procedure>
6:   <step><action>In <icon>Network Neighborhood</icon>,
7:    locate and double-click the computer where the printer
8:   you want to use is located. </action>
9:   <tip targetgroup="beginners">To see which computers have
10:  shared printers attached, click the <menu>View</menu> menu,
11:  click <menu>Details</menu>, and look for printer names or
12:  descriptions in the Comment column of the Network Neighborhood
➥window.</tip>
13:  </step>
14:  <step>
15:  <action>Double click the printer icon in the window that
➥appears.</action>
16:  </step>
17:  <step>
18:  <action>
19:  To set up the printer, follow the instructions on the screen.
20:  </action></step>
21:  </procedure>
22:  <tip>
23:   After you have set up a network printer, you can use it as if
24:   it were attached to your computer. For related topics,
25:   look up &amp;quot;printing&amp;quot; in the Help Index.
26:  </tip>
27:  </helptopic>
```

Abbildung 12.7 zeigt Ihnen, was passiert, wenn Sie diese XML-Datei ohne ein bestimmtes Stylesheet laden.

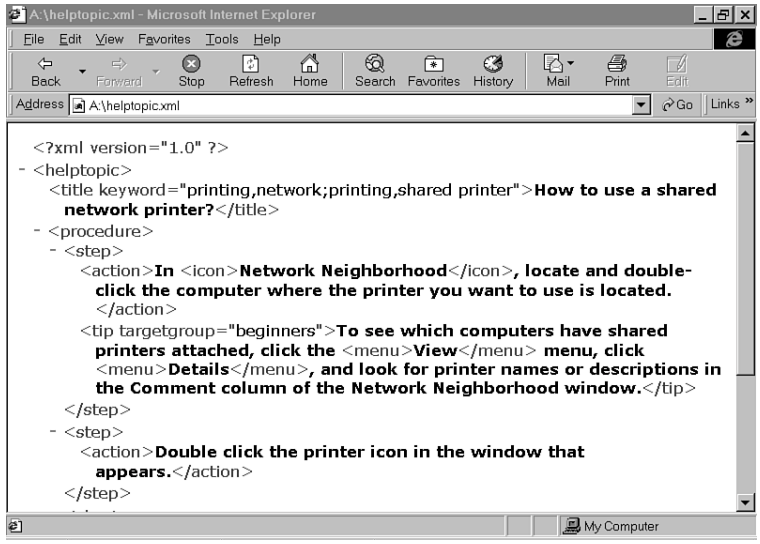


Abbildung 12.7: Direkte Anzeige einer XML-Datei im IE5.

Wenn Sie Ihre XML-Datei jedoch ansprechender präsentieren wollen, sollten Sie ein Stylesheet bereitstellen, anhand dessen die XML-Daten dargestellt werden. Dabei kann es sich um ein XSL- oder um ein CSS-Stylesheet handeln.

Anzeige von XML mit CSS

Das Stylesheet wird angegeben, indem man eine Verarbeitungsanweisung der folgenden Form einfügt:

```
<?xml:stylesheet type="text/css" href="helptopic.css" ?>
```

Wir verwenden die XML-Datei aus Listing 12.22. Die ersten Zeilen sehen Sie in Listing 12.24.

Listing 12.24: helptopic.xml

```
1:   <?xml version="1.0" ?>
2:   <?xml:stylesheet type="text/css" href="helptopic.css" ?>
3:   <helptopic>
4:   <title keyword="printing,network;printing,shared printer">
5:....
```

Der Inhalt unserer `helptopic.css`-Datei könnte aussehen wie in Listing 12.25 gezeigt.

Listing 12.25: helptopic.css

```
1:   helptopic {    display: block;
2:       margin-top:3cm;
3:       margin-left:2cm;
4:       margin-right:2cm;
5:       margin-bottom:6cm;
6:       font-family:Verdana, Arial;
7:       font-size:11pt;
8:       padding:20pt; }
9:
10:  title {display: block;
11:      font-size:20pt;
12:      color:blue;
13:      font-weight:bold;
14:      text-align:center;
15:      margin-bottom:30pt;
16:      text-decoration:underline;}
17:
18:  procedure {display:block;
19:      margin-bottom:30pt}
20:
21:   step {display:block;
22:      margin-bottom:18pt}
23:
24:  action {display:block;
25:      font-weight:bold;}
26:
27:  tip {display:block;
28:      font-size:10pt;
29:      margin-left:+1cm;
30:      margin-top:12pt;
31:      color:blue;}
32:
33:  icon {display:inline;
34:      font-size:12pt;}
35:
36:  todo {display:inline;
```

```
37:      color:red;}
38:
39:  menu {display:inline;
40:      font-style:italic;}
```

Abbildung 12.8 zeigt, was passiert, wenn diese XML-Datei mit einem CSS-Stylesheet geladen wird.

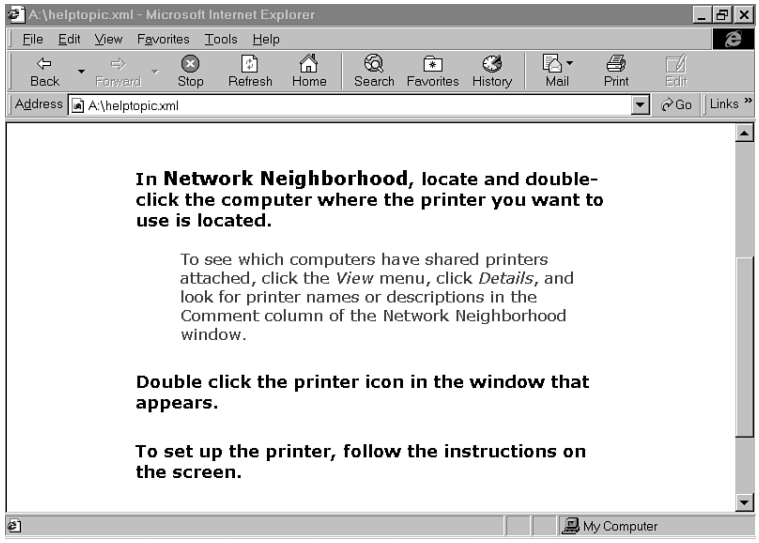


Abbildung 12.8: Anzeige einer XML-Datei mit CSS-Stylesheet in IES.

Anzeige von XML mit XSL

Einem XML-Dokument wird ein XSL-Stylesheet zugeordnet, indem eine Verarbeitungsanweisung der folgenden Form eingefügt wird:

```
<?xml:stylesheet type="text/xsl" href="helptopic.xsl" ?>
```

Die ersten Zeilen sehen Sie in Listing 12.26:

Listing 12.26: HELPTOPIC.XML

```
1:   <?xml version="1.0" ?>
2:   <?xml:stylesheet type="text/css" href="helptopic.css" ?>
3:   <helptopic>
```

```
4:   <title keyword="printing,network;printing,shared printer">
5:....
```

Ein Beispiel für eine `helptopic.xsl`-Datei sehen Sie in Listing 12.27.

Beachten Sie, daß die XSL-Syntax im IE5 nicht mehr mit der neuesten Version vom W3C übereinstimmt, die ständigen Anpassungen unterliegt.

Listing 12.27: helptopic.xsl

```
1:  <?xml version="1.0"?>
2:  <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
3:
4:  <!-- default behaviour, thanks to Ken Holman -->
5:
6:  <xsl:template><xsl:apply-templates/></xsl:template>
7:  <xsl:template match="textnode()"><xsl:value-of/></xsl:template>
8:
9:  <!-- specific behaviour -->
10:
11:  <xsl:template match="/">
12:    <html>
13:      <head>
14:        <title>Using an XSL stylesheet </title>
15:      </head>
16:      <body bgcolor="#FFFFFF">
17:        <xsl:apply-templates/>
18:       </body>
19:    </html>
20:  </xsl:template>
21:
22:  <xsl:template match="title">
23:   <H2>
24:     <xsl:apply-templates/>
25:   </H2>
26:  </xsl:template>
27:
28:  <xsl:template match="procedure">
29:    <OL>
30:      <xsl:apply-templates/>
31:    </OL>
32:  </xsl:template>
33:
34:  <xsl:template match="step">
```

```
35:   <LI>
36:     <xsl:apply-templates/>
37:   </LI>
38:  </xsl:template>
39:
40:  <xsl:template match="action">
41:  <B>
42:    <xsl:apply-templates/>
43:  </B><BR/>
44:  </xsl:template>
45:
46:  <xsl:template match="helptopic/tip">
47:   <H3>Tip!</H3>
48:    <xsl:apply-templates/>
49:  </xsl:template>
50:
51: </xsl:stylesheet>
```

Abbildung 12.9 zeigt, was passiert, wenn Sie diese XML-Datei mit dem ihr zugeordneten XSL-Stylesheet laden.

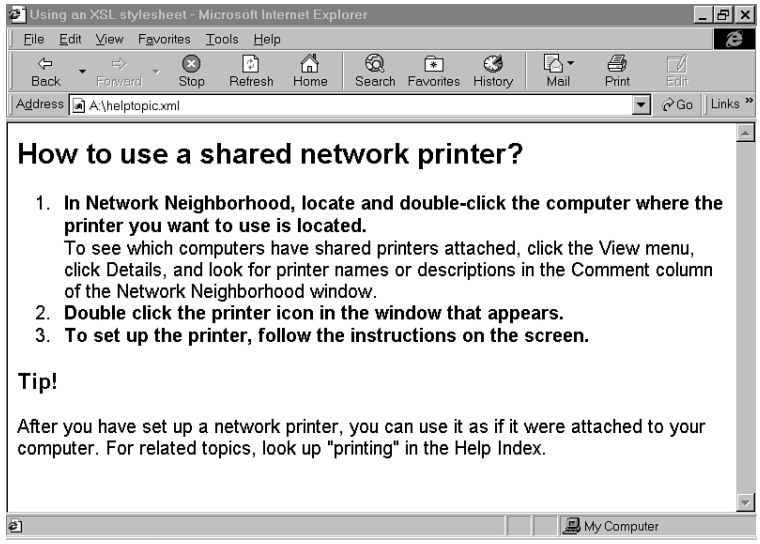


Abbildung 12.9: Anzeige einer XML-Datei mit einem XSL-Stylesheet in IE5.

Zusammenfassung

In diesem Kapitel haben wir beschrieben, wie XML-Daten im Internet Explorer 4 und im Internet Explorer 5 genutzt werden können.

Für jede Version wurden die verschiedenen Möglichkeiten betrachtet:

- XML-Data-Source-Objekt
- DOM API
- CSS-Stylesheets
- XSL-Stylesheets

F&A

F Sollte ich DOM, API, CSS oder XSL verwenden?

A *Wenn man an die Standards denkt, kann man sagen, DOM und CSS sind stabil. Für XSL gilt das nicht – es befindet sich immer noch in der Entwicklungsphase. Was die Implementierung betrifft, ist die Unterstützung von CSS in IE5, so wie sie heute vorliegt, alles andere als vollständig.*

F Ich erhalte im IE5 leere Seiten, wenn ich XML-Dateien öffne.

A *Wahrscheinlich wurde bei der Installation etwas falsch gemacht. Deinstallieren Sie den IE5, und installieren Sie ihn neu.*

Übungen

Laden Sie die folgende Datei mit Hilfe des XML Data Source-Objekts in den Internet Explorer 4.0. Die Daten sind beschränkt und sollten in der folgenden Reihenfolge aufgelistet werden:

- Autor
- Titel
- ISBN-Nummer

Verwenden Sie dieselbe XML-Datei, laden Sie sie aber mit Hilfe des XML-Objektmodell-API in den Internet Explorer, um nur eine Liste der Autoren anzuzeigen.

```
1:   <?xml version="1.0"?>
2:   <books>
3:       <book>
4:           <title>Sam's Teach Yourself C++ in 21 Days, Second Edition
5:           </title>
6:           <author>Jesse Liberty
7:           </author>
8:           <description>This book teaches you the basics of object-
➥oriented programming with C++ and is completely revised to
➥ANSI standards. It can be used with any C++ compiler.
9:           </description>
10:          <ISBN>0-672-31070-8
11:          </ISBN>
12:          <pages>700
13:          </pages>
14:          <targetgroup>Beginning - Intermediate
15:          </targetgroup>
16:          <price unit="USA">29.99
17:          </price>
18:      </book>
19:         <book>
20:          <title>Maximum Java 1.1
21:          </title>
22:          <author>Glenn Vanderburg et al.
23:          </author>
24:          <description>Written by JAVA experts, this book explores
➥the JAVA 1.1 language, tools, and core JAVA API without
➥reviewing fundamentals or basic techniques.
25:          </description>
26:          <ISBN>1-57521-290-0
27:          </ISBN>
28:          <pages>900
29:          </pages>
30:          <targetgroup>Expert
31:          </targetgroup>
32:          <price unit="USA">49.99
33:          </price>
34:      </book>
35:      <book>
36:          <title>JavaScript Unleashed, Second Edition
37:          </title>
38:          <author>Richard Wagner et al.
39:          </author>
40:         <description>This book helps you thoroughly understand
➥and apply JavaScript.
41:          </description>
```

```
42:           <ISBN>1-57521-306-0
43:           </ISBN>
44:           <pages>1000
45:           </pages>
46:           <targetgroup>Casual - Experienced
47:           </targetgroup>
48:           <price>49.99
49:           </price>
50:       </book>
51:       <book>
52:           <title>Sam's Teach Yourself Visual C++ 5 in 21 Days, Fourth
➥Edition
53:           </title>
54:           <author>Nathan and Ori Gurewich
55:           </author>
56:           <description>This book merges the power of the best-selling
➥"Teach Yourself" series with the knowledge of Nathan and
➥Ori Gurewich, renowned experts in code, creating the most
➥efficient way to learn Visual C++.
57:           </description>
58:           <ISBN>0-672-31014-7
59:           </ISBN>
60:           <pages>832
61:           </pages>
62:           <targetgroup>New - Casual
63:           </targetgroup>
64:           <price>35.00
65:           </price>
66:       </book>
67:   </books>
```

Anzeige von XML in anderen Browsern

Woche 2

In Kapitel 12 haben wir beschrieben, wie XML im Microsoft Internet Explorer Version 4 und 5 genutzt werden kann.

Heute werden wir zeigen, wie Sie XML in anderen Browsern anzeigen, unter anderem:

- Netscape Navigator/Mozilla/Gecko
- DocZilla
- Browser, die auf der Inso/Synex Viewport-Engine basieren

Anzeige/Browsing von XML in Netscape Navigator/ Mozilla/Gecko

In diesem Teil beschreiben wir, wie XML in Navigator/Mozilla/Gecko verarbeitet wird.

Netscapes Vision von XML

Netscape sieht die Verwendung von XML für drei Informationstypen vor: Dokumente, Daten und Metadaten.

Für Dokumente verwendet Netscape XML mit CSS (*Cascading Style Sheets*). Was XSL betrifft, wartet Netscape einfach ab, weil es sich dabei noch nicht um einen stabilen Standard handelt.

Für Daten und Metadaten bevorzugt Netscape RDF (*Resource Description Framework*).

RDF ist ein Rahmen für die Beschreibung und den Austausch von Metadaten. Es handelt sich dabei um ein Modell für die Beschreibung von Ressourcen – Ressourcen mit Eigenschaften, die bestimmte Werte annehmen können.

Eine *Ressource* ist ein Objekt, das eine URI (*Uniform Resource Identifier*) haben kann.

EigenschaftsTyp ist der Name der Eigenschaft.

Wert ist der Wert der Eigenschaft.

Eigenschaftstypen und Werte können wiederum selbst Ressourcen sein.

Ein Beispiel für diese Konzepte:

- Resource = http://www.macmillan.com/history/history.htm
- EigenschaftsTyp = `lastupdated`
- Wert = 2/12/1998

Die Beziehungen zwischen Ressourcen, Eigenschaftstypen und Werten werden durch einen gerichteten, beschrifteten Graphen dargestellt, wie in Abbildung 13.1 gezeigt.

Ein Graph ist eine Auflistung von Punkten, wobei einige Punktepaare durch Linien miteinander verbunden sind. In einem *gerichteten Graphen* besitzen diese Linien Pfeilköpfe, die eine Richtung vorgeben.

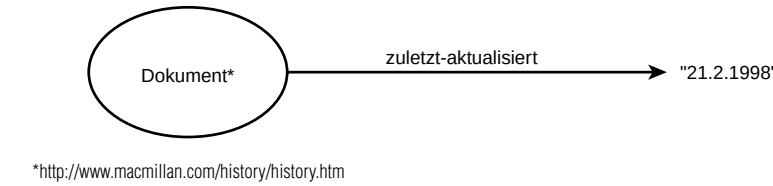


Abbildung 13.1: Unser Beispiel, dargestellt in einem gerichteten, beschrifteten Graphen.

Man geht davon aus, daß XML das Format der Wahl für den Austausch dieser (Meta-)Daten wird.

Mehr über RDF erfahren Sie unter `http://www.w3.org/TR/WD-rdf-syntax/`.

Anzeige von XML im Netscape Navigator 4

Der Netscape Navigator 4 unterstützt XML nicht.

Wenn Sie XML im Netscape Navigator 4 verwenden wollen, müssen Sie es zuvor (außerhalb der Netscape-Umgebung) in HTML oder in ein anderes Format, das der Navigator 4 versteht, umwandeln.

Anzeige von XML in Mozilla 5/Gecko

Netscape hat den Quellcode seines Browsers als Public Domain bereitgestellt, wo er von der Free Software Community unter dem Namen Mozilla weiterentwickelt wird.

Informationen über das Mozilla-Projekt finden Sie auf den folgenden Web-Sites:

- Allgemeine Informationen finden Sie unter `http://www.mozilla.org/`.
- Die FAQ für das Mozilla-Projekt befindet sich unter `http://junior.apk.net/~qc/dok/mozilla_faq`.
- Die kompilierten Programmdateien erhalten Sie unter `http://www.mozilla.org/binaries/html`.

Gecko ist der Name einer Komponente (eines Moduls) der Mozilla-Entwicklungsbemühungen: die neue Layout-Engine. Sie soll ein einbettbares Objekt darstellen und wird von Anwendungsentwicklern genutzt, um Anwendungen Web-Browser-Funktionalität zu geben.

Die Gecko-Homepage finden Sie unter http://developer.netscape.com/software/communicator/ngl/index.html.

Die XML-Unterstützung von Mozilla:

- Er beinhaltet den Expat-Parser von James Clark.
- Er implementiert XML&CSS-Unterstützung auf Dokumentebene.
- Er beinhaltet ein API für das XML-DOM (Dokumentobjektmodell) via JavaScript und Plug-ins.
- Er unterstützt XLink.
- Er unterstützt Elemente aus dem HTML-Namensraum.

Was die Standards betrifft, unterstützt Mozilla Teile von CSS 2.0 und außerdem vollständig die fehlenden Empfehlungen:

- Die XML-1.0-Empfehlung
- Die DOM-1.0-Empfehlung
- Die CSS-1.0-Empfehlung

Um eine XML-Datei in Mozilla anzuzeigen, definieren Sie ein CSS-Stylesheet mit Formatangaben. Es wird in einer separaten Datei abgelegt. Sie können Ihrer XML-Datei die CSS-Datei zuordnen, indem Sie eine Verarbeitungsanweisung in die XML-Datei aufnehmen.

Listing 13.1 zeigt eine XML-Datei, die ein Hilfethema für ein Online-Hilfesystem beschreibt.

Listing 13.1: *helptopic.xml*

```
1:   <?xml version="1.0" ?>
2:   <?protext objid="I5678" ?>
3:   <!DOCTYPE helptopic SYSTEM "dtdv.dtd" [
4:   <!ENTITY doubleclick "<todo>Double-click</todo>">
5:   ]>
6:   <helptopic>
7:   <title keyword="printing,network;printing,shared printer">
➥How to use a shared network printer?</title>
8:   <procedure>
9:   <step><action>In <icon>Network Neighborhood</icon>,
➥locate and double-click the computer where the printer
➥you want to use is located. </action>
10:  <tip targetgroup="beginners">To see which computers
➥have shared printers attached, click the <menu>View</menu> menu,
➥click <menu>Details</menu>, and look for printer names or descriptions
➥in the Comment column of the Network Neighborhood window.</tip>
11:  </step>
12:  <step>
13:  <action>&doubleclick; the printer icon in the window that appears.</action>
14:  </step>
15:  <step>
16:  <action>
17:  To set up the printer, <xref linkend="id45">follow the instructions</xref>
➥on the screen.
18:  </action></step>
19:  </procedure>
20:  <tip>
21:  After you have set up a network printer, you can use it as if it were
➥attached to your computer. For related topics, look up &quot;printing&quot;
➥in the Help Index.
22:  </tip>
23:  </helptopic>
```

Wir wollen unserer XML-Datei ein Stylesheet zuordnen, `helptopic.css`. Dazu fügen wir unserer XML-Datei die folgende Verarbeitungsanweisung (PI, Processing Instruction) hinzu:

```
<?xml-stylesheet type="text/css2" href="helptopic.css"?>
```

Damit sieht unsere XML-Datei aus wie in Listing 13.2 gezeigt.

Listing 13.2: *helptopic.xml* ***mit eingefügten Verarbeitungsanweisungen***

```
1:  <?xml version="1.0" ?>
2:  <?xml-stylesheet type="text/css2" href="helptopic.css"?>
3:  <?protext objid="I5678" ?>
4:  <!DOCTYPE helptopic SYSTEM "dtdv.dtd" [
5:  <!ENTITY doubleclick "<todo>Double-click</todo>">
6:  ]>
7:  <helptopic>
8:  ...
```

In dieser Verarbeitungsanweisung verweisen wir auf eine CSS-Datei, `helptopic.css`.

Weil es keine feststehende Semantik für unsere XML-Elemente gibt (also wie ein Element dargestellt werden oder wie es sich verhalten soll), müssen wir zuerst angeben, ob ein Element *inline* ist (keine Zeilenumbrüche verursacht) oder auf *Blockebene* vorliegt (Zeilenumbrüche verursacht).

Ein Beispiel für ein Element auf Blockebene ist das `action`-Element, das in CSS wie folgt aufgefangen wird:

```
action {display:block}
```

`todo` ist ein Beispiel für ein Inline-Element, das in CSS-Syntax übersetzt wie folgt aussieht:

```
todo {display:inline}
```

Das restliche Stylesheet enthält die bekannten Formatangaben, wie in Listing 13.3 gezeigt.

Listing 13.3: *helptopic.css*

```
1:   helptopic {display: block;
2:       margin-top:3cm;
3:       margin-left:2cm;
4:       margin-right:2cm;
5:       margin-bottom:6cm;
6:       font-family:Verdana, Arial;
7:       font-size:11pt;
8:       padding:20pt;}
9:
10:  title {display: block;
11:      font-size:20pt;
12:      color:blue;
13:      font-weight:bold;
14:      text-align:center;
```

```
15:      margin-bottom:30pt;
16:      text-decoration:underline;}
17:
18:  procedure {display:block;
19:      margin-bottom:30pt}
20:
21:  step {display:block;
22:      margin-bottom:18pt}
23:
24:  action {display:block;
25:      font-weight:bold;}
26:
27:  tip {display:block;
28:      font-size:10pt;
29:      margin-left:+1cm;
30:      margin-top:12pt;
31:      color:blue;}
32:
33:  icon {display:inline;
34:      font-size:12pt;}
35:
36:  todo {display:inline;
37:      color:red;}
38:
39:  menu {display:inline;
40:      font-style:italic;}
41:
42:  xref {display:inline;
43:      text-decoration:underline;}
```

Das Ergebnis in Mozilla 5 sehen Sie in Abbildung 13.2.

Abbildung 13.3 zeigt den Quellcode.

Diese Abbildung zeigt als Quelle die eigentliche XML-Datei, keine konvertierte HTML-Version davon.

Zum Zeitpunkt der Drucklegung dieses Buches gab es noch nicht viele Informationen über die Arbeit mit CSS und XML in Mozilla 5. Beispielsweise ist noch nicht sicher, welche Funktionen von CSS Level 2 implementiert werden.

Aktuelle Informationen über die CSS-Unterstützung in Mozilla finden Sie unter der Adresse `http://www.mozilla.org/rdf/doc/xml.html`.

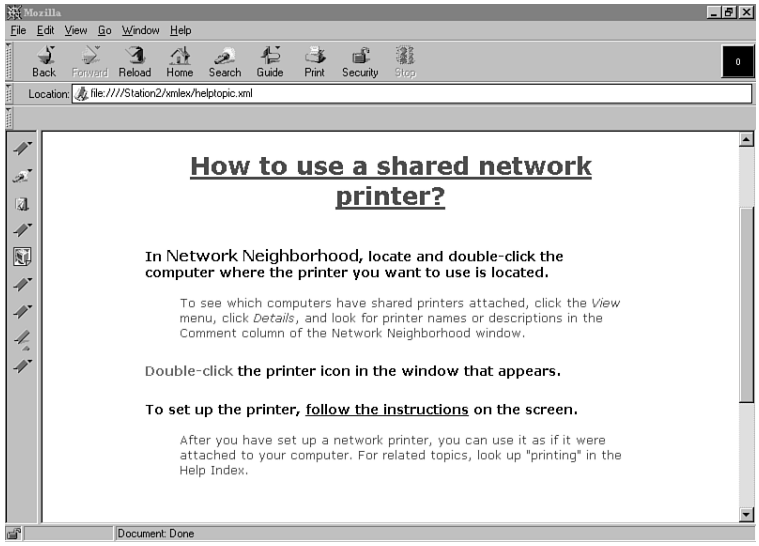


Abbildung 13.2: Die XML-Datei unter Verwendung von CSS in Mozilla 5.

Es gibt zwar noch keine Dokumentation und in der nächsten Zukunft werden auch nicht alle Funktionen von CSS Level 2 zur Verfügung stehen, aber XML+CSS ist die Methode der Wahl für die Anzeige von XML-Dokumenten.

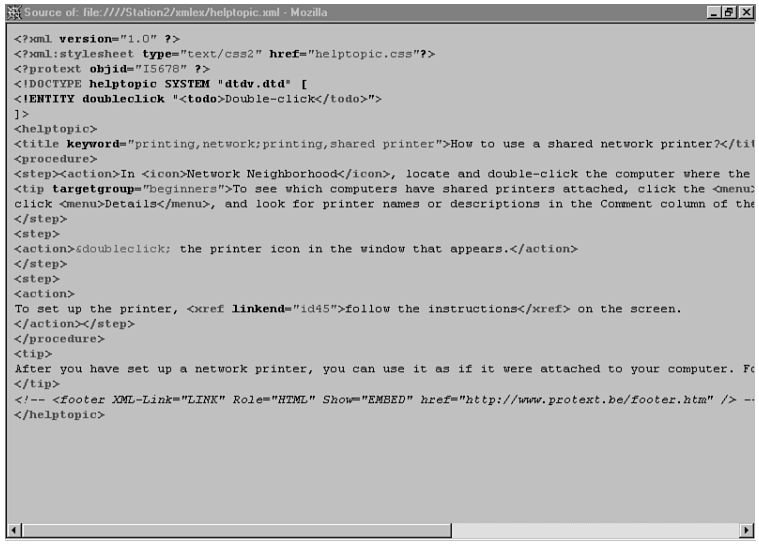


Abbildung 13.3: Die Quelldatei für die Web-Seite.

Die XML-Daten werden direkt in HTML dargestellt, ohne Zwischenumwandlung oder Übersetzungsphase.

Mozilla 5 unterstützt die DOM-1.0-Empfehlung vollständig.

Das folgende Beispiel zeigt JavaScript-Quellcode. Es verwendet das DOM-API, um einen Preis in US-Dollar durch einen Preis in Kanadischen Dollar zu ersetzen.

Listing 13.4: *book2.xml*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="books.css"?>
<books xmlns:html="http://www.w3.org/TR/REC-html40">
      <html:form>
           <html:h3>Convert to Canadian dollars:</html:h3>
           <html:input type="button" onclick="convert()" value="CAN"/>
      </html:form>
      <book>
            <title>Sam's Teach Yourself C++ in 21 Days, Second Edition
            </title>
            <author>Jesse Liberty
            </author>
            <description>This book teaches you the basics of
➥object-oriented programming with C++ and is completely revised to ANSI
➥standards. It can be used with any C++ compiler.
            </description>
            <ISBN>0-672-31070-8
            </ISBN>
            <pages>700
            </pages>
            <targetgroup>Beginning - Intermediate
            </targetgroup>
            <price unit="USA">29.99
            </price>
    </book>
    <book>
            <title>Maximum Java 1.1
            </title>
            <author>Glenn Vanderburg et al.
            </author>
            <description>Written by JAVA experts, this book explores
➥the JAVA 1.1 language, tools, and core JAVA API without reviewing fundamentals
➥or basic techniques.
            </description>
            <ISBN>1-57521-290-0
            </ISBN>
            <pages>900
            </pages>
            <targetgroup>Expert
```

```
36:             </targetgroup>
37:             <price unit="USA">49.99
38:             </price>
39:       </book>
40:       <book>
41:             <title>JavaScript Unleashed, Second Edition
42:             </title>
43:             <author>Richard Wagner et al.
44:             </author>
45:             <description>This book helps you thoroughly understand and apply
➥JavaScript.
46:             </description>
47:             <ISBN>1-57521-306-0
48:             </ISBN>
49:             <pages>1000
50:             </pages>
51:             <targetgroup>Casual - Experienced
52:             </targetgroup>
53:             <price>49.99
54:             </price>
55:       </book>
56:       <book>
57:             <title>Sam's Teach Yourself Visual C++ 5 in 21 Days,
➥Fourth Edition
58:             </title>
59:             <author>Nathan and Ori Gurewich
60:             </author>
61:             <description>This book merges the power of the best-selling
➥"Teach Yourself" series with the knowledge of Nathan and Ori Gurewich, renowned
➥experts in code, creating the most efficient way to learn Visual C++.
62:             </description>
63:             <ISBN>0-672-31014-7
64:             </ISBN>
65:             <pages>832
66:             </pages>
67:             <targetgroup>New - Casual
68:             </targetgroup>
69:             <price>35.00
70:             </price>
71:       </book>
72:       <html:script src="convert.js" />
73:    </books>
```

Zeile 2 enthält die Verarbeitungsanweisung, das in der Datei `books.css` definierte Stylesheet zu verwenden.

In Zeile 3 (das Element `books`) wird der `html`-Namensraum deklariert, weil innerhalb der Datei HTML-Elemente verwendet werden.

In den Zeilen 4 bis 7 finden Sie die HTML-Elemente `form`, `h3` und `input` und das Element `script` in Zeile 72.

Durch die Deklaration dieses Namensraums erkennt der Browser, daß das Element `h3` aus dem Namenraum `html` die für diesen Namensraum definierte Semantik erhält (Formatierung und Verhalten).

Das Element `input` (Zeile 6) bezieht sich auf eine JavaScript-Funktion, die in der Datei `convert.js` definiert wurde, auf die in Zeile 72 verweisen wird und die in Listing 13.5 aufgelistet ist.

Listing 13.5: *convert.js*

```
1:  function convert() {
2:  var priceElements = document.getElementsByTagName("price");
3:  //gibt eine NodeList aller Elemente mit dem Tag price zurück
4:  var i;
5:  for (i=0;i < priceElements.length;i++) {
➥// Durchlaufen aller price-Elemente
6:       var USAprice =
➥parseFloat(priceElements.item(i).firstChild.nodeValue);
7:       //die firstChild-Eigenschaft gibt den Kindknoten dieses
8:       //Textknotens zurück; wir nehmen den nodeValue, den eigentlichen
9:       //Text, und wandeln ihn in eine Zahl um (Floating)
10:      priceElements.item(i).setAttribute("unit","Canadian dollar");
11:      //wir setzen das Attribut "unit" auf den Wert "Canadian dollar"
12:      var convertedprice = USAprice * 1.4;
13:      //wir wandeln den Preis in Kanadische Dollar um
14:      var newprice = document.createTextNode(convertedprice);
15:      //wir legen einen neuen Textknoten mit dem neuen Preis an.
16:      priceElements.item(i).replaceChild(newprice,priceElements.
➥item(i).firstChild);
17:      //wir ersetzen den alten Textknoten durch den neuen
18:      }
19:  }
```

Als erstes erzeugen wir eine `NodeList` mit allen Elementknoten mit dem Namen `price` (Zeile 2).

Wir durchlaufen alle diese Elementknoten (Zeile 5).

Für jedes `price`-Element werten wir zunächst die `firstChild`-Eigenschaft aus, die einen Knoten vom Typ `Text` zurückgibt. Mit Hilfe der `nodeValue`-Eigenschaft übernehmen wir den Inhalt dieses Textknotens. Dieser String wird mit der Funktion `parseFloat()` in eine Ziffer konvertiert.

Diese Zahl wird in Zeile 12 konvertiert.

Mit diesem Wert legen wir einen neuen `TextNode` an (Zeile 14), der den alten Wert ersetzt (Zeile 16).

Mozilla unterstützt durch die XLink-Syntax Transklusionen.

Transklusionen sind andere Dokumente (oder Teile davon), die an der Stelle des Verweises erscheinen, als wären sie dort lokal vorhanden.

Mit anderen Worten, Sie verweisen auf ein anderes Dokument, und der Inhalt dieses Dokuments erscheint an der Stelle, wo der Verweis auftritt.

Hier ein Beispiel für eine Transklusion:

```
<footer xml:link="simple" role="HTML" show="embed" href="footer.htm"/>
```

Das XML-Element `footer` ist ein Link, der den Inhalt von `footer.htm` lädt und ihn an seine Position in der XML-Datei einbettet.

Der Inhalt von `footer.htm` könnte aussehen wie in Listing 13.6 gezeigt.

Listing 13.6: footer.htm

```
1:  <hr style="margin-top:2cm">
2:  <div class="footer" align="center">
3:  <small>
4:  Copyright ACME Company<BR>
5:  January 1997
6:  </small>
7:  </div>
```

Abbildung 13.4 zeigt das Ergebnis im Browser.

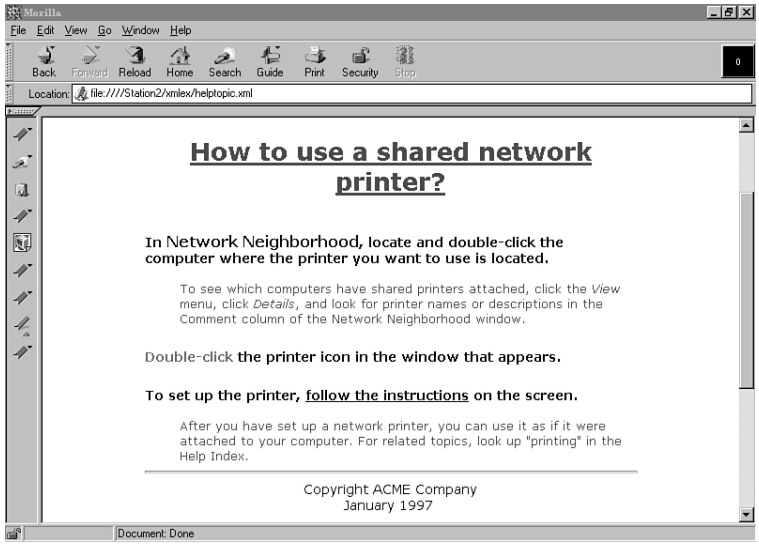


Abbildung 13.4: Transklusion in Mozilla 5.

Die Verwendung von Transklusionen ist eine ausgezeichnete Alternative zu der heutigen Praxis, Server-spezifische, Server-seitige `include`-Anweisungen zu benutzen.

Microsoft scheint ebenfalls an Transklusionen zu arbeiten, aber es ist noch nicht sicher, ob ihre Version mit anderen Browsern kompatibel sein wird.

Anzeige von XML in DocZilla

Das finnische Unternehmen Citec Information Technologies verwendet den Mozilla-5-Code zur Entwicklung eines professionellen Browsers. Dabei soll es sich um einen funktionalen XML-, SGML- und HTML-Browser mit den folgenden Funktionen handeln:

- Die Möglichkeit, Dokumente mit CSS anzuzeigen
- Vollständige Unterstützung von DOM und ECMAscript
- Dynamisches Anlegen von TOCs
- Ein strukturierter Index- und Suchmechanismus

Weitere Informationen über den Citec-Browser finden Sie unter `http://www.citec.fi/multidoczilla`.

Anzeige von XML mit Browsern, die auf der Viewport-Engine von Inso basieren

Jahrelang war die Viewport-Engine von Inso/Synex der Marktführer für das Browsing von reinen SGML-Dateien.

Diese Engine wurde unter anderem in den folgenden Produkten eingesetzt:

- Panorama (Interleaf, zuvor SoftQuad)
- SplitVision (Sörman Information)
- Multidoc Pro (Citec Information Technology)

  Die neueste Version dieser Engine ist außerdem in der Lage, XML-Dateien darzustellen.

Funktionen der Viewport-Engine

Erwähnenswerte Funktionen der Viewport-Umgbung beinhaltet die Verwendung von Stylesheets, Navigatoren, Webs und andere Hypermedia-Unterstützung.

Für jedes Dokument können ein oder mehrere *Stylesheets* definiert werden. Dazu benötigt man einen grafischen Stylesheet-Editor.

Leider ist die Sprache zur Definition der Style-Spezifkation Viewport-spezifisch. In folgenden Releases wird XSL aber das eigene Stylesheet-Format vervollständigen oder überarbeiten.

Ein *Navigator* dient als aktive und dynamische tabellarische Inhaltsansicht.

Jedes Dokumentelement kann extrahiert und in einem Navigator wieder angelegt werden. Die Elemente behalten ihre ursprüngliche Reihenfolge und Hierarchie wie im Dokument bei. Dadurch ist es möglich, ganz leicht ein Inhaltsverzeichnis oder Tabellen mit einem Überblick über die Abbildungen, Tabellen usw. anzulegen.

Ein *Web* ist eine unabhängige Datei, die Anker und Links beinhaltet.

Anker können Textabschnitten oder einen Bereich einer Grafik zugeordnet werden. Sie können Anmerkungen enthalten und als Lesezeichen dienen.

Links verbinden zwei Anker.

Webs können aktiviert und deaktiviert werden. Wird ein Web aktiviert, werden geöffneten Dokumenten Anker zugeordnet und als anklickbare Symbole angezeigt. Das ist ein sehr mächtiger Mechanismus. In einem unserer aktuellen Projekte stellen wir dasselbe Dokument in zwei Ansichten bereit: eine aktuelle Ansicht und eine Änderungsansicht. In der aktuellen Ansicht sieht der Benutzer die Information so, wie sie jetzt ist. In der Änderungsansicht wird ein weiteres Stylesheet geöffnet, das neue Informationen rot, geänderte Informationen blau mit Änderungsmarkierungen und gelöschte Informationen durchgestrichen darstellt. Diese Änderungsinformation wird im Dokument in einem Revisionsattribut abgelegt. Darüber hinaus wird in der Änderungsansicht ein Web geöffnet, das eine Liste von Ankern enthält, die auf alle neuen und geänderten Themen verweisen. Damit ist es dem Benutzer möglich, die Änderungen sehr schnell zu erkennen. Außerdem werden die Gründe für die Änderungen in den zugehörigen Anmerkungen erklärt.

Dasselbe Konzept, Link-Informationen außerhalb des Dokuments abzulegen, ist auch Teil des XLink-Vorschlags.

Und so funktioniert es

Neben Ihrer XML-Datei brauchen Sie zusätzliche Stylesheets und Navigatoren. In beidem wird ein spezielles Markup verwendet, das in einer SGML-DTD definiert ist. Um ein Stylesheet anzulegen, brauchen Sie eine Datei mit der Dateinamenerweiterung `.ssh`. Der Inhalt dieser Datei sollte aussehen wie in Listing 13.7 gezeigt.

Listing 13.7: Eine Viewport-Stylesheet-Definition

```
1:  <!doctype stylesheet PUBLIC "-//Synex Information AB//DTD stylesheet
➥V2.00//EN>
2:  <stylesheet>
3:  </stylesheet>
```

Der weitere Inhalt der Style-Spezifikationen wird automatisch bereitgestellt, wenn Sie den grafischen Stylesheet-Editor benutzen.

Um einen Navigator zu erzeugen, legen Sie eine Datei mit der Dateinamenerweiterung `.nav` an. Der Inhalt dieser Datei sollte aussehen wie in Listing 13.8 gezeigt.

Listing 13.8: Eine Viewport-Navigator-Definition

```
1:  <!doctype toc-def PUBLIC "-//SYNEX Information AB//DTD Navigator v2.00//EN">
2:  <toc-def>
3:  </toc-def>
```

Der weitere Inhalt dieser Datei wird automatisch bereitgestellt, wenn Sie den Browser zur Definition von Navigatoren verwenden.

Nachdem wir unser Stylesheet und den Navigator definiert haben, müssen wir eine Beziehung zwischen unserer XML-Datei und den unterstützenden Dateien einrichten.

Das kann auf zweierlei Arten erfolgen:

- Sie können dem eigentlichen Dokument zugeordnet werden.
- Wurde eine DTD mit einem öffentlichen Bezeichner verwendet, können sie dieser DTD zugeordnet werden.

Für eine Beziehung zu dem eigentlichen Dokument müssen Sie für die Style-Spezifikation die folgende Verarbeitungsanweisung in das Dokument einfügen:

```
<?stylespec "menüname" "dateiname"?>
```

Dabei ist

- `menüname` der Name, der im Anzeige-Menü des Browsers verwendet wird.
- `dateiname` der Name der Stylesheet-Datei (*.ssh).

Und für den Navigator fügen Sie ein:

```
<?navigator "menüname" "dateiname"?>
```

Dabei ist

- `menüname` der Name, der im Navigator-Menü des Browsers verwendet wird.
- `dateiname` der Name der Navigatordatei (*.nav).

Hat die DTD einen öffentlichen Bezeichner, müssen Sie die Datei ENTITYRSC-Datei in der Viewport-Umgebung bearbeiten. In dieser ENTITYRSC-Datei können Sie eine Beziehung zwischen Stylesheets und Navigatoren auf der einen Seite und öffentlichen Bezeichnern auf der anderen Seite einrichten, wie in Listing 13.9 verdeutlicht.

Listing 13.9: Definition der Beziehungen zwischen einer DTD und Viewport-Stylesheets und Navigatoren

```
1:  PUBLIC "-//Pro Text//DTD online help//EN"
2:       STYLESPEC "standard"     "standard.ssh"
3:       STYLESPEC "large"     "large.ssh"
4:       NAVIGATOR "topics"      "topics.nav"
5:       NAVIGATOR "figures"        "figures.nav"
```

Jetzt betrachten wir ein Beispiel, beginnend mit der XML-Datei, die Sie in Listing 13.10 sehen.

Listing 13.10: *musicians.xml*

```
1:   <?xml version="1.0"?>
2:   <!DOCTYPE musicians [
3:   <!ELEMENT musicians  (musician)+ >
4:   <!ELEMENT musician  (name, instrument, NrOfRecordings)>
5:   <!ELEMENT (name, instrument, NrOfRecordings)  (#PCDATA)>
6:   ]>
7:   <musicians>
8:     <musician>
9:   <name>Joey Baron
10:  </name>
11:  <instrument>drums
12:  </instrument>
13:  <NrOfRecordings>1
14:  </NrOfRecordings>
15:    </musician>
16:    <musician>
17:  <name>Bill Frisell
18:  </name>
19:  <instrument>guitar
20:  </instrument>
21:  <NrOfRecordings>3
22:  </NrOfRecordings>
23:    </musician>
24:    <musician>
25:  <name>Don Byron
26:  </name>
27:  <instrument>clarinet
28:  </instrument>
29:  <NrOfRecordings>2
30:  </NrOfRecordings>
31:    </musician>
```

```
32:    <musician>
33:  <name>Dave Douglas
34:  </name>
35:  <instrument>trumpet
36:  </instrument>
37:  <NrOfRecordings>1
38:  </NrOfRecordings>
39:    </musician>
40:  </musicians>
```

Jetzt fügen wir zwei Stylesheets und einen Navigator ein.

Die Dateien `musicians.ssh` und `musician2.ssh` enthalten folgendes:

```
<!doctype stylesheet PUBLIC "-//Synex Information AB//DTD stylesheet V2.00//EN>
  <stylesheet>
  </stylesheet>
```

Die Datei `musician.nav` enthält das folgende:

```
<!doctype toc-def PUBLIC "-//SYNEX Information AB//DTD Navigator v2.00//EN">
  <toc-def>
  </toc-def>
```

Jetzt verbinden wir diese Dateien, indem wir Verarbeitungsanweisungen in unsere XML-Datei aufnehmen, wie in Listing 13.11 gezeigt.

Listing 13.11: *musicians.xml* ***mit Verarbeitungsanweisungen***

```
1:  <?xml version="1.0"?>
2:  <!DOCTYPE musicians [
3:  <!ELEMENT musicians  (musician)+ >
4:  <!ELEMENT musician  (name, instrument, NrOfRecordings)>
5:  <!ELEMENT name  (#PCDATA)>
6:  <!ELEMENT instrument  (#PCDATA)>
7:  <!ELEMENT NrOfRecordings (#PCDATA)>
8:  ]>
9:  <?stylespec "table" "musicians.ssh"?>
10:  <?stylespec "indent" "musician2.ssh"?>
11:  <?navigator "toc" "musicians.nav"?>
12:  <musicians>
13:  ...
14:  </musicians>
```

Jetzt werden dem View-Menü und dem Navigator-Menü `table` und `indent` hinzugefügt.

Öffnen Sie die XML-Datei im Synex-Viewport-basierten Browser, wie in Abbildung 13.5 gezeigt.

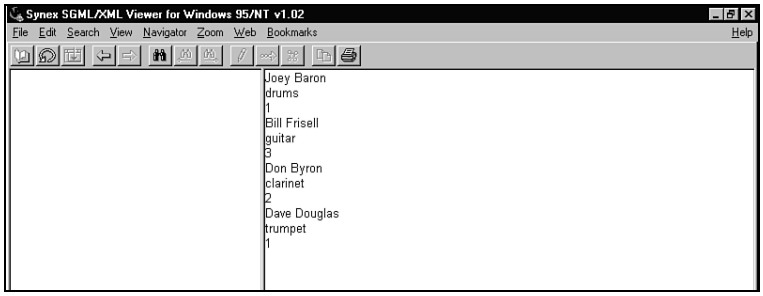


Abbildung 13.5: Die XML-Datei, wie sie zum ersten Mal geöffnet wird.

Wir sehen hier noch nicht viel, aber das kommt daher, daß wir noch keine Informationen in unsere Stylesheets und Navigatoren eingetragen haben.

Wir ändern unsere Ansicht (mit Hilfe des View-Menüs), um Tags und die Baumstruktur unseres Dokuments anzuzeigen, wie in Abbildung 13.6 gezeigt.

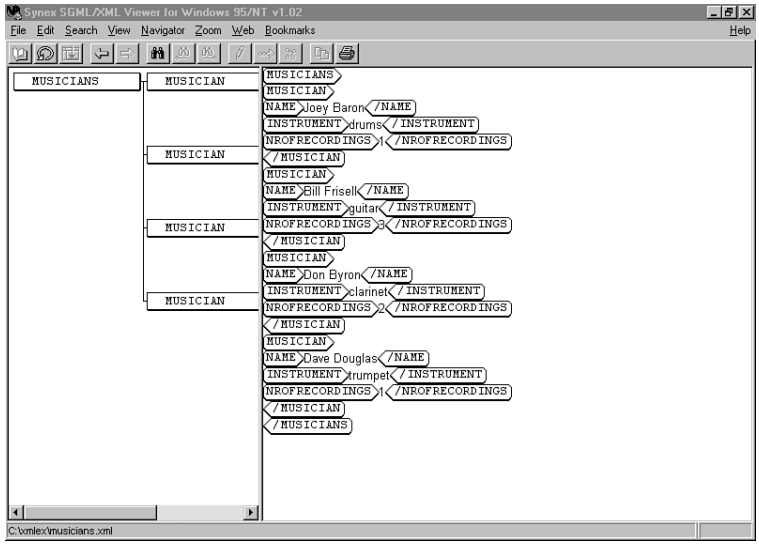


Abbildung 13.6: Tagnamen und Baumstruktur werden angezeigt.

Durch Anklicken eines Tags oder eines Baumknotens mit der rechten Maustaste erhalten Sie die Möglichkeit, den Stylesheet-Editor zu öffnen, wie in Abbildung 13.7 gezeigt.

Abbildung 13.8 zeigt ein mögliches Ergebnis unserer Stylesheet-Bearbeitung.

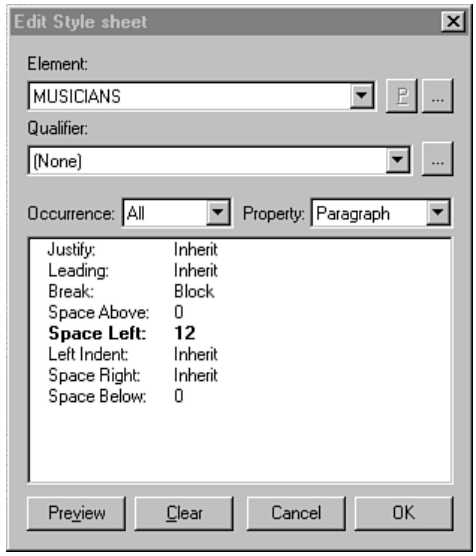


Abbildung 13.7: Der geöffnete Stylesheet-Editor.

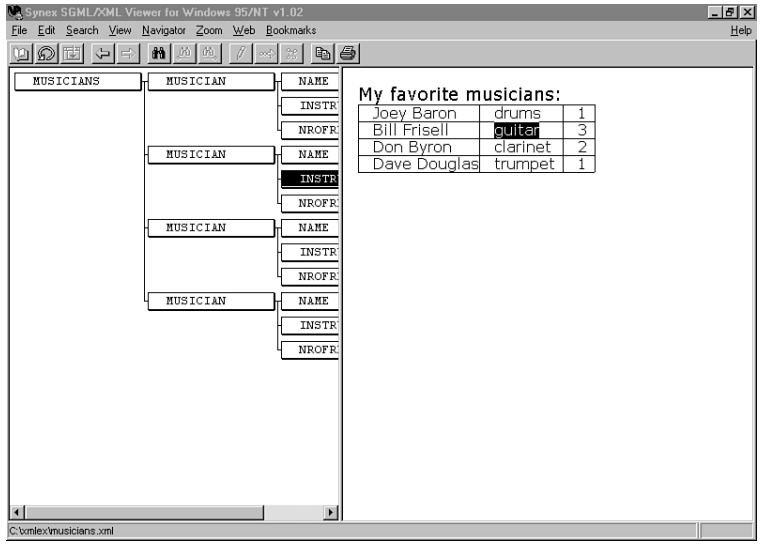


Abbildung 13.8: Ein fertiges Stylesheet nach seiner Anwendung.

Als nächstes definieren wir einen Navigator. In dieser Datei wäre es sinnvoll, einen Überblick über die Namen der Musiker zu haben.

Um den Navigator-Editor zu öffnen, klicken Sie mit der rechten Maustaste auf einen Elementnamen, den Sie in Ihren Navigator aufnehmen möchten, wie in Abbildung 13.9 gezeigt.

Klicken mit der Maustaste zeigt die in Abbildung 13.10 enthaltenen Ergebnisse.

Abbildung 13.9: Der Navigator-Editor.

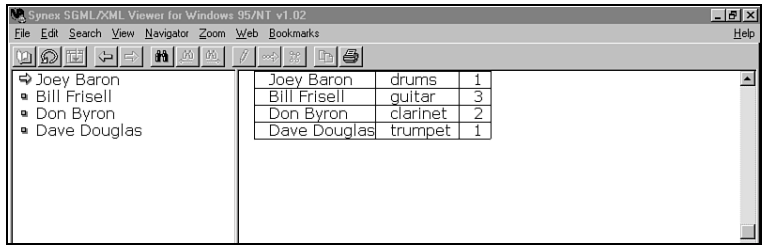


Abbildung 13.10: Der definierte Navigator.

Sie sehen dieselbe Datei, aber mit einem anderen Stylesheet, wie in Abbildung 13.11 gezeigt.

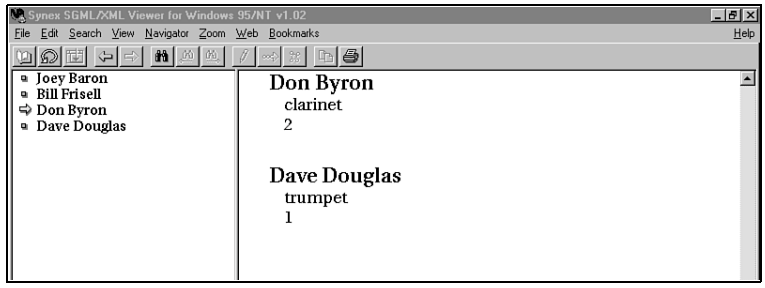


Abbildung 13.11: Hier wurde ein anderes Stylesheet zugeordnet.

Das in diesem letzten Beispiel verwendete Stylesheet sehen Sie in Listing 13.12.

Listing 13.12: Beispiel für ein Stylesheet

```
1:   <!DOCTYPE STYLESHEET PUBLIC "-//SYNEX INFORMATION AB//DTD STYLESHEET
➥V2.00//EN"[
2:   <!ELEMENT sheet (margins?)>
3:   <!ELEMENT margins EMPTY>
4:   <!ATTLIST margins
5:        left CDATA #IMPLIED
6:        right CDATA #IMPLIED
7:        top CDATA #IMPLIED
8:        bottom CDATA #IMPLIED>
9:
10:  <!ELEMENT font-shadow EMPTY>
11:  <!ATTLIST font-shadow
12:       v NUMBER #REQUIRED>
13:
```

```
14:  <!ELEMENT shadow-color EMPTY>
15:  <!ATTLIST shadow-color
16:      v CDATA #REQUIRED>
17:
18:  ]>
19:
20:  <STYLESHEET>
21:
22:  <STYLE TAG="NROFRECORDINGS">
23:  <SPC-LEFT V="36">
24:  <BREAK-BEFORE>
25:  <BREAK-AFTER>
26:  </STYLE>
27:
28:  <STYLE TAG="MUSICIANS">
29:  <FONT-FAMILY V="Utopia">
30:  <FONT-SIZE V=14>
31:  <SPC-ABOVE V="12">
32:  <SPC-LEFT V="24">
33:  <SPC-BELOW V="12">
34:  <BREAK-BEFORE>
35:  <BREAK-AFTER>
36:  </STYLE>
37:
38:  <STYLE TAG="NAME">
39:  <FONT-WEIGHT V=Bold>
40:  <FONT-SCALE V="120">
41:  <SPC-ABOVE V="24">
42:  <BREAK-BEFORE>
43:  <BREAK-AFTER>
44:  </STYLE>
45:
46:  <STYLE TAG="INSTRUMENT">
47:  <SPC-LEFT V="36">
48:  <BREAK-BEFORE>
49:  <BREAK-AFTER>
50:  </STYLE>
51:
52:  </STYLESHEET>
```

Den verwendeten Navigator sehen Sie in Listing 13.13.

Listing 13.13: Beispiel für einen Navigator

```
1:  <!DOCTYPE TOC-DEF PUBLIC "-//SYNEX INFORMATION AB//DTD NAVIGATOR V2.00//EN">
2:
3:  <TOC-DEF SCALE=70>
4:  <TOC BODY="MUSICIAN" TITLE="NAME">
5:  </TOC-DEF>
```

Die Synex-Viewport-Engine ist der aktuelle Stand der Technik für die Anzeige von reinen SGML-Dateien. Daß sie auch die direkte Anzeige von XML-Dokumenten unterstützt, kann nur begrüßt werden.

Es gibt aber auch einige Nachteile:

- Der Stylesheet-Mechanismus ist proprietär.
- Der Browser ist nicht kostenlos.
- Das Produkt ist nur innerhalb der SGML-Gemeinde bekannt, nicht in der sehr viel größeren HTML-Gemeinde.

Zusammenfassung

In diesem Kapitel haben wir andere Browser vorgestellt, mit denen Sie XML anzeigen/darstellen können. Mozilla 5 befindet sich zwar immer noch in der Entwicklung, zeigt aber bereits vielversprechende Ansätze für die direkte Anzeige von XML-Dateien mit zugehörigen CSS-Stylesheets. Netscape erwartet die Veröffentlichung im ersten Quartal 2000.

Darüber hinaus versucht er, die Web-Standards und Empfehlungen konsistent zu implementieren. Allein das ist Grund zur Freude.

DocZilla und Synex Viewport wurden ebenfalls angesprochen.

F&A

F Ich habe Gecko heruntergeladen, und er scheint nicht zu funktionieren. Was ist passiert?

A *Die Gecko-Entwicklung ist noch nicht abgeschlossen. Täglich erscheint ein neuer Build. Wenn der Build irgendeines fernen Tages stabil genug sein wird, ist es sehr wahrscheinlich, daß schon der nächste besser sein wird.*

F Ich sehe in meinem Synex-Browser keinen formatierten Text.

A *Prüfen Sie, ob Ihrer XML-Datei auf irgendeine Weise ein Stylesheet zugeordnet wurde.*

Übungen

Verwenden Sie Listing 13.14 für diese Übung.

- Legen Sie eine CSS-Datei (*Cascading Style Sheet*) an, und ordnen Sie es der XML-Datei zu.
- Öffnen Sie die XML-Datei mit Mozilla/Gecko.

Listing 13.14: *books.xml*

```
1:  <?xml version="1.0"?>
2:  <books>
3:      <book>
4:          <title>Sam's Teach Yourself C++ in 21 Days, Second Edition
5:          </title>
6:          <author>Jesse Liberty
7:          </author>
8:          <description>This book teaches you the basics of
➥object-oriented programming with C++ and is completely revised
➥to ANSI standards. It can be used with any C++ compiler.
9:          </description>
10:          <ISBN>0-672-31070-8
11:          </ISBN>
12:          <pages>700
13:          </pages>
14:          <targetgroup>Beginning - Intermediate
15:          </targetgroup>
16:          <price unit="USA">29.99
17:          </price>
18:      </book>
19:      <book>
20:          <title>Maximum Java 1.1
21:          </title>
22:          <author>Glenn Vanderburg et al.
23:          </author>
24:         <description>Written by JAVA experts, this book
➥explores the JAVA 1.1 language, tools, and core JAVA API without
➥reviewing fundamentals or basic techniques.
```

```
25:          </description>
26:          <ISBN>1-57521-290-0
27:          </ISBN>
28:          <pages>900
29:          </pages>
30:          <targetgroup>Expert
31:          </targetgroup>
32:          <price unit="USA">49.99
33:          </price>
34:      </book>
35:      <book>
36:          <title>JavaScript Unleashed, Second Edition
37:          </title>
38:          <author>Richard Wagner et al.
39:          </author>
40:          <description>This book helps you thoroughly understand and
➥apply JavaScript.
41:          </description>
42:          <ISBN>1-57521-306-0
43:          </ISBN>
44:          <pages>1000
45:          </pages>
46:          <targetgroup>Casual - Experienced
47:          </targetgroup>
48:          <price>49.99
49:          </price>
50:      </book>
51:      <book>
52:          <title>Sam's Teach Yourself Visual C++ 5 in 21 Days, Fourth Edition
53:          </title>
54:          <author>Nathan and Ori Gurewich
55:          </author>
56:          <description>This book merges the power of the best-selling
➥"Teach Yourself" series with the knowledge of Nathan and Ori Gurewich,
➥renowned experts in code, creating the most efficient way to learn Visual C++.
57:          </description>
58:          <ISBN>0-672-31014-7
59:          </ISBN>
60:          <pages>832
61:          </pages>
62:          <targetgroup>New - Casual
63:          </targetgroup>
64:          <price>35.00
65:          </price>
66:      </book>
67:  </books>
```

XML-Verarbeitung

Woche 2

Heute lernen Sie, warum und wie XML-Daten weiterverarbeitet werden. Darüber hinaus geht es um zwei wichtige Programmierparadigmen, die in der XML-Verarbeitung verwendet werden:

- Den ereignisgesteuerten Ansatz
- Den baumbasierten Ansatz

Gründe für die Verarbeitung von XML

Wenn Sie XML einsetzen, haben Sie bereits eine Entscheidung getroffen, Ihre Daten auf die bestmögliche Weise mit Markup zu kennzeichnen. Sie haben auch gesehen, daß einige Browser Ihre XML-Dateien direkt anzeigen können. Warum sollten Sie aber dann Programme schreiben, die Ihr XML verarbeiten?

Der Antwort sind nur durch Ihre eigene Vorstellungskraft Grenzen gesetzt, wir werden aber hier einige mögliche Begründungen aufführen:

- Weitergabe an unterschiedliche Medien
- Weitergabe an unterschiedliche Zielgruppen
- Hinzufügen, Entfernen und Umstrukturierung von Informationen
- Datenbanken laden
- Berichtserstellung

Weitergabe an unterschiedliche Medien

Informationen werden über unterschiedliche Medien veröffentlicht: im Web, auf Papier, auf CD-ROM, als Hilfedateien usw.

Für eine Veröffentlichung im Web wird uns HTML noch eine Weile erhalten bleiben. Aber welcher HTML-Typ? Version 3.2, Version 4, DHTML, die Microsoft-Version oder die DHTML-Netscape-Version?

Bisher hatten Sie nur die folgenden Auswahlmöglichkeiten:

- Sie verwenden nur die HTML-Tags, die von allen Browsern unterstützt werden (den Ansatz des größten gemeinsamen Nenners).
- Sie legen HTML-Seiten an, die für bestimmte Browser optimiert sind.
- Sie nehmen Skripten in Ihre HTML-Seiten auf, um Browser-spezifischen Code zu erzeugen.

Mit XML können Sie eine andere Taktik verfolgen. Sie können prüfen, welcher Browser Informationen anfordert, und Ihre XML-Daten dynamisch in die am besten geeignete HTML-Version umwandeln.

Der große Vorteil bei diesem Ansatz ist, daß Sie nur eine einzige Quelldatei verwalten müssen. Die gesamte Formatierung und Verarbeitung bleibt von Ihren Daten getrennt. Das bedeutet, wenn es eine neue HTML-Version gibt, müssen Sie zwar eine neue Konvertierung vornehmen, aber Ihre Daten müssen dafür nicht verändert werden.

Um einen Ausdruck zu erhalten, können Sie Ihre XML-Daten in RTF (*Rich Text Format*) umwandeln, das in Microsoft Word geladen werden kann, oder auch in eine andere Markup-Sprache, die in anderen Textverarbeitungen genutzt wird (beispielsweise MIF für FrameMaker).

Die XML-Datei in Listing 14.1 kann als Quellcode für die RTF-Ausgabe genutzt werden.

Listing 14.1: *musicians.xml* ***– die XML-Datei, die konvertiert werden soll***

```
1: <?xml version="1.0"?>
 2: <!DOCTYPE musicians [
 3: <!ELEMENT musicians  (musician)+ >
 4: <!ELEMENT musician  (name, instrument, NrOfRecordings)>
 5: <!ELEMENT (name, instrument, NrOfRecordings)  (#PCDATA)>
 6: ]>
 7: <musicians>
 8:   <musician>
 9: <name>Joey Baron
10: </name>
11: <instrument>drums
12: </instrument>
13: <NrOfRecordings>1
14: </NrOfRecordings>
15:   </musician>
16:   <musician>
17: <name>Bill Frisell
18: </name>
19: <instrument>guitar
20: </instrument>
21: <NrOfRecordings>3
```

```
22: </NrOfRecordings>
23:   </musician>
24:   <musician>
25: <name>Don Byron
26: </name>
27: <instrument>clarinet
28: </instrument>
29: <NrOfRecordings>2
30: </NrOfRecordings>
31:   </musician>
32:   <musician>
33: <name>Dave Douglas
34: </name>
35: <instrument>trumpet
36: </instrument>
37: <NrOfRecordings>1
38: </NrOfRecordings>
39:   </musician>
40: </musicians>
```

Daraus könnten wir die RTF-Datei in Listing 14.2 machen.

Das ist nur ein Beispiel dafür, wie XML in eine andere Markup-Sprache umgewandelt werden kann. Wie das genau geht, wird in den nächsten Kapiteln beschrieben.

Listing 14.2: *musicians.rtf* ***– die RTF-Datei nach der Konvertierung***

```
1: {\rtf1\ansi\ansicpg1252\uc1
 2:
 3: ... //lots of rtf code deleted
 4:
 5: \pard\plain \s18\li1440\sb320\widctlpar
 6: \adjustright \b\f15\fs28\lang2057\cgrid {Joey Baron
 7: \par }\pard\plain \s19\li2880\sb120\widctlpar
 8: \adjustright \f15\fs20\lang2057\cgrid {drums
 9: \par }\pard\plain \s20\li3600\sb120\sa400\widctlpar
10: \adjustright \i\f15\fs20\lang2057\cgrid {1
11: \par }\pard\plain \s18\li1440\sb320\widctlpar
12: \adjustright \b\f15\fs28\lang2057\cgrid {Bill Frisell
13: \par }\pard\plain \s19\li2880\sb120\widctlpar
```

```
14: \adjustright \f15\fs20\lang2057\cgrid {guitar
15: \par }\pard\plain \s20\li3600\sb120\sa400\widctlpar
16: \adjustright \i\f15\fs20\lang2057\cgrid {3
17: \par }\pard\plain \s18\li1440\sb320\widctlpar
18: \adjustright \b\f15\fs28\lang2057\cgrid {Don Byron
19: \par }\pard\plain \s19\li2880\sb120\widctlpar
20: \adjustright \f15\fs20\lang2057\cgrid {clarinet
21: \par }\pard\plain \s20\li3600\sb120\sa400\widctlpar
22: \adjustright \i\f15\fs20\lang2057\cgrid {2
23: \par }\pard\plain \s18\li1440\sb320\widctlpar
24: \adjustright \b\f15\fs28\lang2057\cgrid {Dave Douglas
25: \par }\pard\plain \s19\li2880\sb120\widctlpar
26: \adjustright \f15\fs20\lang2057\cgrid {trumpet
27: \par }\pard\plain \s20\li3600\sb120\sa400\widctlpar
28: \adjustright \i\f15\fs20\lang2057\cgrid {1}{\f2\lang2067\cgrid0
29: \par }\pard\plain \widctlpar\adjustright \fs20\lang2057\cgrid {
30: \par }}
```

Diese Datei kann von Microsoft Word und den meisten anderen Textverarbeitungen gelesen werden.

Weitergabe an unterschiedliche Zielgruppen

Sie wollen Ihre Ausgaben nicht nur auf unterschiedlichen Medien vornehmen, sondern auch unterschiedliche Informationen an unterschiedliche Benutzergruppen weitergeben. Einige Beispiele:

- Anfänger, Fortgeschrittene, Experten
- Mitglieder, Nichtmitglieder
- Inland, Ausland

Die XML-Datei in Listing 14.3 könnte als Quellcode für spezifische Ausgaben dienen, die an Benutzer mit unterschiedlichem Vorwissen weitergegeben werden.

Listing 14.3: *delivery.xml* ***– eine XML-Datei für selektive Veröffentlichungen***

```
1: <?xml version="1.0"?>
 2: <procedure type="disassemble">
 3:     <object>
```

```
 4: <type>Mismatcher</type>
 5: <seriesnr>21568</seriesnr>
 6:        </object>
 7:     <title>How to disassemble Mismatcher 21568</title>
 8:     <steps experience="firsttime">
 9:         <step>
10:             <action>Press ....
11:             </action>
12:             <result>The door ....
13:             </result>
14:         </step>
15:         <step>
16:             <action>Push ....
17:             </action>
18:             <result>The back ...
19:             </result>
20:         </step>
21:         <step>
22:             <action>Drill ....
23:             </action>
24:             <result>Part x123 ...
25:             </result>
26:         </step>
27:     </steps>
28:     <steps experience="donebefore">
29:         <step>
30:             <action>Throw it on the ground.
31:             </action>
32:         </step>
33:     </steps>
34:     <tip>Take a bottle of ....
35:     </tip>
36: </procedure>
```

Diese Daten ermöglichen Ihnen, die Schritte abhängig von dem Vorwissen Ihrer Leser anzupassen. Handelt es sich um absolute Anfänger, kann das Element `steps` mit dem Attribut `experience` und dem Wert `donebefore` (Zeilen 28–33) völlig weggelassen werden.

Außerdem könnten Sie entscheiden, daß der `tip` (Zeilen 34 und 35) nur für Anfänger relevant ist, und ihn für die Zielgruppe `donebefore` unterdrücken.

Hinzufügen, Entfernen und Umstrukturierung von Informationen

Bei der Definition von Berichten, die aus Ihrer relationalen Datenbank erzeugt werden, können Sie die Daten nach Bedarf anordnen: Daten weglassen, neue Daten basierend auf bereits existierenden Daten berechnen (beispielsweise einen Mittelwert) usw.

Dasselbe gilt für XML-codierte Daten. Sie können Ihre Daten hinzufügen, entfernen und neu anordnen.

Betrachten Sie das Beispiel in Listing 14.4.

Listing 14.4: *books.xml* ***– eine XML-Datei für die Umstrukturierung von Ideen***

```
1: <?xml version="1.0"?>
 2: <books>
 3:     <book>
 4:         <title>Sams Teach Yourself C++ in 21 Days, Second Edition
 5:         </title>
 6:         <author>Jesse Liberty
 7:         </author>
 8:         <description>This book teaches you the basics of
➥object-oriented programming with C++ and is completely revised
➥to ANSI standards. It can be used with any C++ compiler.
9:         </description>
10:         <ISBN>0-672-31070-8
11:         </ISBN>
12:         <pages>700
13:         </pages>
14:         <targetgroup>Beginning - Intermediate
15:         </targetgroup>
16:         <price unit="USA">29.99
17:         </price>
18:     </book>
19:     <book>
20:         ...
21:     </book>
22: </books>
```

Mit Hilfe dieses Listings können Sie die folgenden Dinge vornehmen:

- Die `targetgroup`-Information weglassen
- Ein zweites `price`-Element in »Canadian dollar« einfügen
- Eine Liste mit allen Buchtiteln oder allen Autoren einfügen
- Die ISBN-Nummern hinter dem Preis angeben

Laden aus Datenbanken

Betrachten Sie noch einmal Listing 14.1. Es wäre sinnvoll, diese Information in einer Datenbank oder einer Kalkulationstabelle bereitzustellen. Und für genau diese Aufgabe wird XML von Microsoft eingesetzt: als Format für die Weitergabe und den Austausch von strukturierten Daten. Und strukturierte Daten befinden sich normalerweise in relationalen Datenbanken.

In der nahen Zukunft wird es XML-Writer und XML-Reader für jede Datenbank geben. Sie können aber auch selbst Code schreiben, um eine Datei mit durch Kommas getrennten Werten zu erzeugen, wie in Listing 14.5 gezeigt.

Listing 14.5:** `musicians.csv` **– eine Datei mit durch Kommas voneinander getrennten Werten.

```
1: Name,Instrument,NrOfRecordings
2: Joey Baron,drums,1
3: Bill Frisell,guitar,3
4: Don Byron,clarinet,2
5: Dave Douglas,trumpet,1
```

Sie können aber auch SQL-Code schreiben, wie in Listing 14.6 gezeigt.

Listing 14.6: SQL-Code

```
1: create table musicians (
 2:     name text not null primary key,
 3:     instrument text,
 4:     nrofrecordings integer
 5: );
 6: insert into musicians values ('Joey Baron','drums',1);
 7: insert into musicians values ('Bill Frisell','guitar',3);
 8: insert into musicians values ('Don Byron','clarinet',2);
 9: insert into musicians values ('Dave Douglas','trumpet',1);
```

Berichte

Durch die Weiterverarbeitung von XML-Dateien können ganz einfach Fragen wie die folgenden beantwortet werden:

- Wie viele Musiker sind in der Datei festgehalten?
- Wie viele davon spielen Gitarre?
- Wie viele Aufnahmen gibt es insgesamt?

Nachdem Sie Ihre Dokumente in echte Textdatenbanken umgeformt haben, können Sie sie auf die unterschiedlichsten Arten verarbeiten und Ihre Ausgaben nach Bedarf aufbereiten.

Drei Verarbeitungsparadigmen

XML-Dokumente können betrachtet werden als:

- Eine spezielle Art von Textdatei
- Eine Abfolge von Ereignissen
- Eine Hierarchie

Ein XML-Dokument als Textdatei

Eine XML-Datei kann als Textdatei betrachtet werden, in der Sie eine Kombination aus echten Daten und der Markup-Sprache finden. Es gibt zahlreiche kostenlose Werkzeuge, die die Textmanipulation unter Verwendung regulärer Ausdrücke erlauben: `awk`, `grep`, `perl` und `python`.

Wenn Sie wissen wollen, wie viele Musiker in Ihrer Datei `musicians.xml` verwaltet werden, schreiben Sie den folgenden `grep`-Befehl:

```
grep -c "<musician>" musicians.xml
```

Dieser Befehl setzt sich aus den folgenden Komponenten zusammen:

- `-c` ist die Option, nur die Anzahl der übereinstimmenden Zeilen auszugeben.
- In den Anführungszeichen wird der Text angegeben, für den eine Übereinstimmung gesucht wird, in diesem Fall das Start-Tag für das Element `musician`.
- Die zu durchsuchende Datei, beispielsweise `musicians.xml`.

Dieses Beispiel ergibt 4.

Ein XML-Dokument als Abfolge von Ereignissen

Heute weiß jeder, wie man sich auf einer grafischen Benutzeroberfläche wie Windows, Mac OS oder Motif bewegt. In diesen Umgebungen fängt das Betriebssystem alle Ereignisse auf: Mausbewegungen und -klicks, Tastatureingaben usw. Anschließend sendet das Betriebssystem Nachrichten an das Programm, die diesem mitteilen, welche Ereignisse aufgetreten sind. Der Programmierer ist dafür verantwortlich, wie der Code diese Ereignisse verarbeitet.

Man erkennt eine Analogie zu XML-Dateien. Wenn Sie eine XML-Datei sequentiell lesen, treten alle möglichen Ereignisse auf. Sie gelangen an den Dokumentanfang, lesen Start- und Ende-Tags, erkennen Kommentare und Verarbeitungsanweisungen usw. Dabei handelt es sich um Datenereignisse.

Der Programmierer ist dafür verantwortlich, im Code auf diese Ereignisse zu reagieren. Diese Reaktionen werden auch als *Ereignis-Handler* bezeichnet.

Ein *Ereignis-Handler* ist der Code, der ausgeführt wird, wenn ein Ereignis aufgetreten ist.

Für die Datei `musicians.xml` (Listing 14.1) erzeugt ein ereignisorientierter Prozessor die in Listing 14.7 aufgelisteten Ereignisse.

Listing 14.7: Ereignisse, die von der Datei `musicians.xml` erzeugt wurden

```
1: Start document
 2: Start element: musicians
 3: Start element: musician
 4: Start element: name
 5: Characters: Joey Baron
 6: End element: name
 7: Start element: instrument
 8: Characters: drums
 9: End element: instrument
10: Start element: NrOfRecordings
11: Characters: 1
12: End element: NrOfRecordings
13: End element: musician
14: ...
15: End element: musicians
16: End document
```

Beachten Sie, daß ein XML-Dokument in hierarchischer Reihenfolge sequentiell von links nach rechts durchlaufen wird.

Abbildung 14.1 zeigt die XML-Struktur.

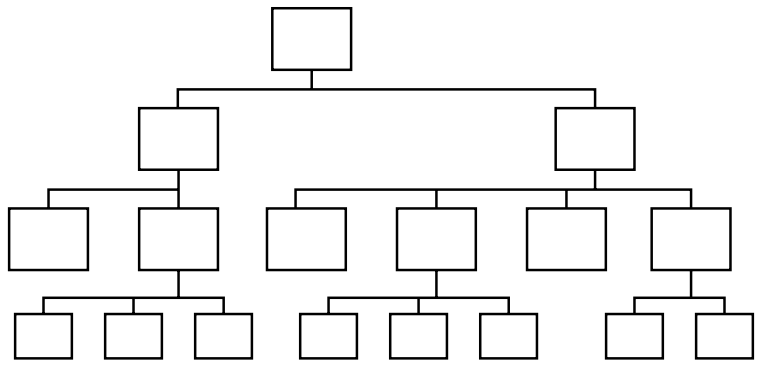


Abbildung 14.1: Der XML-Baum.

Abbildung 14.2 zeigt die Reihenfolge, in der die einzelnen Knoten des Baums durchlaufen werden.

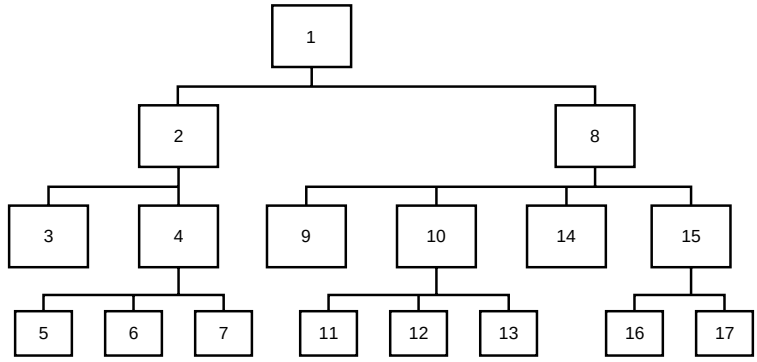


Abbildung 14.2: Die Reihenfolge, in der die Knoten durchlaufen werden.

Beim Durchlaufen legen die meisten ereignisbasierten Prozessoren Informationen über bereits durchlaufene Knoten ab.

Im ereignisgesteuerten Modus verwaltet Balise von AIS die Elementtypen und Attribute für übergeordnete Knoten bis zur Wurzel und für den ersten linken Geschwisterknoten dieser übergeordneten Knoten, wie in Abbildung 14.3 gezeigt.

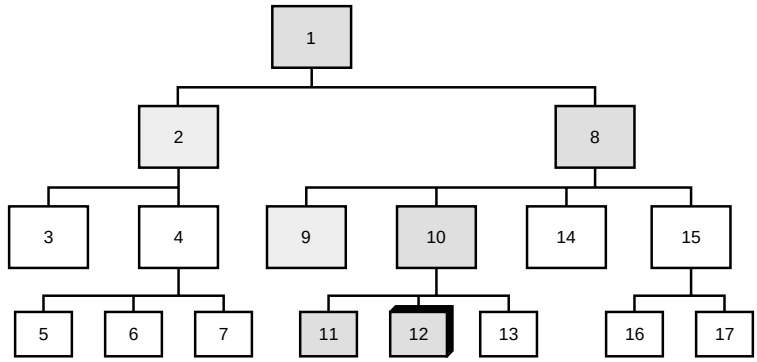


Abbildung 14.3: Die verwalteten Kontextinformationen.

Abbildung 14.3 zeigt, welche anderen Knoten von Knoten Nummer 12 aus verwaltet werden.

Welche Kontextinformation verwaltet wird, kann sich zwischen den Produkten unterscheiden. Sie können jedoch davon ausgehen, daß mindestens die Informationen des vorhergehenden Geschwisterknotens mit demselben Elternelement und alle Vorgängerknoten aufgezeichnet werden.

Dank dieser Katalogisierung (wenn auch nur in begrenztem Umfang) können Sie Handler für diesen Kontext schreiben:

- Start-Element: `para`, erstes nach `title`.
- Auf diese Weise können Sie eine andere Reaktion definieren, wenn Ihr `para`-Element als erstes nach `title` erscheint.
- Start-Element: `li`, mit dem Vorgänger `ol` und dem Attribut `type` mit dem Wert `i`.
- Auch für diese Bedingung können Sie eine bestimmte Reaktion festlegen.

Andererseits können Sie keine Fragen beantworten, die eine Vorschau im Datenstrom erforderlich machen, beispielsweise:

- Ist ein Element das letzte Kindelement eines Elternelements?
- Das können Sie erst wissen, nachdem Sie das Element verarbeitet haben, und nicht, wenn Sie das Ereignis zum Start-Element erhalten.
- Enthält dieses Element ein Element mit dem Attribut `experience` und dem Wert `firsttime`?
- Auch das können Sie erst wissen, nachdem Sie das Element verarbeitet haben.

Es gibt natürlich Lösungen dafür, die aber einen zweiten Durchlauf erforderlich machen.

Vorteile dieser ereignisgesteuerten Verarbeitung sind:

- Sie ist einfach.
- Sie ist schnell.
- Sie verbraucht nicht viel Speicher.

Nachteil ist:

- Man kann nicht vorausschauen.

Hier werden zwei Implementierungen vorgestellt:

- Omnimark
- SAX

Omnimark ist der Marktführer für komplexe Konvertierungen in der SGML-Gemeinde. Seit kurzem ist es XML-fähig, und es gibt eine kostenlose (jedoch eingeschränkte) Version, Omnimark LE, im Web unter `http://www.omnimark.com/devleop/omle40/index.html`.

Omnimark wird in Kapitel 15 noch genauer beschrieben.

SAX steht für *Simple API for XML*. Er entstand, weil Peter Murray-Rust, einer der ersten Anwender von XML, eine Beschwerde in der Mailing-Liste der XML-Entwickler absetzte. Es ging darum, daß er den Code neu schreiben mußte, wenn er den an seinen XML-Browser, JUMBO, gekoppelten Parser ändern wollte, weil sich die APIs unterschiedlicher Parser unterschieden. Damit entstand die Frage: »Können wir uns auf ein einfaches, ereginisgesteuertes API einigen, während wir auf das von der W3C (DOM) definierte API warten?«

Diese Thematik wurde auf der XML-DEV-Liste öffentlich diskutiert, und viele Leute trugen dazu bei. David Megginson schrieb den SAX-Vorschlag ebenso wie seine Implementierung in Java.

Sie finden SAX unter `http://www.megginson.com/SAX/index.html`. SAX wird ebenfalls Thema von Kapitel 15 sein.

XML als Hierarchie/Baum

Ein baumbasierter Prozessor übersetzt das XML-Dokument in eine interne Baumstruktur und erlaubt einer Anwendung, sich innerhalb dieser Struktur zu bewegen.

Eine mögliche Baumstruktur für die Datei `musicians.xml` sehen Sie in Abbildung 14.4.

Bäume können mehr oder weniger umfangreich sein, abhängig davon, ob Attribute, Entities usw. als separate Knoten angelegt werden müssen.

Nachdem der Baum im Speicher angelegt wurde, kann er durchlaufen werden. Beachten Sie, daß es hier zwei Phasen gibt:

- Durchlauf 1: Parsing und Aufbau des Baums
- Durchlauf 2: Die eigentliche Datenverarbeitung

Auf diese Weise können Sie die Fragen beantworten, für die vorausgeschaut werden muß:

- Ist ein Element das letzte Kindelement seines Elternelements?
- Hat dieses Element noch ein untergeordnetes Element mit dem Attribut `experience` mit dem Wert `firsttime`?

Da Sie Zugriff auf das vollständige Dokument (also den gesamten Baum) haben, haben Sie auch Zugriff auf alle erforderlichen Informationen.

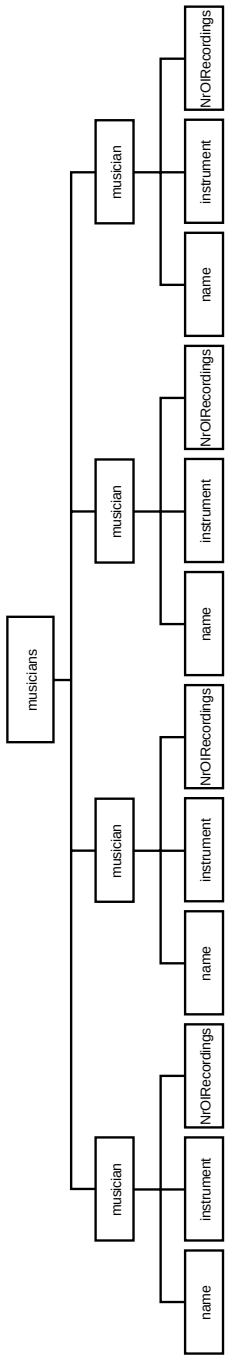


Abbildung 14.4: Ein Baum, der bei `musicians.xml` *beginnt.*

Der Vorteil bei diesem Ansatz ist, daß er Ihnen Zugriff auf das gesamte Dokument bietet, so daß ganz einfach vorausgeschaut werden kann. Die Nachteile sind, daß es schwieriger ist, einen Baum aufzubauen und ihn dann zu durchlaufen, daß er mehr Speicher verbraucht und daß er langsamer ist, weil er zwei Durchläufe bedingt.

Das W3C (World Wide Web Consortium) hat ein baumbasiertes Standard-API für XML und HTML entwickelt. Es wird als das *Document Object Model* (DOM) bezeichnet und ist eine W3C-Empfehlung vom 1. Oktober 1998. Die Spezifikation finden Sie unter `http://www.w3.org/TR/REC-DOM-Level-1/`.

Das DOM wird in Version 5 des Internet Explorer und von Mozilla (Netscape) implementiert. Eine genauere Beschreibung finden Sie in Kapitel 16.

Es gibt auch Software, die Ihnen beide Welten eröffnet und den ereignisgesteuerten mit dem baumbasierten Ansatz kombiniert. Balise von AIS ist in der SGML-Gemeinde dafür bekannt und beinhaltet in seiner neuesten Version einen nicht-analysierenden XML-Parser. Leider gibt es keine kostenlose Version.

Zusammenfassung

Heute haben Sie viele Gründe dafür kennengelernt, warum man XML-Dateien weiterverarbeiten sollte. Diese Verarbeitung kann auf unterschiedliche Weise stattfinden, abhängig davon, wie Sie Ihre XML-Daten betrachten:

- Als reine Textdateien
- Als Abfolge von Ereignissen
- Als Baum, der durchlaufen und manipuliert werden kann

In den nächsten Kapiteln werden Sie diese Perspektiven und ihre Implementierungen genauer kennenlernen.

F&A

F Ist XML einfacher zu verarbeiten als HTML?

A *Ja, aus zwei Gründen.*

Erstens können Sie Ihre Daten normalerweise mit XML besser modellieren als mit HTML. Sie können die hierarchische Struktur, die Semantik und die Metadaten Ihrer Informationen besser festhalten, ohne sich allzuviel mit Formatierungen beschäftigen zu müssen.

Der zweite Grund ist, daß XML-Dateien wohlgeformt sind. Sie können viel besser erkennen, was genau im Datenstrom steht. HTML-Dateien dagegen können viele Formen und Erscheinungsbilder annehmen.

F Welchen Verarbeitungsansatz empfehlen Sie?

A *Das ist von dem jeweiligen Problem abhängig.*

Wenn Sie beispielsweise zählen möchten, wie oft das Element `tool` in einem XML-Dokument auftritt, dann ist es nicht effizient, zuerst eine Baumdarstellung des Dokuments aufzubauen und diese dann zu durchlaufen. Wenn Sie Objekte mit komplexen Strukturen haben, wie beispielsweise Tabellen, wofür sehr viele Vorschauen erforderlich sind, vereinfacht der baumbasierte Ansatz die Verarbeitung.

Beachten Sie auch, daß das DOM, das baumbasiert ist, heute eine offizielle W3C-Empfehlung ist; viele Werkzeuge unterstützen es als Standard-API.

Übung

Betrachten Sie die folgende XML-Datei:

```
1: <?xml version="1.0"?>
 2: <memo>
 3:     <meta>
 4:         <from>P. Hermans</from>
 5:         <to>S. North</to>
 6:         <regarding>deadlines</regarding>
 7:     </meta>
 8:     <body>
 9:         <dear>Simon</dear>
10:         <p>I will <verystrong>not</verystrong> be able to finish
11: all chapters before leaving on 11..holidays.</p>
12:         <p>Please advise what to do.</p>
13:         <close>Paul</close>
14:     </body>
15: </memo>
```

Führen Sie anhand dieser Datei die folgenden Aufgaben aus:

- Schreiben Sie die sequentielle Abfolge der Ereignisse auf.
- Zeichnen Sie die Baumstruktur.
- Zeigen Sie an, in welcher Reihenfolge der Baum durchlaufen wird.
- Stellen Sie für das letzte `p`-Element im Baum fest, welche anderen Knoten nach dem ereignisgesteuerten Ansatz noch zu erreichen sind.
- Stellen Sie für das `body`-Element im Baum fest, welche anderen Knoten nach dem baumbasierten Ansatz noch zu erreichen sind.

Übersicht

Auf einen Blick

Ereignis-gesteuerte Programmierung

Woche 3

In Kapitel 14 haben wir die Unterschiede zwischen der ereignisgesteuerten und der baumbasierten Verarbeitung von XML-Dateien erklärt.

Heute werden wir uns detaillierter mit der ereignisgesteuerten Verarbeitung beschäftigen. Dabei demonstrieren wir die Verwendung von zwei Umgebungen für die ereignisgesteuerte Verarbeitung:

- Omnimark LE
- SAX

Omnimark LE

In diesem Teil beschäftigen wir uns mit der Programmiersprache Omnimark von Omnimark Technologies.

Was ist Omnimark LE?

Omnimark LE ist die eingeschränkte und kostenlose Version von Omnimark, einer ereignisgesteuerten Programmiersprache für die Verarbeitung von Datenströmen. Aktuell liegt es in Version 4.01 vor und ist XML-fähig.

Der XML-Parser von Omnimark ist analysierend und bedingt für die Verarbeitung von XML-Dokumenten eine gültige XML-DTD.

Diese kostenlose Version ist jedoch eingeschränkt – Sie können nur Programme mit weniger als 200 zählbaren Aktionen im Quellcode kompilieren und ausführen. Eine zählbare Aktion ist eine von Omnimark ausgeführte Anweisung.

Was als zählbare Aktion betrachtet wird, ist unter der folgenden URL beschrieben: `http://www.omnimark.com/develop/omle40/sampcode.html`.

Für kleine oder mittlere Konvertierungen ist die kostenlose Version gut geeignet.

Wenn Sie die Grenze der zählbaren Aktionen erreichen, können Sie Ihr Skript in mehrere Teile zerlegen und sie nacheinander ausführen oder die unter `http://www.sesha.com/omlette/` beschriebenen Tips und Techniken anwenden.

Omnimark LE laden und installieren

Sie finden Omnimark LE unter der URL `http://www.omnimark.com/develop/omle40/index.html`. Die Datei `omle40.exe` ist 1955 Kbyte groß.

Zur Installation doppelklicken Sie auf diese Datei. Der Installationsassistent führt Sie durch die Installation.

Die Arbeitsweise von Omniwork

Im ereignisgesteuerten Paradgima werden Codeabschnitte ausgeführt, wenn während der Verarbeitung des Eingabedokuments bestimmte Ereignisse auftreten.

Wir müssen also Regeln formulieren. Regeln definieren die auszuführenden Aktionen sowie die Ereignisse, durch die sie ausgelöst werden.

Die Struktur einer Regel setzt sich aus zwei Komponenten zusammen:

- Einem Regelkopf
- Einem Regelrumpf

Im Regelkopf wird das Ereignis definiert. Er hat eine eigene Struktur und definiert das Ereignis und optional weitere Bedingungen, die erfüllt sein müssen, damit bestimmte Aktionen ausgelöst werden.

Im Regelrumpf befinden sich die Aktionen, die ausgeführt werden sollen.

Ein Omnimark-Skript/-Programm besteht aus mehreren solchen Regeln.

Ausführung von Omnimark LE

Omnimark wird von der Befehlszeile aus ausgeführt:

```
Omle -s script.xom input.xml -of output.htm
```

Dabei gilt:

- `omle` ruft das Programm auf.
- Der Parameter `-s` verweist auf das zu verwendende Skript (beispielsweise `script.xom`).
- Als nächstes kommt die Eingabedatei, in unserem Fall eine XML-Datei (beispielsweise `input.xml`).
- `-of` gibt die Datei an, in die die Ausgaben geschrieben werden (beispielsweise `output.htm`).

Grundlegende Ereignisse in Omnimark

Omnimark erkennt unter anderem die folgenden grundlegenden Ereignisse:

- document-start
- document-end
- element
- processing-instruction

`document-start` erlaubt Ihnen, beim Erkennen eines XML-Dateianfangs eine Verarbeitung auszuführen und Ausgaben zu erzeugen, wie in Listing 15.1 gezeigt.

Listing 15.1: Eine `document-start`-Regel

```
1:  document-start  ;event
2:        output "<HTML>%n"     ;unsere Aktion
3:        !! "<HEAD>%n"
4:        !! "<TITLE>Our first example</TITLE>%n"
5:        !! "</HEAD>%n<BODY>"
```

Der Regelkopf in Zeile 1 definiert das Ereignis `document-start`.

Der Regelrumpf (Zeilen 2 bis 5) gibt die auszuführenden Aktionen an. Beim Starten des Dokuments (Ereignis) schreiben wir einen String mit HTML-Tags in die Ausgabe (Aktion).

`%n` steht für »Zeilenende«, End-of-Line

`!!` verknüpft zwei Strings

`;` kennzeichnet einen Kommentar in Ihrem Code

Diese Regel erzeugt das folgende Ergebnis:

```
<HTML>
  <HEAD>
  <TITLE>Our first example</TITLE>
  </HEAD>
  <BODY>
```

`document-end` erlaubt Ihnen, beim Erkennen eines XML-Dateiendes eine Verarbeitung vorzunehmen und Ausgaben zu erzeugen, wie in Listing 15.2 gezeigt.

Listing 15.2: Eine `document-end`-Regel

```
1:  document-end ;event
2:       output "</BODY>%n</HTML>%n"
```

Wenn das Dokument endet (Ereignis), schreiben wir einen String mit HTML-Tags in die Ausgabe (Aktion).

Diese Regel erzeugt das folgende Ergebnis:

```
</BODY>
  </HTML>
```

Zeile 3, die leer ist, ist das Ergebnis des `%n` (Zeilenende) hinter dem HTML-Ende-Tag in Zeile 2 von Listing 15.2.

Tritt ein `element` auf, müssen Sie drei Dinge verarbeiten:

- Den Anfang von `element`
- Seinen Inhalt
- Das Ende von `element`

Für den Inhalt von `element` haben Sie beim Parsing in Omnimark nicht mehr die Möglichkeit, festzulegen, wann und wie dieser verarbeitet werden soll. Sie müssen das Parsing explizit ausführen. Dazu gehen Sie nach einer der beiden folgenden Methoden vor:

- Sie verwenden den Operator zur Fortsetzung des Parsings, `%c`, wie in Listing 15.3 gezeigt.
- Sie verwenden `suppress`, so daß das Parsing fortgesetzt, aber die Ausgabe des geparsten Inhalts unterdrückt wird, wie in Listing 15.4 gezeigt.

Listing 15.3: Der Operator für die Fortsetzung des Parsings

```
1:  element musician   ; event, when the element musician is encountered
2:       output "<HR>%n" ; at the start of the element
3:       output "%c"     ; continue parsing
4:       output "<HR>%n" ; at the end of the element
```

oder

Listing 15.4: Die `suppress`-Aktion

```
1:  element meta   ; event, when the element meta is encountered
2:      suppress ; continue parsing but suppress output
```

Jede Regel muß entweder den Operator für die Fortsetzung des Parsings (`%c`) oder `suppress` enthalten.

Achten Sie darauf, in jeder Regel `%c` oder `suppress` anzugeben.

Abhängig von der Elementattributinformation können unterschiedliche Aktionen erforderlich sein. Listing 14.3 zeigt ein Beispiel für die Bereitstellung unterschiedlicher Informationen, abhängig davon, welches Vorwissen der Benutzer mitbringt:

```
<steps experience="firsttime">…</steps>
```

oder

```
<steps experience="donebefore">…</steps>
```

Omnimark erlaubt Ihnen, abhängig von dieser Attributinformation Bedingungen in Ihre Regelköpfe einzuführen, wie in Listing 15.5 gezeigt.

Listing 15.5: Verwendung einer Bedingung, die den Attributwert auswertet

```
1:  element steps when attribute experience = "firsttime"
2:       ;action
```

Diese Regel wird nur ausgelöst, wenn wir im XML-Dokument auf das Element `steps` treffen, das das Attribut `experience` mit dem Wert `firsttime` besitzt.

Für jedes Element im Dokument darf nur eine einzige Elementregel definiert werden.

In Kapitel 14 haben wir gezeigt, daß ereignisgesteuerte Systeme Informationen über zuvor durchlaufene Knoten verwalten. Das ist auch bei Omnimark so.

Omnimark bietet Ihnen Zugriff auf:

- Das Elternelement
- Die Vorgängerelemente
- Elemente vor den Elternelementen
- Alle geöffneten Elemente

Sie können auch abfragen, welche Elemente zuvor geschlossen wurden:

- Das vorhergehende Element
- Das letzte Unterelement verweist auf das zuletzt geschlossene Unterelement

Die Listings 15.6 bis 15.9 zeigen einige Beispiele, wie diese Kontextinformation berücksichtigt werden kann.

Listing 15.6: Überprüfung des Elternelements

```
1:       element li when parent is ol
2:       ;action
```

Diese Regel wird ausgelöst, wenn wir im XML-Dokument auf ein `li`-Element treffen, das `ol` als Elternelement hat.

Listing 15.7: Überprüfung der Vorgängerelemente

```
1:  element p when ancestor is li
2:       ; action
```

Diese Regel wird ausgelöst, wenn das Element `p` mit dem Vorgängerelement `li` im XML-Dokument angetroffen wird.

Listing 15.8: Überprüfung des vorhergehenden Elements

```
1:  element p when previous is title
2:       ; action
```

Diese Regel wird ausgelöst, wenn wir im XML-Dokument auf ein `p`-Element nach einem `title`-Element treffen.

Diese Kontextüberprüfungen können auch kombiniert werden, wie in Listing 15.9 gezeigt.

Listing 15.9: Eine Kombination der Kontextbedingungen

```
1:  element def when parent of parent is ol
2:       ; action
```

Diese Regel wird ausgelöst, wenn wir auf ein `def`-Element treffen, das ein Großelternelement namens `ol` im XML-Dokument hat.

Sie können den Kontext auch definieren und überprüfen, indem Sie Elemente zählen:

- Auftreten – Gibt an, wie oft ein Element wiederholt wurde, wie in Listing 15.10 gezeigt.
- Anzahl der Kindelemente – Zählt, wie oft ein Kindelement des angegebenen Elements bisher aufgetreten ist, wie in Listing 15.11 gezeigt.

Listing 15.10: Eine vom Auftreten abhängige Bedingung

```
1:  element li when occurrence = 2
2:       ;action
```

Diese Regel wird ausgelöst, wenn das Element `li` das zweite in einer Abfolge von `li`-Elementen ist.

Was Sie tun sollten Verwenden Sie statt dessen »number of children« (Anzahl der Kindelemente).

Was Sie nicht tun sollten Verwenden Sie `occurrence` nicht, um das n-te Element im Elternelement zu finden.

Listing 15.11: Eine von der Anzahl der Kindelemente abhängige Bedingung

```
1:  element p when children of parent = 1
2:       ;action
```

Diese Regel wird ausgelöst, wenn das Element `p` das erste Kindelement seines Elternelements ist.

Normalerweise werden `processing instructions` ignoriert, es sei denn, Sie geben einen Ereignis-Handler für Verarbeitungsanweisungen an.

Beispiel:

```
processing-instruction "break"
       output "<BR>"
```

Vorausschau

In Kapitel 14 haben wir über die Nachteile des ereignisgesteuerten Ansatzes gesprochen. Es ist nicht möglich, Entscheidungen abhängig von Informationen zu treffen, die später im Datenstrom auftreten.

Betrachten Sie den folgenden XML-Abschnitt.

Listing 15.12: Ein XML-Abschnitt, der konvertiert werden soll.

```
1: <cities id="L5">
2:        <city>Amsterdam</city>
3:        <city>Paris</city>
```

```
4:        <city>Toronto</city>
5:        <city>Vancouver</city>
6:        <city>Brussels</city>
7: </cities>
```

Ich möchte diesen XML-Abschnitt in die folgende HTML-Darstellung umwandeln:

```
<p>City 1 of 5<BR>
   Amsterdam</p>
<p>City 2 of 5<BR>
   Paris</p>
...
```

Wenn wir im Datenstrom auf die Stadt Amsterdam treffen, wollen wir vielleicht wissen, an welcher Position sie steht, aber es ist zu diesem Zeitpunkt noch nicht möglich, zu wissen, daß insgesamt fünf Städte aufgelistet werden.

Um diese Situation aufzulösen, verwendet Omnimark sogenannte Referenten.

Ein Referent gibt einen Wert aus, der an einer anderen Stelle der Verarbeitung ermittelt wurde, insbesondere später im Datenstrom.

Mit Hilfe von Referenten können Sie Platzhalter in Ihrer Ausgabe anlegen. Später weisen Sie ihnen Werte zu, wie in Listing 15.13 gezeigt.

Listing 15.13: *nrofcities.xom* ***– Verwendung von Referenten***

```
1:  global counter nrofcities initial {0} ; a variable with
➥ name "nrofcities" and value "0"
2:
3:  element cities ;rule
4:        output "%c" ; continue parsing
5:
6:  element city
7:        increment nrofcities ; the variable "profcities"
➥is incremented (+1)
8:        output "<p>"
```

```
9:       output "City %d(nrofcities) of " ; the value of
➥variable "nrofcities" is sent to the output
10:      output referent "totalofcities"
11:              ; the referent is the placeholder for the
➥not known yet value of the total of cities
12:              ; the content of the placeholder is sent to
➥the output
13:      output "<BR>"
14:      output "%c"
15:      output "</p>"
16:      set referent "totalofcities" to "%d(nrofcities)"
17:              ; the value of "nrofcities" is placed in
➥the placeholder
18:              ; after the first its 1, then 2, and
➥finally for our example 5
```

In Zeile 1 deklarieren wir die Variable `nrofcities`. Der Ausgangswert ist 0.

In Zeile 3 haben wir unsere Elementregel für das Element `cities`. Die hier definierte Aktion ist `continue parsing`.

In Zeile 6 definieren wir unsere Elementregel für das Element `city`. Als erstes inkrementieren wir die Variable `nrofcities`. Für Amsterdam ist der Wert gleich `1`, für Paris gleich `2` usw. In Zeile 8 beginnen wir, in die Ausgabe zu schreiben: `<p>City` gefolgt vom aktuellen Wert der Variablen `nrofcities`, gefolgt von `of`, gefolgt vom Platzhalter `totalofcities`, dessen Wert wir noch nicht kennen.

In Zeile 16 wird der Wert des Platzhalters `totalofcities` auf den Wert von `nrofcities` gesetzt. Das bedeutet, nach der Verarbeitung der Datei ist der Wert von `totalofcities` gleich 5.

Weil wir Referenten verwendet haben, muß die Datei ein zweites Mal durchlaufen werden. In diesem zweiten Durchgang wird der Platzhalter durch seinen endgültigen Wert ersetzt.

Der Referentenmechanismus ist auch sehr praktisch zum Anlegen aller möglichen Arten von Hypertext-Links.

Eingaben und Ausgaben

Omnimark ist nicht nur eine ereignisgesteuerte Verarbeitungssprache, sondern auch eine Streaming-Sprache. Eingabe- und Ausgabeströme können ganz einfach verarbeitet werden, was sehr praktisch ist, wenn Sie eine große XML-Datei in mehrere kleinere Abschnitte unterteilen wollen.

Andere Funktionen

Omnimark bietet eine vollständige Umgebung mit gebräuchlichen Konstrukten aus normalen Programmiersprachen, wie etwa Bedingungsanweisungen, Schleifen, Variablen, Funktionen (auch extern in C, C++ oder Java geschrieben), Makros oder Arrays – auch assoziative Arrays.

Insbesondere soll die Mustersuchsprache erwähnt werden; Muster erzeugen ebenfalls Ereignisse, wenn sie erkannt werden.

Beispiel für ein Omnimark-Skript

Listing 15.14: *biblio.xml* ***Konvertierung nach HTML***

```
1:  <?xml version="1.0"?>
2:  <!DOCTYPE biblio [
3:  <!ELEMENT biblio (books, authors)>
4:  <!ELEMENT books (book)+ >
5:  <!ELEMENT book (title, author+, description, ISBN, pages,
➥targetgroup, price, related*) >
6:  <!ATTLIST book id ID #REQUIRED>
7:  <!ELEMENT title (#PCDATA) >
8:  <!ELEMENT author (#PCDATA) >
9:  <!ELEMENT description (#PCDATA) >
10:  <!ELEMENT ISBN (#PCDATA) >
11:  <!ELEMENT pages (#PCDATA) >
12:  <!ELEMENT targetgroup (#PCDATA) >
13:  <!ELEMENT price (#PCDATA) >
14:  <!ATTLIST price unit CDATA "USA">
15:  <!ELEMENT related (#PCDATA) >
16:  <!ATTLIST related linkend IDREF #REQUIRED>
17:  <!ELEMENT authors (authordesc)+ >
18:  <!ELEMENT authordesc (name, specialty+) >
19:  <!ELEMENT name (#PCDATA)>
20:  <!ELEMENT specialty (#PCDATA)>
21:  ]>
```

```
22:  <biblio>
23:  <books>
24:       <book id="TY1">
25:            <title>Sam's Teach Yourself C++ in 21 Days, Second Edition
26:            </title>
27:            <author>Jesse Liberty
28:            </author>
29:            <description>This book teaches you the basics of
➥object-oriented programming with C++ and is completely
➥revised to ANSI standards. It can be used with any
➥C++ compiler.
30:            </description>
31:            <ISBN>0-672-31070-8
32:            </ISBN>
33:            <pages>700
34:            </pages>
35:            <targetgroup>Beginning - Intermediate
36:            </targetgroup>
37:            <price unit="USA">29.99
38:            </price>
39:            <related linkend="TY2">Teach Yourself Visual C++ 5
➥</related>
40:       </book>
41:         <book id="MJ">
42:            <title>Maximum Java 1.1
43:            </title>
44:            <author>Glenn Vanderburg
45:            </author>
46:            <description>Written by JAVA experts, this book explores
➥the JAVA 1.1 language, tools, and core JAVA API without
➥reviewing fundamentals or basic techniques.
47:            </description>
48:            <ISBN>1-57521-290-0
49:            </ISBN>
50:            <pages>900
51:            </pages>
52:            <targetgroup>Expert
53:            </targetgroup>
54:            <price unit="USA">49.99
55:            </price>
56:      </book>
57:      <book id="JS">
58:            <title>JavaScript Unleashed, Second Edition
59:            </title>
60:            <author>Richard Wagner
61:            </author>
```

```
62:             <description>This book helps you thoroughly understand
➥and apply JavaScript.
63:             </description>
64:             <ISBN>1-57521-306-0
65:             </ISBN>
66:             <pages>1000
67:             </pages>
68:             <targetgroup>Casual - Experienced
69:             </targetgroup>
70:             <price>49.99
71:             </price>
72:       </book>
73:       <book ID=”TY2”>
74:             <title>Sam's Teach Yourself Visual C++ 5 in 21 Days,
➥Fourth Edition
75:             </title>
76:             <author>Nathan Gurewich
77:             </author>
78:             <author>Ori Gurewich
79:             </author>
80:             <description>This book merges the power of the best-➥selling
”Teach Yourself” series with the knowledge of Nathan and
➥Ori Gurewich, renowned experts in code, creating the most
➥efficient way to learn Visual C++.
81:             </description>
82:             <ISBN>0-672-31014-7
83:             </ISBN>
84:             <pages>832
85:             </pages>
86:             <targetgroup>New - Casual
87:             </targetgroup>
88:             <price>35.00
89:             </price>
90:       </book>
91:  </books>
92:  <authors>
93:       <authordesc>
94:             <name>Jesse Liberty
95:             </name>
96:             <specialty>C
97:             </specialty>
98:      </authordesc>
99:      <authordesc>
100:            <name>Glenn Vanderburg
101:            </name>
102:            <specialty>Java
```

```
103:            </specialty>
104:    </authordesc>
105:    <authordesc>
106:            <name>Richard Wagner
107:            </name>
108:            <specialty>Opera
109:            </specialty>
110:    </authordesc>
111:    <authordesc>
112:            <name>Nathan Gurewich
113:            </name>
114:            <specialty>Visual Basic
115:            </specialty>
116:            <specialty>Visual C++
117:            </specialty>
118:    </authordesc>
119:  </authors>
120:  </biblio>
```

Wir geben dieser Datei den Namen `biblio.xml` und schreiben ein Skript, das das XML in HTML umwandelt.

Als erstes schreiben wir Ereignis-Handler, die alle XML-Elemente in HTML-Elemente umwandeln, wie in Listing 15.15 gezeigt.

***Listing 15.15:** `tohtml.xml` – unser Omnimark-Skript zur Umwandlung der Datei `biblio.xml` in HTML*

```
1:  DOWN-TRANSLATE with XML
2:
3:  element biblio
4:       output "<HTML>%n"
5:       ¦¦ "<HEAD>%n"
6:       ¦¦ "<TITLE>My bookshop</TITLE>%n"
7:       ¦¦ "</HEAD>%n"
8:       ¦¦ "<BODY>%n"
9:       ¦¦ "<H1>My bookshop</H1>%n<P> </P>%n"
10:      ¦¦ "%c" ; continue parsing
11:
12:  element books
13:      output "%c"
14:
15:  element book
16:      output "%c"
17:      do when last subelement is related
18:            output "</P>"
```

```
19:       done
20:       output "<HR>%n"
21:
22:   element title
23:       output "<H2 style='margin-top:40pt'>"
24:       using attribute id of parent
25:             do when attribute id is specified
26:             output "<A name='%v(id)'></A>"
27:             done
28:       output "%c</H2>%n"
29:
30:   element author
31:   do when previous is title
32:             output "<P><I>%c</I>"
33:       else
34:             output "<BR>%n"
35:             ¦¦ "<I>%c</I>"
36:       done
37:
38:   element description
39:       output "</P>%n" ; closing the p element started [cc]
with the first author
40:      ¦¦ "<HR>%n"
41:      ¦¦ "<P>%c</P>%n"
42:
43:   element ISBN
44:      output "<P><CODE>ISBN-number: %c</CODE><BR>%n"
45:
46:   element pages
47:      output "%c pages<BR>%n"
48:
49:   element targetgroup
50:      output "Aimed at the '%c'<BR>%n"
51:
52:   element price
53:      output "%c US$</P>%n"
54:
55:   element related
56:      do when occurrence = 1
57:             output "<P><I>See also: </I><BR>"
58:      done
59:      output "<A HREF='#%v(linkend)'>%c</A><BR>"
60:
61:   element authors
62:      output "<H2 style='margin-top:40pt'>An overview of
➥our writers</H2>%n"
```

```
63:     output "<TABLE CELLPADDING='5'>"
64:     output "<TR><TH>Name</TH><TH>Specialty</TH></TR>%n"
65:     output "%c</TABLE>%n"
66:
67:  element authordesc
68:     output "<TR>%c</I></TD></TR>%n"
69:
70:  element name
71:     output "<TD>%c</TD>%n"
72:
73:  element specialty
74:     do when occurrence = 1
75:            output "<TD><I>%c"
76:     else
77:            output " - %c"
78:     done
79:
80:  document-end
81:     output "</BODY></HTML>"
```

Wir speichern dieses Skript unter dem Namen `tohtm.xom`.

Sie sehen, daß wir für jedes XML-Element eine `element`-Regel definiert haben.

Zuvor haben wir Bedingungen nur als Teil des Regelkopfs gesehen, aber Bedingungen können auch Teil des Regelrumpfs sein, wo die Aktionen definiert werden. Betrachten Sie das Element `specialty` in Zeile 73. Dort habe Sie beginnend ab Zeile 74 das Konstrukt `do when ... , else..., done`, wobei zwischen einem `specialty`-Element, das als erstes in einer ganzen Folge von `specialty`-Elementen auftritt, und den später auftretenden `specialty`-Elementen unterschieden wird.

Auch die Zeilen 24, 25 und 26 sollen genauer erklärt werden. Für das Element `title` wird das Attribut `id` des Elternelements von `title` ausgewertet, also `book` (Zeile 24). Wurde für diese Attribut-ID ein Wert angegeben (Zeile 25), wird ein HTML-Anker in die Ausgabe geschrieben, wobei der Attributname den Wert dieser ID annimmt (Zeile 26).

Jetzt führen wir in der Befehlszeile folgendes aus:

```
omle -s tohtml.xom biblio.xml -of biblio.htm
```

In diesem Beispiel setzen wir voraus, daß alle Dateien im selben Ordner wie das Programm `omle.exe` abgelegt sind.

Damit erzeugen Sie die in Abbildung 15.1 gezeigte Datei `biblio.htm`.

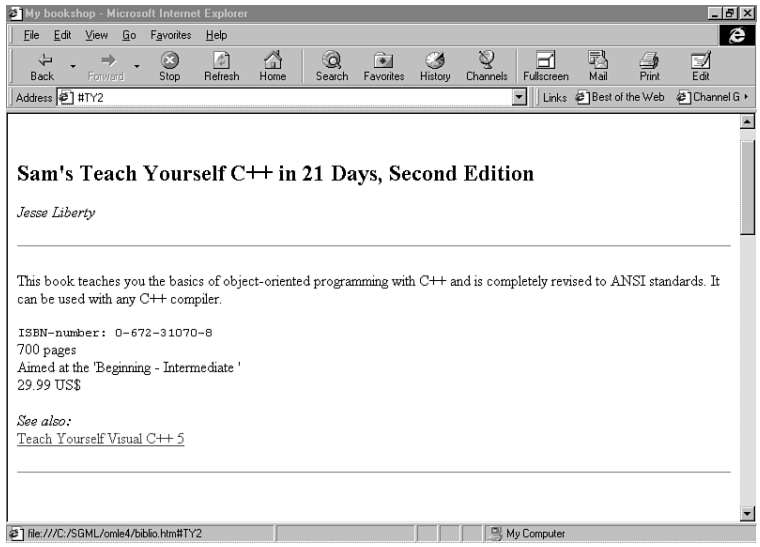


Abbildung 15.1: Das Ergebnis unserer ersten Umwandlung.

Jetzt versuchen wir, die Datei aufzuteilen und für jedes Buch eine separate HTML-Datei und für den Autorenüberblick eine separate Datei anzulegen.

Wir müssen den Code für zwei Elemente ändern: `book` und `authors`, wie in Listing 15.6 gezeigt.

***Listing 15.16:** `tohtml.xml` – Umwandlung in mehrere HTML-Dateien*

```
1:  element book
2:       local stream bookhtm ;creating a variable with
➥name "bookhtm"" of type stream
3:       open bookhtm as file "%v(id).htm"
; this stream is written to a file,
➥based on the id of the book element
4:       using output as bookhtm ; now we write the output to
➥this stream (file)
5:       do ;everything between 'do' and 'done' needs to go to the stream
6:              output "<HTML>%n"
7:              ¦¦ output "<HEAD>%n"
```

```
8:              ¦¦ output "<TITLE>Book %v(id)</TITLE>"%n
9:              ¦¦ output "</HEAD>%n"
10:             ¦¦ output "<BODY>%n"
11:             output "%c"
12:             do when last proper subelement is related
13:                    output "</P>"
14:             done
15:             output "<HR>%n"
16:             output "</BODY></HTML>%n"
17:       done
18:
19:  element authors
20:       local stream authorshtm
21:       open authorshtm as file "authors.htm"
22:       using output as authorshtm
23:       output "<HTML>%n"
24:             ¦¦ output "<HEAD>%n"
25:             ¦¦ output "<TITLE>Authors</TITLE>"%n
26:             ¦¦ output "</HEAD>%n"
27:             ¦¦ output "<BODY>%n"
28:             output "<H2 style='margin-top:40pt'>An overview of our ➥writers</
H2>%n"
29:             output "<TABLE CELLPADDING='5'>"
30:             output "<TR><TH>Name</TH><TH>Specialty</TH></TR>%n"
31:             output "%c</TABLE>%n"
32:             output "</BODY></HTML>%n"
33:             done
```

Wir deklarieren für jedes `book`-Element eine lokale Variable mit dem Typ `stream`, `bookhtm`. In Zeile 3 öffnen wir den Stream als Datei, deren Name durch den String-Ausdruck `%v(id).htm` angegeben wird, das ist der Wert der Attribut-ID, für das erste Buch in unserer XML-Datei also `TY1`. Diesem Wert wird ein `.htm` angefügt, wir erhalten also `TY1.htm`. In Zeile 4 schreiben wir unsere Ausgabe in diese Datei. Weil wir jetzt in unterschiedliche Dateien schreiben, müssen wir in jede dieser Dateien die entsprechende HTML-Start- und Ende-Information schreiben, was in den Zeilen 6 bis 10 und 16 passiert. Dieselbe Technik wird auf das Element `authors` angewendet.

Wir führen das Programm noch einmal aus und erhalten damit sechs Dateien:

- biblio.htm
- TY1.htm
- MJ.htm

- JS.htm
- TY2.htm
- authors.htm

Das ist nicht schlecht, aber es gibt zwei Probleme:

1. Der Link, den der Ereignis-Handler des Elements `related` erzeugt, funktioniert nicht mehr. Er wurde als Link angelegt, der auf eine Stelle innerhalb desselben Dokuments verweist. Jetzt muß ein Link daraus werden, der auf ein anderes Dokument verweist.
2. Wir haben keine Navigationshilfe, die uns in die anderen Dateien bringt. Wir bräuchten zweierlei:
   - Ein TOC (Inhaltsverzeichnis) innerhalb der ersten `biblio.htm`-Datei mit einem Überblick über alle Bücher.
   - Links auf jeder Seite, mit denen man zum vorhergehenden oder nächsten Buch weitergehen kann.

Zunächst lösen wir das Problem mit dem defekten Link.

Der Link innerhalb derselben Datei auf den Anker,

```
<A HREF="#TY2">.
```

wird wie folgt abgeändert:

```
<A HREF="TY2.htm">
```

Wir ändern das Omnimark-Skript wie folgt ab:

```
55  element related
        ...
59      output "<A HREF='%v(linkend).htm'>%c</A><BR>"
```

Beachten Sie, daß die Referenten nur Platzhalter sind, deren Inhalt später definiert wird.

Um ein TOC zu erzeugen, fügen wir zuerst eine globale Stream-Variable ein, die als Referent für unser Skript verwendet wird, wie in Listing 15.17 gezeigt.

***Listing 15.17:** `tohtml.xml` – Einfügen eines TOC*

```
1:  global stream toc; to be used later on as a referent
2:
3:  element biblio
4:        output "<HTML>%n"
```

```
5:        ¦¦ "<HEAD>%n"
6:        ¦¦ "<TITLE>My bookshop</TITLE>%n"
7:        ¦¦ "</HEAD>%n"
8:        ¦¦ "<BODY>%n"
9:        ¦¦ "<H1>My bookshop</H1>%n<P></P>%n"
10:
11:       output referent "toc" ; here we will (out)put our toc
12:
13:       open toc as referent "toc" ; and we initialise it
14:       put toc "<HR><H3>Table of Contents</H3>%n<UL>"
15:       close toc
16:
17:       output "%c" ; continue parsing
```

Probieren Sie es aus. Sie erhalten den Titel »Table of Contents« (toc, Inhaltsverzeichnis) in Ihrer `biblio.htm`-Datei.

In Zeile 11 haben wir den Inhalt unseres Platzhalters mit dem Namen »toc« ausgegeben. Dieser Platzhalter wurde in den Zeilen 13 bis 15 mit dem Titel gefüllt (Zeile 14).

Wir wollen dem Inhaltsverzeichnis die Titel unserer Bücher hinzufügen; deshalb müssen wir jedesmal zusätzliche Informationen in unser Platzhalter-TOC schreiben, wenn wir einen Buchtitel erkennen, wie in Listing 15.18 gezeigt.

Listing 15.18: *`tohtml.xml`* ***– das toc wird gefüllt***

```
1:   element title
2:        ; putting the content into this variable since we
➥will use this content twice
3:        local stream titlecontent ; creating a variable of
➥type stream
4:        open titlecontent as buffer
5:        put titlecontent "%c"
6:        close titlecontent
7:
8:        ; existing code
9:        output "<H2 style='margin-top:40pt'>"
10:        using attribute id of parent
11:               do when attribute id is specified
12:               output "<A name='%v(id)'></A>"
13:               done
14:   output "%g(titlecontent)</H2>%n"
```

```
15:
16:   ; referent
17:   reopen toc as referent "toc"
18:       using attribute id of parent
19:       put toc "<LI><A HREF='%v(id).htm'>%g(titlecontent)</A></LI>"
20:       close toc
```

Zuerst schreiben wir den Inhalt unseres Titels in eine lokale Variable, `title-content` (Zeilen 3 bis 7). Dieser gepufferte Inhalt wird in Zeile 15 wiederverwendet und in ein `H2`-HTML-Element ausgegeben.

Der wichtigste Abschnitt beginnt in Zeile 18, wo unser Platzhalter `toc` wieder geöffnet und der Inhalt in Zeile 20 zum bereits existierenden Inhalt unseres Referenten hinzugefügt wird.

Sie sehen das Ergebnis in Abbildung 15.2.

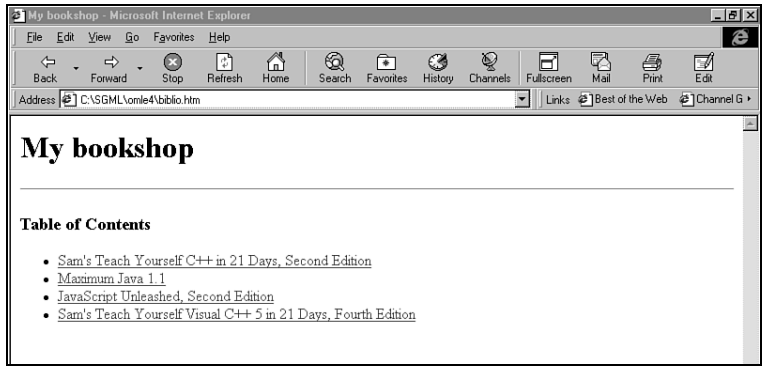


Abbildung 15.2: Das durch unser Skript erzeugte Inhaltsverzeichnis.

Jetzt werden unten auf jeder Seite die Links eingefügt, die zum vorhergehenden und zum nächsten Buch führen.

Wir beginnen mit einem Link auf die vorhergehende Seite. Weil wir diese Seite bereits übergeben haben, können wir diese Information puffern und sie auf unserer aktuellen Seite verwenden, wie in Listing 15.19 gezeigt.

***Listing 15.19:** `tohtml.xml` – ein Link auf die vorhergehende Seite wird eingefügt*

```
1:   element book
2:       local stream bookhtm
3:       open bookhtm as file "%v(id).htm"
4:       using output as bookhtm
5:       do
6:              output "%c"
7:              do when last proper subelement is related
8:                     output "</P>"
9:              done
10:              output "<HR>%n"
11:              output "%g(prevpage)" ; here we add our previous link
12:              set prevpage to "<P><A HREF='%v(id).htm'>Previous</A></P>"
13:              ; and we keep the information of this page for the next one
14:       done
```

In Zeile 12 setzen wir die Variable `prevpage` auf einen Link mit dem Namen unserer aktuellen HTML-Seite. Dieser Inhalt wird in Zeile 11 für die nächste Seite verwendet.

Listing 15.20 zeigt, wie man den Link auf die nächste Seite anlegt. Hier braucht man einen Referenten.

***Listing 15.20:** `tohtml.xml` – ein Link auf die nächste Seite wird angelegt*

```
1:   element book
2:       local stream bookhtm
3:       set referent "next-%v(id)" to ""
4:   ; the next on this page is set to empty
5:   ; this is done since we don't know yet if there is a next page
6:       set referent "next-%g(previd)" to "<A HREF='%v(id).htm'>Next</A>"
7:       ; the next of previous page gets the info of this (being next) page
8:       open bookhtm with referents-allowed as file "%v(id).htm"
9:       ; referents-allowed added since we put placeholders in this stream now
10:       using output as bookhtm ; writing to the file
11:       do
12:              output "%c"
13:              do when last proper subelement is related
14:                     output "</P>"
15:              done
16:              output "<HR>%n"
```

```
17:              output "%g(prevpage)"          ; the info of previous page
18:              output referent "next-%v(id)" ; our placeholder
19:              set prevpage to "<P><A HREF='%v(id).htm'>Previous</A> 
 "
20:              set previd to "%v(id)" ; used in next page for filling in
➥placeholder
21:       done
```

Nach der Ausgabe unseres Links auf die vorherige Seite geben wir einen Link auf die nächste Seite aus, aber wir kennen seinen Inhalt noch nicht – deshalb der Referent in Zeile 18. Der Name dieses Referenten verweist auf den `id`-Wert des Buchs.

Weil wir jetzt auch noch nicht wissen, ob es überhaupt eine nächste Seite gibt, setzen wir den Inhalt des Links auf die nächste Seite auf eine leere Zeichenkette »« (Zeile 3). Gibt es eine nächste Seite, wird dieser Inhalt durch die Aktion in Zeile 6 überschrieben. Beachten Sie, daß wir die ID des Buchs am Ende des Regelrumpfs in der Variablen `previd` festgehalten haben. Dieser Wert wird in Zeile 6 verwendet, um den Platzhalter auf der vorherigen Seite einzutragen.

Abbildung 15.3 zeigt das Ergebnis.

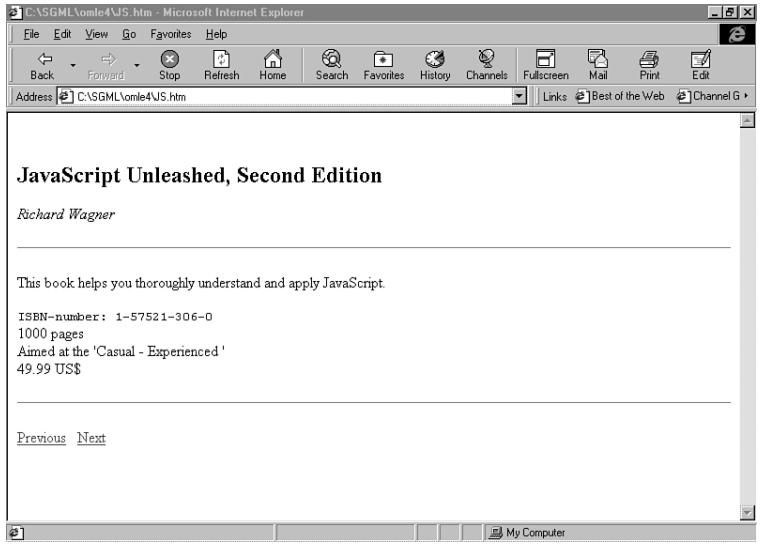


Abbildung 15.3: Unsere HTML-Dateien mit den Navigations-Links.

Weitere Informationen

Weitere Tips und Techniken zur Konvertierung von XML in HTML finden Sie unter der folgenden URL:

```
http://www.omnimark.com
```

Wenn Sie Omnimark in der Praxis sehen wollen, laden Sie den Omnimark 4.0 Samples Viewer unter `http://www.omnimark.com/magazine/tips/freeware.html` herunter.

SAX

SAX steht für *Simple API for XML*. Dabei handelt es sich um eine Standardschnittstelle für ereignisbasiertes XML-Parsing – entstanden durch die Zusammenarbeit von Mitgliedern der XML-DEV-Mailingliste, die von David Megginson koordiniert und finanziert wird.

Momentan gibt es SAX-Implementierungen in Java und Python.

Die folgenden Parser unterstützen heute die SAX-Schnittstelle:

- XML for Java (IBM)
- XP (James Clark)
- DXP (Datachannel)
- Ælfred (Microstar)
- SXP (Silfide)
- XML Library (Sun)

Dank Treibern von Drittanbietern können dieser Liste zwei weitere Parser hinzugefügt werden:

- Lark (Tim Bray)
- MSXML (Microsoft)

Die folgenden Abschnitte beziehen sich auf die SAX-Java-Distribution von David Megginson (`http://www.megginson.com`).

Alles in allem

Für die Entwicklung einer SAX-Anwendung brauchen Sie:

- Einen XML-Parser, der SAX unterstützt
- Eine SAX-Distribution, wobei es sich um eine Sammlung von Java-Klassen und Schnittstellen handelt. Sie finden sie unter `http://www.megginson.com/SAX/sax-java-1.0.zip`.
- Ereignis-Handler, die Sie basierend auf diesen Schnittstellen und Klassen definieren.

Hintergrundinformationen zu OO und Java-Konzepten

Klassen sind abstrakte Blaupausen, aus denen individuelle Objekte angelegt werden können. Sie definieren Eigenschaften und Verhalten (Methoden).

Eine *Klasse* ist eine Schablone für die Definition des Verhaltens und der Eigenschaften eines bestimmten Objekttyps.

In Tabelle 15.1 sehen Sie ein Beispiel für eine Klasse.

Konzept	Beispiel
Klasse	Auto
Eigenschaft	Geschwindigkeit
Verhalten	Gang wechseln

Tabelle 15.1: Ein Beispiel für eine Klasse.

Anhand der Schablone in Tabelle 15.1 können wir ein Auto (Objekt) anlegen, das eine bestimmte Geschwindigkeit hat und den Gang wechseln kann. Schnittstellen definieren bestimmte Methoden, implementieren sie aber nicht.

Eine *Schnittstelle* ist eine spezielle Klasse, die verschiedene Methoden definiert.

Eine Klasse kann eine Schnittstelle implementieren, was einem Vertrag entspricht, die in der Schnittstelle definierten Methoden bereitzustellen (zu implementieren).

Schnittstellen und Klassen in der SAX-Distribution

Teile dieser Schnittstellen sind insbesondere für die *Parser-Entwickler* vorgesehen. Wir werden sie hier nicht beschreiben.

Andere dagegen sind für die *Anwendungsentwickler* gedacht:

- DocumentHandler
- ErrorHandler
- DTDHandler
- EntityResolver

Einfache XML-Anwendungen können unter Verwendung von `DocumentHandler` und gegebenenfalls `ErrorHandler` entwickelt werden.

Die folgende Auflistung beschreibt die wichtigsten Methoden der `DocumentHandler`-Schnittstelle:

- `startDocument()` – Zum Empfang von Benachrichtigungen über den Anfang eines Dokuments.
- `endDocument()` – Zum Empfang von Benachrichtigungen über das Ende eines Dokuments.
- `startElement(String name, AttributeList atts)` – Zum Empfang von Benachrichtigungen über den Anfang eines Elements, wie in Tabelle 15.2 gezeigt.
- `endElement(String name)` – Zum Empfang von Benachrichtigungen über das Ende eines Elements, wie in Tabelle 15.3 gezeigt.
- `characters(char ch[], int start, int length)` – Zum Empfang von Benachrichtigungen über Zeichendaten, wie in Tabelle 15.4 gezeigt.
- `processingInstruction (String target, String data)` – Zum Empfang von Benachrichtigungen über eine Verarbeitungsanweisung, wie in Tabelle 15.2 gezeigt.

Name	Inhalt	Typ
`name`	Elementname	String
`atts`	Die dem Element zugeordneten Attribute	`AttributeList`

Tabelle 15.2: Parameter von `startElement`

`AttributList` erlaubt Ihnen, eine Attributliste zu durchlaufen.

Name	Inhalt	Typ
`name`	Elementname	`String`

Tabelle 15.3: Parameter von `endElement`

Name	Inhalt	Typ
`ch`	Die Zeichen aus dem XML-Dokument	Array mit Zeichen
`start`	Anfangsposition im Array	Integer
`length`	Anzahl der zu lesenden Zeichen	Integer

Tabelle 15.4:

Name	Inhalt	Typ
`target`	Ziel der Verarbeitungsanweisung	String
`data`	Daten der Verarbeitungsanweisung	String

Tabelle 15.5:

Beachten Sie die Ähnlichkeit mit den Ereignis-Handlern von Omnimark.

Das SAX-Paket beinhaltet außerdem die Klasse `HandlerBase`, die Standardimplementierungen der vier oben aufgeführten Schnittstellen bietet.

Die Anwendungsentwickler haben damit zwei Möglichkeiten:

- Sie können eine Klasse definieren, die die benötigten Schnittstellen enthält.
- Sie können eine Unterklasse der Klasse `HandlerBase` anlegen – die Standardimplementierungen der Schnittstellen enthält.

Eine *Unterklasse* ermöglicht uns, die Variablen (Eigenschaften) und Methoden (Verhalten) von der Oberklasse zu erben, wobei die Methoden auch überschrieben werden können.

Eine *Unterklasse* ist eine spezielle Klasse.

Listing 15.20 zeigt, wie wir eine Unterklasse der Klasse `HandlerBase` anlegen.

Listing 15.21: Anlegen einer Unterklasse von *HandlerBase*

```
1:  public class OurHandler extends HandlerBase { //Unterklasse zu HandlerBase
2:
3:       public void startDocument ()
4:             {
5:   //unsere Implementierung der Methode überschreibt HandlerBase
6:   }
7:
8:   public void endDocument ()
9:              {
10:                      // unsere Implementierung der Methode überschreibt
HandlerBase
11:             }
12:
13:       public void startElement (String name, AttributeList atts)
14:             {
15:                      // unsere Implementierung der Methode überschreibt
HandlerBase
16:             }
17:
18:       public void endElement (String name)
19:             {
20:                      // unsere Implementierung der Methode überschreibt
HandlerBase
21:             }
22:
23:       public void characters (char ch[], int start, int length)
24:             {
25:                      // unsere Implementierung der Methode überschreibt
HandlerBase
26:             }
27:  }
```

Die in der Schnittstelle `DocumentHandler` definierten Methoden sind in den Zeilen 3, 8, 13, 18 und 23 implementiert.

Ein Beispiel

Die in Listing 15.23 gezeigte XML-Datei dient als Eingabe.

Listing 15.22: *musicians.xml* ***– Info über Musiker***

```
1:  <?xml version="1.0"?>
2:  <musicians>
3:        <musician>
4:               <name>Joey Baron
5:               </name>
6:               <instrument>drums
7:               </instrument>
8:               <NrOfRecordings>1
9:               </NrOfRecordings>
10:       </musician>
11:       <musician>
12:              <name>Bill Frisell
13:              </name>
14:              <instrument>guitar
15:              </instrument>
16:              <NrOfRecordings>3
17:              </NrOfRecordings>
18:       </musician>
19:       <musician>
20:              <name>Don Byron
21:              </name>
22:              <instrument>clarinet
23:              </instrument>
24:              <NrOfRecordings>2
25:              </NrOfRecordings>
26:       </musician>
27:       <musician>
28:              <name>Dave Douglas
29:              </name>
30:              <instrument>trumpet
31:              </instrument>
32:              <NrOfRecordings>1
33:              </NrOfRecordings>
34:       </musician>
35:  </musicians>
```

Die Klasse `OurHandler` ist in Listing 15.23 definiert.

***Listing 15.23:** OurHandler.java* **– konkrete Implementierungen von Ereignis-Handlern**

```
1:  import org.xml.sax.HandlerBase;
2:  import org.xml.sax.AttributeList;
3:  import java.io.*;
4:
5:  public class OurHandler extends HandlerBase { //Unterklasse von HandlerBase
6:
7:   private PrintWriter fout;
8:
9:      public OurHandler() throws IOException
10:           {
11:                   fout = new PrintWriter(new FileWriter("out.htm"));
12:                   //Objekt, das in die Datei "out.htm" geschrieben wird
13:           }
14:
15:     public void startDocument ()
16:           {
17:                   fout.println("<HTML>");
18:                   fout.println("<HEAD><TITLE>SAX example</TITLE></HEAD>");
19:                   fout.println("<BODY>");
20:           }
21:
22:     public void endDocument ()
23:           {
24:                   fout.println("</BODY></HTML>");
25:                   fout.close();
26:           }
27:
28:     public void startElement (String name, AttributeList atts)
29:                           {
30:                   if (name == "musicians")
31:                           fout.println("<TABLE BORDER='1'
➥CELLPADDING='5'>");
32:                   else if (name == "musician")
33:                           fout.println("<TR>");
34:                   else
35:                           fout.println("<TD>");
36:           }
37:
38:     public void endElement  (String name)
39:                   {
40:                   if (name == "musicians")
41:                           fout.println("</TABLE>");
42:                   else if (name == "musician")
```

```
43:                             fout.println("</TR>");
44:                     else
45:                             fout.println("</TD>");
46:             }
47:
48:     public void characters (char ch[], int start, int length)
49:             {
50:                     for (int i=start; i < start+length; i++)
51:                             fout.print(ch[i]);
52:              }
53:     }
```

Als erstes importieren wir Klassen und Schnittstellen, die wir für unsere Arbeit brauchen (Zeilen 1 bis 3). Anschließend legen wir eine Unterklasse von `HandlerBase` an (Zeile 5).

In Zeile 11 legen wir ein Objekt an, das in die Datei `out.htm` geschrieben wird.

Beginnend in Zeile 15 sehen Sie die verschiedenen Ereignis-Handler:

- Beim Auftreten des Ereignisses `start of the musicians.xml document` wird der in den Zeilen 17, 18 und 19 definierte HTML-Inhalt in die Datei `out.htm` geschrieben.
- Beim Auftreten des Ereignisses `end of the musicians.xml document` wird der in Zeile 24 definierte HTML-Inhalt in die Datei `out.htm` geschrieben.
- Beim Auftreten des Ereignisses `start of an element` erfolgen zusätzliche Überprüfungen (Zeilen 30–35).
- Ist der Name des Elements gleich `musicians`, wird ein Tabellen-Start-Tag ausgegeben.
  - Ist der Name des Elements `musician`, wird ein Tabellenzeilen-Start-Tag `<TR>` ausgegeben.
  - Hat das Element einen anderen Namen, wird ein Tabellenzellen-Start-Tag `<TD>` ausgegeben.
  - Beim Auftreten des Ereignisses `end of an element` erfolgt dieselbe Überprüfung.
- Beim Auftreten des Ereignisses `characters encountered` werden diese Zeichendaten ausgegeben.

Außerdem brauchen wir eine Anwendung (siehe Listing 15.24), die das Dokument mit Hilfe unseres Handlers parst.

Listing 15.24:** `convert.java` **– das Programm, das unsere** `OurHandler`**-Klasse einsetzt.

```
1:  import org.xml.sax.Parser;
2:  import org.xml.sax.DocumentHandler;
3:  import org.xml.sax.helpers.ParserFactory;
4:
5:  public class convert {
6:        static final String parserClass = "com.ibm.xml.parser.SAXDriver";
7:        //Verwendung des XML-Parsers von IBM
8:        //static final String parserClass =
➥"com.datachannel.xml.sax.SAXDriver";
9:        //falls wir den DXP-Parser verwenden wollen
10:       static final String xmlfile="c:\\xmlex\\musicians.xml";
11:
12:       public static void main (String args[]) throws Exception
13:             {
14:                   Parser parser;  //Variablenerkennung
15:                   //parser verweist auf ein DocumentHandler-Objekt
16:                   DocumentHandler handler; //Variablendeklaration
17:                   //handler verweist auf ein DocumentHandler-Objekt
18:                   parser = ParserFactory.makeParser(parserClass);
19:                   //Ein Parser-Objekt wird erzeugt, indem der ParserFactory
ein Klassenname übergeben wird
20:                   handler = new OurHandler();
21:                   //Das neue Objekt wird erzeugt und initialisiert
22:
23:                   parser.setDocumentHandler(handler);
24:                   //Der Handler wird beim Parser registriert
25:                   parser.parse(xmlfile);
26:                   //unsere XML-Datei wird geparst
27:
28:            }
29:     }
```

Wir deklarieren eine Variable mit dem Namen `parserClass` vom Typ `String`. Wir weisen dieser Variablen einen Wert zu – den vollständigen Klassennamen des SAX-

Treibers des verwendeten Parsers: `com.ibm.xml.parser.SAXdriver` (Zeile 6). Dieser Wert kann beispielsweise durch `com.datachannel.xml.sax.SAXdriver` (Zeile 10) oder `com.microstar.xml.SAXdriver` ersetzt werden.

Im restlichen Code werden Variablen deklariert und instantiiert (Zeilen 14 – 21).

Schließlich (Zeile 23) wird der Handler `OurHandler` beim Parser registriert und anschließend unsere XML-Datei `musicians.xml` geparst.

In Abbildung 15.4 sehen Sie beide Dateien, `OurHandler.java` und `convert.java`, erstellt und kompiliert in der KAWA IDE (`http://www.tek-tools.com/kawa/`).

Ausführung der Konvertierung

Nachdem wir wissen, worum es sich bei der SAX-Distribution handelt und wie wir die SAX-Schnittstellen implementieren, wollen wir die einzelnen Komponenten zusammensetzen.

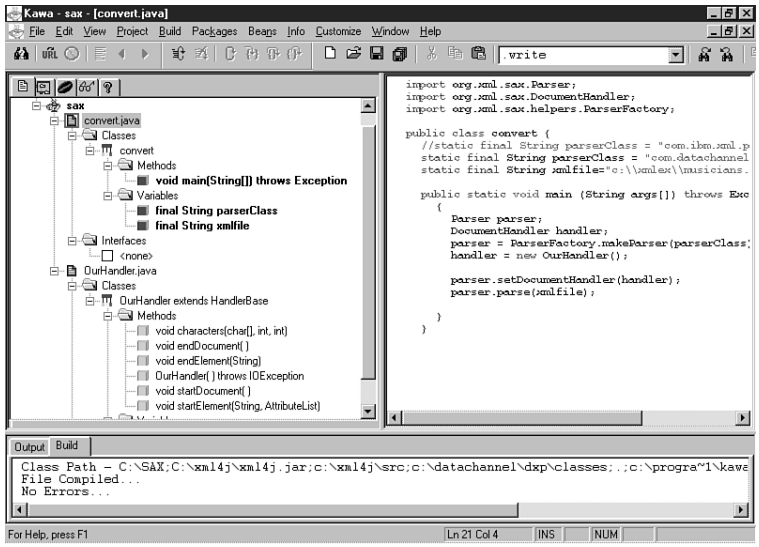


Abbildung 15.4: Unsere Java-Entwicklungsumgebung.

Wir brauchen die folgenden Komponenten:

- Einen SAX-fähigen Parser
- Die SAX-Distribution
- Die Ereignis-Handler

Wir haben auf unserem System zwei SAX-fähige Parser installiert (siehe Kapitel 5 und Kapitel 9):

- xml4j von IBM
- DXP von Datachannel

Beachten Sie den vollständigen Klassennamen ihres SAX-Treibers, der in unserer `convert`-Klasse verwendet wird:

- com.ibm.xml.parser.SAXDriver für xml4j
- com.datachannel.xml.sax.SAXDriver für DXP

Gegebenenfalls setzen Sie diesen Klassennamen auf den von Ihnen verwendeten Parser. Andere Änderungen im Code sind nicht erforderlich.

Aus Kapitel 14 wissen Sie, daß dies eine der Hauptaufgaben der SAX-Initiative war: Man wollte, daß die Parser gewechselt werden können, ohne daß Code umgeschrieben werden muß.

Wir müssen die SAX-Distribution (`saxjava-1_0-zip`) auf unserem System entzippen. Außerdem brauchen wir die beiden kompilierten Klassen: `convert.class` (das Hauptprogramm) und `OurHandler.class` (die die Ereignis-Handler implementiert).

Jetzt stellen wir sicher, daß alle erforderlichen Klassen zusammen mit ihren Pfaden in der `CLASSPATH`-Variablen aufgeführt sind.

In unserem Fall sieht `CLASSPATH` wie folgt aus:

```
C:\SAX;C:\xml4j\xml4j.jar;c:\datachannel\dxp\classes;.;
c:\progra~1\kawa30\classes.zip;c:\jdk1.1.5\lib\classes.zip
```

Anschließend können wir den Befehl von der Befehlszeile aus ausführen:

```
java.exe convert
```

Damit legen wir die Datei `out.htm` an, die unsere Musiker in einer HTML-Tabelle anzeigt.

Weitere Implementierungen

Anwender, die Kenntnisse der Sprache Python mitbringen, sollten sich die SAX-Implementierung von Lars Marius Garshol ansehen, die Sie unter der folgenden URL finden: `http://www.stud.ifi.uio.no/~larsga/download/python/xml/saxlib.html`.

Wie es mit SAX weitergeht

SAXON ist eine Java-Klassenbibliothek, die auf SAX aufbaut und zusätzliche Dienste bietet:

- Sie erlaubt Ihnen, separate Handler für jeden Elementtyp bereitzustellen.
- Sie liefert Kontextinformationen (Eltern, Großeltern usw.).
- Sie erlaubt Ihnen, einzelnen Elementen oder Typen unterschiedliche Ausgabeströme zuzuordnen.

Die Standard-Element-Handler unterstützen die folgenden Dinge:

- Unverändertes Kopieren eines Elements in die Ausgabe
- Überspringen eines Elements
- Ersetzen von Start- und Ende-Tags

Offensichtlich wurde diese Bibliothek für Konvertierungen von XML in XML oder HTML entwickelt.

Einen Überblick über SAX-basierte Anwendungen finden Sie unter der folgenden URL:

`http://www.megginson.com/SAX/index.html`.

Zusammenfassung

In diesem Kapitel haben wir die beiden Implementierungen ereignisgesteuerter Verarbeitung beschrieben: Omnimark und SAX. Omnimark ist seit langer Zeit der Markführer für SGML-Konvertierungen, während SAX das Ergebnis einer Zusammenarbeit auf der XML-DEV-Mailingliste ist.

F&A

F Soll ich Omnimark oder SAX verwenden?

A *Das alteingesessene Omnimark bietet alle Schikanen für die Anforderungen Ihrer Verarbeitungsaufgaben. Es ist schnell, stabil und sehr zuverlässig. Andererseits handelt es sich dabei nicht um eine Standardsprache, und es gibt keine Garantie, daß Omnimark Technologies irgendwann eine kostenlose Version bereitstellt.*

F Welchen Programmierstil sollte ich verfolgen (ereignisgesteuert oder baumbasiert)?

A *Die ereignisgesteuerte Programmierung ist einfach und schnell, aber die baumbasierte Programmierung ist leistungsfähiger, weil der Baum im Speicher aufgebaut wird und beliebig durchlaufen werden kann. Darüber hinaus hat die W3C das baumbasierte API standardisiert, das sogenannte Document Object Model. Dieses Modell wird voraussichtlich von allen Konkurrenten in diesem Bereich implementiert. Wir empfehlen Ihnen, Ihre Aufmerksamkeit insbesondere dieser Schnittstelle zu widmen.*

Übungen

1. Schreiben Sie den Regelkopf für Omnimark, der die folgenden Ereignisse auffängt:

- Das Element `note`
- Das Element `tool`, das Enkelkind des Elements `procedure` ist
- Das Element `result`, das nach einer `action` kommt
- Das Element `listitem`, das als erstes Element in einer Folge von `listitems` steht, aber nicht das erste Element im Elternelement ist.

2. Schreiben Sie für SAX einen Ereignis-Handler, der für die Eingabe der Datei `musicians.xml` den folgenden Bericht ausgibt:

```
1:  Start document
2:  Start element: musicians
3:  Start element: musician
4:  Start element: name
5:  Characters: Joey Baron
6:  End element: name
7:  Start element: instrument
8:  Characters: drums
9:  End element: instrument
10:  Start element: NrOfRecordings
11:  Characters: 1
12:  End element: NrOfRecordings
13:  End element: musician
14:  ...
15:  End element: musicians
16:  End document
```

Verwenden Sie dazu die folgende Schablone:

```
1:  import org.xml.sax.HandlerBase;
2:  import org.xml.sax.AttributeList;
3:
4:  public class YourHandler extends HandlerBase {
5:
6:       public void startDocument ()
7:            {
8:                     System.out.println("XXXXX");
9:            }
10:
11:      public void endDocument ()
12:           {
13:                    System.out.println("XXXXX");
14:           }
15:
16:      public void startElement (String name, AttributeList atts)
17:           {
18:                    System.out.println("XXXXX");
19:           }
20:
21:      public void endElement (String name)
22:           {
23:                    System.out.println("XXXXX");
24:           }
25:
26:      public void characters (char ch[], int start, int length)
27:           {
28:                    System.out.println("XXXXX");
29:           }
30:  }
```

16

Programmierung mit dem Document Object Model

Woche 3

In diesem Kapitel geht es um die folgenden Themen:

- Der Hintergrund des Document Object Model (DOM)
- Die W3C-Empfehlung vom 1. Oktober 1998
- Ein Beispiel für die Verwendung des DOM
- DOM-Implementierungen

Hintergrund

Das Document Object Model ist ein Modell, das davon ausgeht, daß ein Dokument Objekte enthält, die Eigenschaften (Attribute) und Methoden besitzen, so daß sie manipuliert werden können.

Das bedeutet, das DOM ermöglicht Ihnen die folgenden Dinge:

- Elemente hinzufügen, löschen und ändern
- Den Inhalt von Elementen ändern
- Attribute hinzufügen, löschen und ändern

Sowohl Netscape als auch Microsoft bieten in der Version 4 ihrer Browser die Möglichkeit, HTML-Seiten dynamisch zu ändern. Diese Funktionalität basierte auf einem HTML-Document Object Model. Leider waren die zugehörigen Implementierungen höchst inkompatibel.

Man brauchte Standards; deshalb wurde im W3C die Arbeitsgruppe DOM gegründet. Ihre Arbeit führte zu der W3C-Empfehlung vom 1. Oktober 1998. Sie finden die Arbeit des Konsortiums unter `http://www.w3.org/TR/WD-DOM/`.

Diese DOM-Spezifikation, Level 1, definiert die Schnittstellen nur generisch, wozu sie die *IDL* (*Interface Definition Language*) einsetzt. Es bleibt den Entwicklern überlassen, das DOM für eine bestimmte Sprache zu implementieren (JavaScript, VBScript, Java, C++, Python, Perl usw.).

Die Spezifikation

Jetzt wollen wir die Spezifikation genauer betrachten. Zunächst untersuchen wir ihren Aufbau und erklären dann die wichtigsten Objekte/Schnittstellen.

Struktur

Die W3C-Empfehlung besteht aus zwei Teilen – dem Kern (Core) und HTML.

Der HTML-Anteil des DOM ist für unsere XML-Zwecke nicht relevant und wird hier nicht weiter besprochen.

Der Kern-Teil des W3C stellt grundlegende Schnittstellen bereit, die beliebige strukturierte Dokumente (XML und HTML) darstellen können, ebenso wie erweiterte Schnittstellen, die für XML-Dokumente benötigt werden.

Eine genauere Beschreibung der Schnittstellen in Hinblick auf die OO-Konzepte finden Sie in Kapitel 15.

Die Schnittstellen

Weil dieses Modell aus dem objektorientierten Paradigma stammt, sind die Schnittstellen als Objekte definiert.

Die grundlegenden Objekte sind:

- DOMException
- DOMImplementation
- DocumentFragment
- Document
- Node
- NodeList
- NamedNodeMap
- CharacterData
- Attr
- Element
- Text
- Comment

Und hier die erweiterten Schnittstellen; das sind Objekte, die insbesondere für XML-Dokumente vorgesehen sind:

- CDATASection
- DocumentType
- Notation
- Entity
- EntityReference
- ProcessingInstruction

Beziehungen zwischen den Schnittstellen

Diese Objekte weisen Beziehungen zueinander auf, und einige erben Eigenschaften und Methoden von anderen Objekten.

Diese Beziehungen sind in Abbildung 16.1 dargestellt. Objekte unten im Baum erben von den darüberliegenden Objekten.

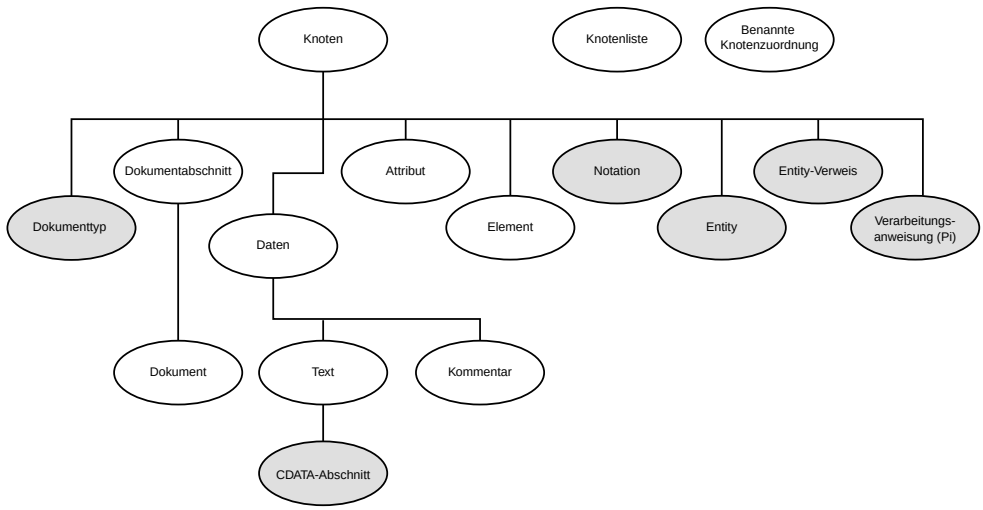


Abbildung 16.1:
Vererbungsbeziehungen. Die grau dargestellten Elemente gehören zu den erweiterten Schnittstellen.

Neben diesem objektorientierten Ansatz mit seiner Vererbungshierarchie ist auch eine vereinfachtere Arbeitsweise möglich. Sie nehmen einfach alle Manipulationen über die `Node`-Schnittstelle vor (siehe nächster Abschnitt).

Jetzt wollen wir die wichtigsten Objekte /Schnittstellen genauer betrachten.

Das Node-Objekt

Das `Node`-Objekt ist der wichtigste Datentyp im gesamten Modell. Es repräsentiert einen Knoten im XML-Dokumentbaum. Es gibt unterschiedliche Knotentypen, die in Tabelle 16.1 aufgelistet sind.

Typ	DOM-Code
`Element`	1
`Attribute`	2
`Text`	3
`CDATA Section`	4
`Entity Reference`	5
`Entity`	6
`Processing Instruction`	7
`Comment`	8
`Document`	9
`Document Type`	10
`Document Fragment`	11
`Notation`	12

Tabelle 16.1: Knotentypen

Das Node-Objekt/die Schnittstelle stellt Ihnen die allgemeinen Eigenschaften und Methoden zum Ermitteln und Setzen der Knoteninformation bereit. Andere, spezialisiertere Schnittstellen/Objekte (Element, Attribut usw.) enthalten möglicherweise zusätzliche und bequemere Mechanismen für diese Aufgabenstellung.

Zunächst wollen wir die allgemeinen Eigenschaften des `Node`-Objekts betrachten, wie in Tabelle 16.2 gezeigt.

Name	Wert
`nodeName`	Name (String) des Knotens (z.B. `tag`, `attributname`)
`nodeValue`	Wert (String) des Knotens (z.B. `attributevalue`, `text`)
`nodeType`	Der Knoten, der den Objekttyp darstellt (1 für Element, 2 für Attribut, 3 für Text usw.).

Tabelle 16.2: Attribute/Eigenschaften des `Node`-Objekts

Name	Wert
parentNode	Der Elternknoten für den aktuellen Knoten.
childNodes	Ein Nodelist-Objekt, das alle Kinder dieses Knotens auflistet.
firstChild	Der erste Kindknoten eines Knotens.
lastChild	Der letzte Kindknoten eines Knotens.
previousSibling	Der Knoten unmittelbar vor dem aktuellen Knoten.
nextSibling	Der Knoten unmittelbar nach dem aktuellen Knoten.
attributes	Eine NamedNodeMap mit den Attributen.

Tabelle 16.2: Attribute/Eigenschaften des Node-Objekts

Listing 16.1 ist ein JavaScript-Beispiel, in dem nodeName, nodeValue und nodeType verwendet werden.

Listing 16.1: Verwendung der Attribute nodeType, nodeValue und nodeName eines Node-Objekts

```
1:  //var node enthält ein Node-Objekt
2:  if (node.nodeType == 2) //falls Attribut
3:        return node.nodeValue;// Attributwert zurückgeben
4:  if (node.nodeType == 1) // falls Element
5:        return node.nodeName; // Tag-Namen zurückgeben
```

Wenn ein Knoten im Baum den Typ attribute hat, wird der Wert (attributevalue) zurückgegeben.

Hat dieser Knoten den Typ element, wird der Tag-Name zurückgegeben.

Listing 16.2 zeigt ein weiteres JavaScript-Beispiel.

Listing 16.2: Verwendung der Attribute previousSibling, nextSibling und parentNode eines Node-Objekts

```
1:  //var node holds a Node Object
2:  var previous = node.previousSibling;
3:  var next = node.nextSibling;
4:  var parent = node.parentNode;
5:
6:  return "Hi, I'm " + node.nodeName + " .";
```

```
7:
8:  if (parent != null)
9:       return "My father is " + parent.nodeName + " .";
10:  if (previous != null)
11:       return "My older brother is " + previous.nodeName + " .";
12:  if (next != null)
13:       return "My younger brother is " + next.nodeName + " .";
```

Der vorhergehende und nächste Geschwisterknoten und der Elternknoten eines bestimmten Knotens werden in drei Variablen abgelegt (Zeilen 2 bis 4). Anschließend wird der Name des Knotens zurückgegeben, gefolgt vom Namen des Elternknotens und gegebenenfalls des nächsten Geschwisterknotens, falls es einen solchen gibt.

Listing 16.3 zeigt das JavaScript-Beispiel, das das `childNodes`-Attribut eines `Node`-Objekts verwendet.

Listing 16.3: Verwendung des `childNodes`-Attributs eines `Node`-Objekts

```
1:  //var node holds a Node Object
2:  var children = node.childNodes; //gibt eine NodeList (cf.infra) zurück
```

Tabelle 16.3 bietet einen Überblick über die Methoden des `Node`-Objekts.

Name	Rückgabe
`insertBefore(newChild,refChild)`	Der einzufügende Knoten, wobei `newChild` der einzufügende Knoten ist, `refChild` der Knoten, vor dem er eingefügt werden soll.
`replaceChild(newChild,oldChild)`	Der zu ersetzende Knoten, wobei `newChild` der Knoten ist, der den Knoten `oldChild` ersetzt.
`removeChild(oldChild)`	Der entfernte Knoten.
`appendChild(newChild)`	Ein Knoten, wobei `newChild` der Knoten ist, der am Ende der Kindknotenliste eingefügt wird.
`hasChildNodes()`	`true` oder `false`, abhängig davon, ob der Knoten Kindknoten besitzt.
`cloneNode(deep)`	Ein Duplikat des Knotens. Ist der Parameter `deep` auf `true` gesetzt, erfolgt das Duplizieren rekursiv für alle im Unterbaum enthaltenen Knoten.

Tabelle 16.3: Node-Objektmethoden

Listing 16.4 zeigt ein weiteres JavaScript-Beispiel.

Listing 16.4: Verwendung der Methoden eines `Node`-Objekts

```
1:  //var node enthält ein Node-Objekt
2:  //der Knoten newElement wurde mit der Methode createElement des
3:  // Document-Objekts erzeugt (cf. infra)
4:  node.insertBefore(newElement,node.lastChild);
5:  //Einfügen des neuen Elements vor dem letzten Kindelement
6:  node.replaceChild(newElement,node.firstChild);
7:  //Ersetzen des ersten Kindelements durch das neue Element
```

In diesem Codeabschnitt setzen wir voraus, daß wir bereits eine Variable namens `node` haben, die ein `Node`-Objekt enthält, und daß wir mit Hilfe der Methode `createElement` des `Document`-Objekts einen neuen Knoten angelegt haben, `newElement`, wie später noch erklärt wird.

Zeile 4 fügt den Knoten `newElement` vor dem letzten Kindknoten mit dem Namen `node` ein.

Zeile 6 ersetzt das erste Kindelement des Knotens `node` durch das neue Element, `newElement`.

NodeList-Objekt/Schnittstelle

Das `NodeList`-Objekt/die Schnittstelle ist eine sortierte Auflistung von Knoten. Sortiert bedeutet, daß die Knoten eine bestimmte Position in der Auflistung haben.

Die `childNodes`-Eigenschaft des `Node`-Objekts gibt eine `NodeList` zurück, wie in Listing 16.4 gezeigt.

Listing 16.5: Eine Eigenschaft eines Objekts, die ein `NodeList`-Objekt zurückgibt

```
1:  //var docroot enthält das Wurzelelement des Dokuments
2:  var children = docRoot.childNodes;
```

Tabelle 16.4 bietet einen Überblick über die Eigenschaften des `NodeList`-Objekts.

Name	Wert
`length`	Die Knotenanzahl in der `NodeList`.

Tabelle 16.4: `NodeList`-Eigenschaften

Listing 16.6 zeigt ein JavaScript-Beispiel, das diese Eigenschaft nutzt.

Listing 16.6: Die Länge einer `NodeList` wird ermittelt

```
1:  //children enthält das NodeList-Objekt mit den Kindknoten
2:  var nrofchildren = children.length;
```

Tabelle 16.5 bietet einen Überblick über die Methoden des `NodeList`-Objekts.

Name	Rückgabe
`item(n)`	Das n-te Element (Knoten) der Auflistung.

Tabelle 16.5: `NodeList`-Methoden

Listing 16.7 zeigt ein JavaScript-Beispiel, das diese Methode einsetzt.

Listing 16.7: Verwendung des n-ten Elements einer `NodeList`

```
1:  //children enthält das NodeList-Objekt mit den Kindknoten
2:  var i;
3:  for (i=0, i < children.length, i++) //Schleife über alle Kindknoten
4:       return children.item(i).nodeName; //Namen zurückgeben
```

Mit Hilfe der Variablen `I` durchlaufen wir alle Elemente einer `NodeList`, die die Kindknotenliste enthält.

In Zeile 4 geben wir den Namen für die einzelnen Kinder zurück (den Tag-Namen, falls es sich bei dem Kind um ein Element handelt, den Attributnamen, falls es sich um ein Attribut handelt, usw.).

Das NamedNodeMap-Objekt

Das `NamedNodeMap`-Objekt ist ebenfalls eine Auflistung von Knoten, mit dem Unterschied, daß der Zugriff auf diese Auflistung über den Namen erfolgt.

Tabelle 16.6 zeigt die Eigenschaften des `NamedNodeMap`-Objekts.

Name	Wert
`length`	Die Anzahl der Knoten in der `NamedNodeMap`.

Tabelle 16.6: `NameNodeMap`-Eigenschaften

Listing 16.8 zeigt ein JavaScript-Beispiel, das diese Eigenschaft verwendet.

Listing 16.8: Hier wird die Länge einer `NamedNodeMap` ermittelt

```
1:  //var node enthält ein Node-Objekt
2:  if (node.attributes != null)
3:  //Prüfen, ob der Knoten Attribute hat
4:       return node.attributes.length;
5:  //Gibt die Anzahl der Attribute zurück
```

Tabelle 16.7 zeigt die Methoden des `NamedNodeMap`-Objekts.

Name	Rückgabe
`getNamedItem(name)`	Ein Knoten mit dem angegebenen Namen (String) oder Null.
`setNamedItem(arg)`	Nichts. Fügt der `NamedNodeMap` den in `arg` angegebenen Knoten hinzu.
`removeNamedItem(name)`	Der entfernte Knoten mit dem angegebenen Namen (String).
`item(n)`	Das n-te Element der Auflistung.

Tabelle 16.7: `NamedNodeMap`-Methoden

Listing 16.9 zeigt ein JavaScript-Beispiel, das die Methode `getNamedItem()` verwendet.

Listing 16.9: Verwendung der Methode `getNamedItem()` eines/einer `NamedNodeMap`-Objekts/-Schnittstelle

```
1:  //var node enthält ein Node-Objekt
2:  return node.attributes.getNamedItem("unit");
3:  //Gibt den Knoten des Typs attribute mit dem Namen "unit" zurück
```

Ein `DocumentFragment` ist ein Objekt, das Dokumentabschnitte enthalten kann, um sie zu kopieren und einzufügen, neu anzuordnen usw. Es erbt von der Objektklasse `Node`.

Das Document-Objekt

Das `Document`-Objekt repräsentiert das vollständige XML-Dokument. Es erbt von der Objektklasse `Node`. Dies ist die einzige Schnittstelle, die Ihnen ermöglicht, andere Objekte anzulegen, die nur innerhalb von Dokumenten auftreten kann.

Tabelle 16.8 zeigt die Eigenschaften des `Document`-Objekts.

Name	Wert
documentElement	Das Wurzel-Element-Objekt des Dokuments
doctype	DocumentType-Objekt (in den erweiterten Schnittstellen definiert)

Tabelle 16.8: Attribute/Eigenschaften des Document-Objekts

Listing 16.10 zeigt ein JavaScript-Beispiel, das das Wurzelelement eines XML-Dokuments ermittelt.

Listing 16.10: Ermittlung des Wurzelelements eines XML-Dokuments

```
1:  //var xml enthält das XML-Dokument
2:  var docRoot = xml.documentElement;
3:  return "Das Wurzelelement ist " + docRoot.nodeName + " element.";
4:  //die Eigenschaft  nodeName gibt den Tag-Namen des Elements zurück
```

Tabelle 16.9 zeigt die Methoden des Document-Objekts.

Name	Rückgabe
createElement(tagName)	Ein neues Element-Objekt namens tagname (String).
createTextNode(data)	Einen neuen Textknoten mit den angegebenen Daten (String).
createComment(data)	Einen neuen Kommentar mit den angegebenen Daten (String).
getElementsByTagName(tagname)	Eine NodeList aller nachfolgenden Elemente mit dem vorgegebenen tagname (String).

Tabelle 16.9: Methoden des Document-Objekts

Listing 16.11 zeigt ein JavaScript-Beispiel für das Anlegen eines neuen Elements in einem XML-Dokument.

Listing 16.11: Ein Element anlegen

```
1:  //var xml enthält das XML-Dokument-Objekt
2:  var newElement = xml.createElement("remark");
```

Diese Methode macht den Knoten nicht zum Teil des Dokumentbaums. Für diesen Zweck werden die Methoden insertBefore oder appendChild des Node-Objekts aufgerufen.

Das Data-Objekt

Die CharacterData-Schnittstelle stellt Ihnen Eigenschaften und Methoden für den Zugriff und die Manipulation auf bzw. von Zeichendaten zur Verfügung. Es gibt keine direkte Entsprechung von Dokumentobjekten mit den Data-Objekten. Die Objekte Text, Comment und CDATASection erben davon.

Tabelle 16.10 zeigt die Eigenschaften des CharacterData-Objekts.

Name	Wert
data	Die Zeichendaten (String) des Knotens.
length	Die Anzahl der Zeichen.

Tabelle 16.10: Attribute/Eigenschaften des CharacterData-Objekts

Seine Methoden sind in Tabelle 16.11 aufgelistet.

Name	Rückgabe
substringData(offset,count)	Der angegebene Teilstring, wobei start die Anfangsposition und count die Anzahl der Zeichen angibt.
appendData(arg)	Nichts. Fügt den String arg am Ende der Daten im Objekt ein.
insertData(offset,arg)	Nichts. Fügt den String arg an der angegebenen Position ein.
deleteData(offset,count)	Nichts. Entfernt einen Zeichenbereich, der durch offset und count angegeben wird.
replaceData(offset,count,arg)	Nichts. Ersetzt die Zeichen, beginnend an offset mit der Länge count durch den in arg angegebenen String.

Tabelle 16.11: Methoden des CharacterData-Objekts

Listing 16.12 zeigt ein Beispiel für die Methode appendData.

Listing 16.12: Text ermitteln und ändern

```
1:  //var node enthält ein Node-Objekt
2:  if (node.nodeType == 3) //falls Text
3:        var content = node.nodeValue;
4:        content.appendData("  !");
```

Wenn die Variable einen Text-Knoten enthält, wird ihr Inhalt in der `content`-Variablen abgelegt. Diesem Inhalt wird ein Ausrufezeichen angefügt.

Die anderen Objekte

Im letzten Abschnitt wurden die wichtigsten Schnittstellen beschrieben. Die spezielleren Schnittstellen, wie beispielsweise das `Element`-Objekt, erweitern das allgemeinere `Node`-Objekt und führen einige spezifischere Eigenschaften und Methoden ein, die Ihnen erlauben, bequemer und spezifischer auf das Objekt zuzugreifen und es zu manipulieren.

Das `Element`-Objekt bietet die Möglichkeiten (Eigenschaften und Methoden), auf Tag-Namen, Attributnamen und Attributwerte zuzugreifen und diese zu verändern.

Weitere Informationen finden Sie unter der URL `http://www.w3.org/TR/WD-DOM/`.

Beispiel für die Verwendung des DOM

Wir wollen die Preise der in Listing 16.13 gezeigten XML-Datei in Kanadische Dollar umwandeln.

Listing 16.13: Die umzuwandelnden Preise

```
1:  <?xml version="1.0"?>
2:  <books>
3:       <book id="TY1">
4:               <title>Sams Teach Yourself C++ in 21 Days, Second Edition
5:               </title>
6:               <author>Jesse Liberty
7:               </author>
8:               <price unit="USA">29.99
9:               </price>
10:      </book>
11:         <book id="MJ">
12:             <title>Maximum Java 1.1
13:             </title>
14:             <author>Glenn Vanderburg
15:             </author>
16:             <price unit="USA">49.99
```

```
17:             </price>
18:      </book>
19:      <book id="JS">
20:             <title>JavaScript Unleashed, Second Edition
21:             </title>
22:             <author>Richard Wagner
23:             </author>
24:             <price unit="USA">49.99
25:             </price>
26:      </book>
27:      <book ID="TY2">
28:             <title>Sams Teach Yourself Visual C++ 5 in 21 Days, Fourth
➥Edition
29:             </title>
30:             <author>Nathan Gurewich
31:             </author>
32:             <author>Ori Gurewich
33:             </author>
34:             <price unit="USA">35.00
35:             </price>
36:      </book>
37:  </books>
```

Listing 16.14 zeigt einen Abschnitt des JavaScript-Codes, in dem das DOM verwendet wird, um die US-Preise in Kanadische Dollar umzuwandeln.

Listing 16.14: JavaScript zur Umwandlung in Kanadische Dollar

```
1:  //var xml enthält das Document-Objekt
2:  var priceElements = xml.getElementsByTagName("price");
3:  //Gibt eine NodeList aller Elemente mit dem Tag price zurück
4:  var i;
5:  for (i=0;i < priceElements.size;i++) { // Schleife über alle price-Elemente
6:       var USAprice = parseFloat(priceElements.item(i)
➥.firstChild.nodeValue);
7:       //die firstChild-Eigenschaft gibt den Text-Knoten zurück
8:       //wir nehmen den nodeValue, also den eigentlichen Text und wandeln
9:       //ihn in eine Zahl (Floating) um
10:      priceElements.item(i).setAttribute("unit","Canadian dollar");
11:      //wir setzen das Attribut "unit" auf "Canadian dollar"
12:      var convertedprice = USAprice * 1.4;
13:      //wir wandeln den Preis in Kanadische Dollar um
14:      var newprice = xml.createTextNode(convertedprice);
15:      //wir erzeugen einen neuen Text-Knoten mit dem neuen Preis
16:      priceElements.item(i).replaceChild(newprice,
➥priceElements.item(i).firstChild);
```

```
17:      //wir ersetzen den alten Text-Knoten durch den neuen
18:      }
19:
```

In Zeile 2 von Listing 16.14 erzeugen wir ein `NodeList`-Objekt, das alle Elemente unseres XML-Dokuments mit dem Tagnamen `price` enthält.

Für jedes dieser Elemente (Schleife in Zeile 5) nehmen wir den ersten Kindknoten (`firstChild`-Eigenschaft), einen Textknoten. Wir ermitteln den Wert dieses Textknotens (`nodeValue`-Eigenschaft), den Inhalt. Dieser Inhalt wird in der Funktion `ParseFloat()` in JavaScirpt in eine Zahl umgewandelt. Die resultierende Zahl wird verwendet, um den Preis in Kanadischen Dollar zu berechnen (Zeile 12).

In Zeile 14 erzeugen wir mit Hilfe der Methode `createTextnode()` des `Document`-Objekts einen neuen Textknoten.

Dieser Textknoten ersetzt den alten Textknoten (Zeile 16).

Implementierungen des DOM

Der Microsoft Internet Explorer 5 Beta 2 bietet vollständige DOM-Unterstützung, die konform mit der W3C-DOM-Empfehlung ist.

Dasselbe gilt für Mozilla/Gecko. Man erwartet, daß auch bald die Hersteller von Bearbeitungswerkzeugen, wie etwa SoftQuad, das DOM-API in ihre Produkte aufnehmen, ebenso wie die Hersteller von Repositories, wie etwa POET, Chriystal, Oracle, Texcel usw.

Auf der Parser-Seite implementierten DataChannel, IBM und Sun das DOM-API bereits in ihren Java-XML-Parsern.

Die Zukunft des DOM

Heute ist DOM Level 1 auf die Methoden für die Darstellung und Manipulation von Struktur und Inhalt begrenzt.

Fehlende Komponenten, die in zukünftigen DOM-Levels berücksichtigt werden sollten, sind:

- Die Schnittstellen für interne und externe Teilmengen
- Die Möglichkeit einer Analyse
- Die Möglichkeit, die Darstellung durch Stylesheets zu steuern
- Die Möglichkeit einer Zugriffssteuerung

Viele Anbieter von Werkzeugen werden dieses API implementieren. Trotz unterschiedlicher Sprachen (EcmaScript, VBScript, Java usw.) werden Sie es immer mit denselben Objekten mit denselben bekannten Eigenschaften und Methoden zu tun haben. Man muß das DOM also kennen.

Zusammenfassung

In diesem Kapitel ging es um Level 1 des DOM:

- Was ist das DOM?
- Was können Sie mit dem DOM machen?
- Warum wurde das DOM entwickelt?
- Die Schnittstellen (Objekte) im DOM
- Ein kurzes Beispiel
- Implementierungen des DOM
- Was die Zukunft bringt

F&A

F Ich bin nicht sicher, ob mein Code den richtigen Knoten auswählt. Gibt es ein Utility, das mir hilft?

A *Auf der Web-Site von Microsoft finden Sie den Site Builder Workshop in den XML-Demos, den XML Tree Viewer. Dies ist eine Webseite für den IE5, wo Sie eine Baumansicht Ihrer XML-Dateien anzeigen können. In einem Eingabefeld wählen Sie die Knoten aus, die Sie suchen. Das Ergebnis Ihrer Eingabe wird im Baum angezeigt.*

F Ich kann den Textinhalt eines Knotens nicht ermitteln.

A *Ermitteln Sie zuerst den eigentlichen Textknoten. Verwenden Sie die* `nodeValue`*-Eigenschaft dieses Textknotens, um den Inhalt zu erhalten.*

F Ich habe in meinem Dokument ein neues Element angelegt, aber es erscheint nicht.

A *Sie müssen das neue Element nicht nur anlegen, sondern es explizit in Ihr Dokument einfügen. Weitere Informationen finden Sie in den Abschnitten zu den Methoden* `insertBefore` *oder* `appendChild`*.*

Übungen

Wir verwenden das JavaScript, das im Internet Explorer 4 benutzt wurde, um eine XML-Datei zu konvertieren.

Das Skript in Listing 16.15 wurde mit einer veralteten Syntax des DOM geschrieben. Schreiben Sie es so um, daß es der neuen DOM-Spezifikation entspricht.

Listing 16.15: Ein zu aktualisierendes Skript

```
1:  <SCRIPT LANGUAGE="JScript" FOR="window" EVENT="onload">
2:    document.write("<HTML><HEAD><TITLE>My favorite
➥musicians</TITLE></HEAD>\n");
3:    document.write("<BODY><H2>My favorite musicians</H2><HR>\n")     ;
4:    var xml = new ActiveXObject("msxml");
5:    xml.URL = "file:///c¦/xmlex/musicians.xml";
6:    var docroot = xml.root;
7:    output_doc(docroot);
8:
9:    function traverse(elem)
10:       {var i;
11:            if (elem.children != null)
12:                 {
13:                 for (i=0; i < elem.children.length; i++)
14:                 output_doc(elem.children.item(i));
15:                    }
16:       }
17:
18:    function output_doc(elem)
19:    {
20:                 if (elem.type == 0)
21:                    {
22:                            if (elem.tagName == "MUSICIANS")
23:                                    {
24:                                           document.write("<TABLE
➥BORDER='1' CELLPADDING='5'>");
25:                                           traverse(elem);
```

```
26:                                         document.write("</TABLE>");
27:                                   }
28:                              else if (elem.tagName == "MUSICIAN")
29:                                   {
30:                                         document.write("<TR>");
31:                                         traverse(elem);
32:                                         document.write("</TR>");
33:                                   }
34:                              else
35:                                   {
36:                                         document.write("<TD>");
37:                                         traverse(elem);
38:                                         document.write("</TD>");
39:                                   }
40:
41:                    }
42:               else if (elem.type==1)
43:                              document.write(elem.text);
44:               else
45:                              alert("Unknown type encountered");
46:                    }
47:  </SCRIPT>
```

Die Verwendung von Metadaten zur Beschreibung von XML-Daten

Woche 3

Eine der am dringendsten erwarteten Funktionen ist die Möglichkeit, Markup für Daten zu nutzen, beispielsweise für die Codierung von Datenbankinhalt (ein ewig unerfüllter Traum der SGML-Gemeinde).

Die XML-DTD ist die einzige Möglichkeit, das Datenmodell (das Schema) eines XML-Dokuments zu beschreiben. Weil die DTD so unzulänglich ist, gibt es bereits zahlreiche Initiativen zur Suche nach Alternativen. Die Experten sagen, daß es womöglich noch mehrere Jahre dauern wird, bis ein geeignetes Ersatzschema für XML-Dokumente gefunden ist. Glücklicherweise gibt es bereits einige vielversprechende Entwicklungen.

In diesem Kapitel erfahren Sie

- warum die XML-DTD für Datenanwendungen nicht geeignet ist.
- die Grundlagen der drei wichtigsten Entwicklungen, die das zukünftige XML-Schema darstellen könnten.

Welche Probleme gibt es mit den DTDs?

Sie wissen, daß die XML-DTD die Struktur der Elemente in einem XML-Dokument beschreibt und daß das Dokument anhand der DTD daraufhin ausgewertet werden kann, ob es der dort vorgegebenen Struktur entspricht. Diese Möglichkeit, den Dokumentinhalt auszuwerten, verschafft XML einen Vorteil gegenüber anderen Methoden der Textauszeichnung. Beispielsweise gibt es keine Möglichkeit, das Markup von TeX-Code (einer Computer-Satzsprache, die unter Mathematikern und anderen Wissenschaftlern immer noch sehr verbreitet ist) oder TROFF-Code (das ist die primitive Markup-Sprache zur Formatierung von Online-Hilfen mit dem Unix-Utility `man`) zu überprüfen.

Die DTD bietet außerdem eine gewisse Kontrolle über Attribute – aber nur wenig Unterstützung bei der Auswertung der eigentlichen Daten in den Elementen. Sie kann prüfen, ob es ein DATA-Element gibt, aber nicht, ob der Inhalt dieses DATA-Elements reiner Unsinn ist, der unter keinem noch so großen Aufwand in sinnvolle Daten umgewandelt werden kann.

Wenn Sie XML verwenden wollen, um den Inhalt einer Datenbank zu codieren oder um Transaktionsdaten für Kreditkarten im Internet zu übertragen, gibt es Möglichkeiten, eine strengere Kontrolle über die Daten zu erzwingen. Die meisten dieser Systeme arbeiten nach dem alten GIGO-Prinzip »Garbage in – Garbage out« (wenn Müll eingegeben wird, wird auch Müll ausgegeben), und die einzige Methode, manuelle Aufräumarbeiten weitestgehend zu vermeiden, ist die Eingabe korrekter Daten. Sie wollen sicherstellen, daß ein numerischer Wert keinen Text enthält, daß ein Währungswert nur zwei Stellen hinter dem Komma hat usw. Die DTD scheitert kläglich – vielleicht, weil XML sich von SGML ableitet, das sehr dokumentorientiert ist.

Darüber hinaus müssen die Benutzer die Syntax der XML-Sprache lernen. Die XML-DTD ist jedoch kein XML-Dokument, noch nicht einmal ein SGML-Dokument – die DTD hat eine eigene Syntax. Man muß also nicht nur sehr viel mehr lernen, sondern es geht auch eine der wichtigsten Stärken von XML verloren – die Automatisierung.

Es gibt noch nicht viele XML-Werkzeuge – was sich aber vielleicht bald ändern wird. Die Situation ist vergleichbar mit der Entwicklung von HTML. Am Anfang gab es nur sehr wenige Werkzeuge dafür, und die meisten Benutzer setzten das ein, was es gerade gab. Häufig mußten sie ihren HTML-Code sogar manuell schreiben. Schnell wurden komplexe Werkzeuge angeboten, und wir haben heute einen Punkt erreicht, wo es ganz einfach geworden ist, HTML-Dokumente zu schreiben, ohne auch nur eine Ahnung davon zu haben, wie HTML überhaupt aussieht.

Die Geschichte der XML-Entwicklung wurde durch die Forderung der Automatisierung gesteuert – dem Wunsch, XML von Maschinen erzeugen zu lassen. XSL war nicht mehr länger eine komplexe Variante der Programmiersprache LISP, sondern eine vereinfachte Stilsprache, die die XML-Syntax verwendet. XLink und XPointer sind einen ähnlichen Weg gegangen – zuvor waren es Varianten von SGML und HyTime, und jetzt sind sie vereinfachte Sprachen, die ebenfalls die XML-Syntax verwenden. Das Ergebnis ist die realistische Perspektive, daß es möglich wird, XML-Code von Maschinen erzeugen zu lassen ebenso wie den Code für seine Darstellung und die Verknüpfung mit anderen Dokumenten.

Der allgemeine Trend in den Initiativen zur Entwicklung einer Alternative zu DTDs geht zu einem XML-Schema, das eine strengere Inhaltskontrolle für Datenanwendungen bietet, wozu aber die XML-Synatx verwendet wird, um eine Umgebung zu implementieren, wo der gesamte Code durch Maschinen erzeugt werden kann.

Eine der immer deutlicher sichtbaren Schwächen von XML ist sein Datenmodell – was der DTD den Rest gibt. Die Beziehungen zwischen Elementen in einem XML-Dokument sind rein hierarchisch. Es gibt keine Möglichkeit, komplexere Beziehungen auszudrücken. (Es gibt eine SGML-Anwendung, Topic Map Navigation, die eine Lösung dafür bietet, sich aber bisher noch nicht wirklich durchgesetzt hat.) Wenn Sie die Probleme betrachten, die entstehen, wenn Sie große Datenmengen über ein bereits überlastetes Internet übertragen wollen, gewinnen Aspekte der Größe und Wirtschaftlichkeit plötzlich an Bedeutung. Viele glauben, XML brauche einfach eine objektorientierte Hierarchie; es müßte möglich sein, mit Objektklassen zu arbeiten, wie beispielsweise `bestellungen` oder `verkäufe`. Diese Klassen könnten Unterklassen allgemeinerer Klassen sein, wie beispielsweise `transaktion`, von denen die Klassen Eigenschaften wie `wert`, `datum`, `kreditkartennummer` usw. erben gleichzeitig aber auch »lokale« Eigenschaften erhalten könnten, wie beispielsweise `rabatt`. Nichts davon kann mit einer DTD realisiert werden, könnte aber im Entwurf möglicher Alternativen berücksichtigt werden.

Die XML-DTD ist noch nicht tot, aber es ist nur noch eine Frage der Zeit, wann sie veraltet sein wird. Es gibt keinen direkten Ersatz dafür, aber mögliche Alternativen warten schon. Man behauptet, daß es konkurrierende Ansätze geben wird, insbesondere weil Microsoft (XML-Data) und Netscape (RDF) zu den Wettstreitern gehören. Das Rennen ist jedoch ungleich, weil diese beiden Schemata nicht unbedingt auf dieselben Problemstellungen ausgelegt sind. Ein drittes Schema (DCD) könnte beides kombinieren. Wir können nur abwarten.

XML-Data

XML-Data war das erste XML-Schema, und es ist definitiv eine Lösung für Programmierer, die Schemata zur Definition der Eigenschaften von Objektklassen einsetzt.

Schemata setzen sich aus Deklarationen für Konzepte und Objektklassen mit Klassenhierarchien, Eigenschaften, Beschränkungen und Beziehungen zusammen.

Wie alle XML-Datenvorschläge verwendet XML-Data Elemente und Namensräume, die die Bedeutung der Elemente kennzeichnen. Die Syntax eines typischen XML-Data-Schemas sieht wie folgt aus:

```
XML version='1.0' ?>
<?xml:namespace name="urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA110C14882/"
   as="msschema"/?>
<msschema:schema id='ExampleofSchemaSyntax'>
  <!-- schema goes here. -->
</msschema:schema>
```

Wir haben bereits über Namensräume gesprochen. Normalerweise verweisen Namensraumdeklarationen auf Web-Sites, die (hoffentlich) Vokabular enthalten, das die im Schema verwendeten Elemente beschreibt; ihre Namen, ihre Aufgabe und ihren Inhalt. In diesem Fall ist der Zeiger ein URN, und die Adresse ist dem Registrierungseintrag von Microsoft Windows ganz ähnlich (so wie der Registrierungseintrag für das Microsoft `Data Source`-Objekt, der verwendet wird, um XML-Elemente auf HTML-Elemente abzubilden und sie auf diese Weise im Internet Explorer darzustellen, wie in Kapitel 12 erklärt).

XML-Data ist, wie der Name schon sagt, insbesondere auf Daten ausgelegt. Es verwendet Datentypen wie `string` und `integer`; es verwendet Beschränkungen, wie beispielsweise Minimal- und Maximalwerte; es erlaubt Mehrfachschlüssel, so daß Daten aus unterschiedlichen Quellen kombiniert werden können. Und darüber hinaus können andere Schemata, wie beispielsweise Datenbankschemata, auf das XML-Data-Schema abgebildet werden – die vielleicht wichtigste Eigenschaft.

Resource Description Framework – der Rahmen für die Ressourcenbeschreibung

Das RDF (*Resource Definition Framework*) soll eine Lösung für das Problem bieten, daß zwar alles im Web von Maschinen *gelesen* werden kann, daß Sie es aber nicht *verstehen*. RDF stellt ein Schema für Metadaten bereit, also Daten über Daten, die die Daten im Web beschreiben.

Aufbauend auf der Vorarbeit der PICS-Initiative und dem Dublin Core (zwei weitere wichtige Metadaten-Initiativen) konzentriert sich RDF auf Funktionen, die die automatische Verarbeitung von Web-Ressourcen ermöglichen sollen. RDF ist für die Verwendung in unterschiedlichsten Anwendungbereichen vorgesehen; deshalb bleibt es den Anwendungen überlassen, zu unterscheiden, was Daten und was Metadaten sind. Die Autoren der RDF-Arbeitsskizze haben sich insbesondere die folgenden Ziele für ihre Bemühungen gesetzt:

- Vereinfachte Erkennung von Ressourcen, um bessere Suchfunktionen zu unterstützen
- Katalogisierung des Inhalts und der Beziehungen zwischen bestimmten Web-Sites, Seiten oder digitalen Bibliotheken
- Aktivierung intelligenter Software-Agenten für den Austausch und die gemeinsame Nutzung von Wissen
- Die Bereitstellung von Informationen, die eine Auflistung von Seiten beschreiben und die ein einziges logisches Dokument darstellen
- Schutz des geistigen Eigentums von Webseiten
- Implementierung digitaler Signaturen für den E-Kommerz.

PICS (*Platform for Internet Content Selection*) wurde ursprünglich für die profanere Aufgabe entwickelt, »Erwachsenen«-Websites so zu überwachen, daß Filter-Software »ungeeigneten« Inhalt vor versehentlichen Zugriffen schützen konnte.

Der *Dublin Core* (benannt nach Dublin, Ohio, der Heimat des Online Computer Library Center) ist eine Zusammenstellung von Elementen, die von Bibliothekaren, Sachverständigen und Text-Markup-Spezialisten entwickelt wurde. Man suchte dabei nach Informationsatomen. Diese Elemente umfassen beispielsweise `DATE`, `TITLE`, `SUBJECT` und `LANGUAGE`.

In der Praxis (stark vereinfacht betrachtet) ist der RDF-Mechanismus extrem einfach. Das Modell besteht aus drei Komponenten: einer Ressource (einer URI), einem Eigenschaftstyp und einem Wert. Diese drei Informationselemente sind einfache XML-Ele-

mente, deshalb verwendet der XML-Code zu ihrer Identifikation den RDF-Namensraum. Listing 17.1 zeigt den RDF-Code (natürlich in XML) für ein Buch. Gemäß dem Dublin Core hat ein Buch einen Titel und der Autor wird als `creator` bezeichnet.

Listing 17.1: Ein einfaches RDF-Beispiel für ein Buch

```
1: <?xml version="1.0"?>
2:   <rdf:RDF
3:     xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
4:     xmlns:books="http://www.booksRus.com/schemas/books/">
5:     <rdf:Description about="http://www.xs4all.nl/~sintac">
6:       <books:Creator>Simon North</books:Creator>
7:     </rdf:Description>
8:   </rdf:RDF>
```

Der RDF-Namensraum (Sie müssen nicht unbedingt den String `rdf` als Bezeichner verwenden, sondern auch einen beliebigen Wert auswählen) bezieht sich auf die W3C-Webseite, die RDF beschreibt (`http://www.w3.org/TR/WD-rdf-syntax`). Der Namensraum `books` zeigt auf die Position auf der Seite Books'R' Us, wo ein Vokabular angibt, was diese Elemente bedeuten. Durch diese Art der Verwendung von Namensräumen und Web-Sites sind Sie nicht auf bestimmte Elementnamen beschränkt. Durch einen Verweis auf das Vokabular kann jede Anwendung die Bedeutung oder die Aufgabe der in einem Element enthaltenen Information herausfinden.

Neben der Verwendung einfacher Elemente erlaubt Ihnen RDF auch, Ressourcenauflistungen anzulegen, sogenannte `bags` und `sequences`. Grundsätzlich sind diese Auflistungen gleich. Ein `bag` ist jedoch im Gegensatz zu einem `sequence` nicht sortiert. Listing 17.2 zeigt die Verwendung eines `bag`, wobei das Buch mehrere Autoren hat.

Listing 17.2: Ein einfaches RDF-Beispiel für ein Buch

```
1:  <?xml version="1.0"?>
2:    <rdf:RDF
3:      xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
4:      xmlns:books="http://www.booksRus.com/schemas/books/">
5:      <rdf:Description about="http://www.xs4all.nl/~sintac">
6:        <books:Creator>
7:           <RDF:Seq>
8:              <books:LI>Simon North</books:LI>
9:              <books:LI>Paul Hermans</books:LI>
10:          </RDF:Seq>
11:       </books:Creator>
12:     </rdf:Description>
13:   </rdf:RDF>
```

Ich verwende ein `Seq`-Element zur Gruppierung der beiden Autoren, weil die Reihenfolge, in der sie aufgelistet werden, eine Rolle spielt. Andernfalls hätte ich das `Bag`-Element verwendet.

Es gibt natürlich sehr viel mehr zu RDF zu sagen, was aber nicht im Rahmen dieses Buchs besprochen werden kann, weil es alles recht theoretisch ist. Eine praktische Seite von RDF werden Sie jedoch bemerken, weil Netscape es bereits in Mozilla implementiert hat (das ist die Testumgebung für zukünftigen Versionen des Netscape Communicator), um Lesezeichen zu beschreiben. In dieser Hinsicht ist RDF Netscapes Antwort auf den Active Desktop von Microsoft sowie die Channels (die auf CDF basieren, einer anderen XML-Anwendung). Listing 17.3 zeigt eine typische RDF-Datei von Mozilla.

Listing 17.3: Eine RDF-Ressourcendatei aus Mozilla

```
1:  <RDF:RDF>
2:  <Topic id="NC:Toolbar">
3:  <child>
4:  <Topic id="NC:CommandToolBar" name="Command Toolbar"
5:        toolbarBitmapPosition="top"
```

```
6:  toolbarButtonsFixedSize="yes"  >
7:  <child href="command:back" name="Back"/>
8:  <child buttonTooltipText="Reload this page from the server"
9:     buttonStatusbarText="Reload the current page"
10:     href="command:reload" name="Reload"/>
11: <child href="command:stop" name="Stop"/>
12: <child href="command:forward" name="Forward"/>
13: <child name="separator0" href="nc:separator0"/>
14: <child href="command:urlbar" name=" "
15:     buttonStatusBarText="Location/Search Bar"
16:    buttonTooltipText="Location/Search Bar"
17: urlBar="Yes" urlBarWidth="*"/>
18: <child name="separator2" href="nc:separator2"/>
19: </Topic>
20: </child>
21:
22: <child>
23: <Topic id="NC:InfoToolbar" name="Info Toolbar">
24:    <child>
25:    <Topic id="NC:Bookmarks" name="Bookmarks"></Topic>
26:    </child>
27:    <child>
28:    <Topic id="NC:History"
29:   largeIcon="icon/large:workspace,history" name="History">
30:           <child href="NC:HistoryMostVisited"
31:        name="Most Frequented Pages"/>
32:     <child href="NC:HistoryBySite" name="History By Site"/>
33:   <child href="NC:HistoryByDate" name="History By Date"/>
34:         </Topic>
35:         </child>
36:     <child href="NC:Sitemaps" name="Related"
37:      htmlURL="http://rdf.netscape.com/rdf/navcntradvert.html"/>
38: </Topic>
39: </child>
40:
41: <child>
42: <Topic id="NC:PersonalToolbar" name="Personal Toolbar">
43: </Topic>
44: </child>
45: </Topic>
46:
47: <Topic id="NC:NavCenter">
48: <child href="NC:Bookmarks" name="Bookmarks"/>
49: <child href="NC:Search"
50:     largeIcon="icon/large:workspace,search" name="Search"/>
51: <child href="NC:History" name="History"/>
```

```
52: <child id="NC:Sitemaps" name="Site Tools"
53:  htmlURL="http://rdf.netscape.com/rdf/navcntradvert.html" />
54: <child id="NC:LocalFiles" name="Files"
55:   largeIcon="http://rdf.netscape.com/rdf/heabou.gif"/>
56: </Topic>
57:
58: <Topic id="NC:SmartBrowsingProviders">
59:   <child href="http://altavista.digital.com/
60:           cgi-bin/query?q=link%3A"
61:           name="Who points to me?"
62:           resultType="TEXT/HTML"/>
63:     <child href="http://www-rl1.netscape.com/wtgn?"
64:         name="Related Links"
65:           resultType="TEXT/RDF" />
66:  </Topic>
67:
68: </RDF:RDF>
```

DCD (Document Content Description, Dokumentinhaltsbeschreibung)

XML-Data und RDF wurden dem W3C innerhalb weniger Monate zur Begutachtung unterbreitet. Sofort wurden die Beteiligten (Microsoft: XML-Data und Netscape: RDF) beschuldigt, proprietäre Lösungen anzubieten, die die Benutzer auf ganz bestimmte Produkte festlegen.

Der DCD-Vorschlag (*Document Content Description*) sucht einen Kompromiß. Er nimmt eine Untermenge des XML-Data-Vorschlags und drückt sie so aus, daß sie mit RDF konsistent ist (d.h. DCD ist ein RDF-Vokabular).

Einfach ausgedrückt, die Seele von DCD ist ein `ElementDef`-Element, das die Attribute `Type`, `Model`, `Occurs`, `RDF:Order`, `Content`, `Root`, `Fixed` und `Datatype` hat oder diese Elemente enthält. Das `ElementDef`-Element beschreibt also ein in einem XML-Dokument enthaltenes Element. Neben diesen »speziellen« Elementen (oder Attributen) enthält das `ElementDef`-Element auch Details zu den Elementen, die es enthalten kann, sowie die Definitionen ihrer Attribute. Eine typische DCD-Deklaration in einer XML-DTD könnte wie folgt aussehen:

```
<ElementDef Type="shape" Model="Elements">
  <AttributeDef Name="n" Occurs="Optional"/>
  <AttributeDef Name="Sides"/>
</ElementDef>
```

```
<ElementDef Type="squareshape" Model="Element">
  <AttributeDef Name="Sides" Occurs="Optional">
     <Default>3</Default>
  </AttributeDef>
  <Element>side</Element>
  <Extends Type="shape"/>
</ElementDef>
```

In dem entsprechenden XML-Dokument könnte ein typisches Element wie folgt aussehen:

```
<squareshape n="4">
   <side><dimension unit='in'>5<dimension></side>
</squareshape>
```

Wie in RDF können Kindelemente gruppiert werden, und bei Bedarf können Sie auch deklarieren, ob die Reihenfolge der Elemente eingehalten werden muß, wie in Listing 17.4 gezeigt.

Listing 17.4: Typische DCD-Elementdefinitionen

```
1:  <ElementDef Type="Candidate" Model="Elements" >
2:    <Description>Personal Details</Description>
3:    <Group RDF:Order="Seq">
4:      <Element>GivenName</Element>
5:      <Group Occurs="Optional">
6:         <Element>Initial</Element>
7:      </Group>
8:      <Element>FamilyName</Element>
9:      <Element>common:Address</Element>
10:     <ElementDef Type="Telephone" Datatype="string"/>
11:   </Group>
12: </ElementDef>
13:
14: <ElementDef Type="Loan">
15:   <Description>Morgage Loan</Description>
16:   <Group RDF:Order="Seq">
17:     <Element>InterestRate</Element>
18:     <Element>Amount</Element>
19:     <Element>Maturity</Element>
20:   </Group>
21: </ElementDef>
```

Weil DCD aus XML-Data und RDF entstanden ist, besitzt es ihrer beider Vorteile, wie beispielsweise die Datentypen von XML-Data und die Flexibilität von RDF.

XSchema

Während hinter XML-Data, RDF und DCD große Namen stehen (Microsoft, Netscape und IBM), ist XSchema das Ergebnis langer Diskussionen auf der XML-DEV-Mailingliste im Internet. Zwar waren berühmte Namen daran beteiligt, aber XSchema erhielt keine größere Unterstützung in Hinblick auf die kommerzielle Entwicklung und ist einfach das Ergebnis der harten Arbeit einiger weniger. Unter anderem war auch Simon Str. Laurent beteiligt, der als Koordinator, Motivator, Mitarbeiter und Editor fungierte.

Die Vorschläge für XML-Data, RDF und DCD bedingen, daß Sie die Schema-Information in die DTD aufnehmen. XSchema ist anders. Es besitzt eine eigene DTD, was in gewisser Weise eine sinnvolle Formalisierung der Syntax bedeutet, aber sein Schema-Dokument liegt in Standard-XML-Syntax vor und nicht in einer separaten DTD-Syntax. Es gibt bereits Software, mit deren Hilfe eine XML-DTD in ein XSchema-Dokument umgewandelt werden kann, und es wird sicher bald mehr davon geben – auch für die Konvertierung in Gegenrichtung.

Wie bei den anderen Vorschlägen ist die Elementdeklaration das Herz von XSchema. XSchema verwendet eine Elementdeklaration, deren eigene Deklaration (aus der XSchema-DTD) wie folgt aussieht:

```
<!ELEMENT ElementDecl (Doc?, More?, Model, AttGroup?)>
<!ATTLIST ElementDecl
          Name    NMTOKEN  #REQUIRED
          id      ID       #REQUIRED
          prefix  NMTOKEN  #REQUIRED
          ns      CDATA    #IMPLIED
          Root    (Recommended ¦ Possible ¦ Unlikely) "Possible">
```

Die meisten dieser Kindelemente und Attribute sprechen für sich selbst, aber eines davon ist nicht so offensichtlich und soll weiter erklärt werden. Das Element `More` erlaubt jedem, das XSchema durch eigene Ergänzungen zu erweitern. Sie könnten beispielsweise eine strengere Kontrolle für den möglichen Inhalt von Elementmodellen einführen.

Zusammenfassung

XML-Schemata sind ein sehr trockenes, sehr technisches Thema. Gleichzeitig sind sie unabdingbar für den Erfolg oder Mißerfolg von XML und werden in seiner Zukunft eine immer größere Rolle spielen. Dieses Kapitel hat versucht, Ihnen einige Grundlagen der vorgeschlagenen Schemata näherzubringen.

F&A

F Warum sind DTDs für Daten nicht ausreichend?

A *DTDs bieten keine Beschränkung in Hinblick auf den Inhalt – und wo sie es tun, sind sie zu locker. Die Beziehungen zwischen den Elementen sind zu stark vereinfacht, und DTDs haben eine andere Syntax als XML-Dokumente, wodurch ihre Erstellung durch Maschinen schwierig wird.*

F Warum trafen XML-Data und RDF auf so viel Kritik?

A *Beide wurden – zurecht oder zu unrecht – zunächst mißtrauisch betrachtet, weil es sich um proprietäre Lösungen handelte, die der offenen Politik von XML entgegenstanden.*

F Was unterscheidet XSchema so deutlich von den anderen Vorschlägen?

A *Anders als die anderen Vorschläge, die die DTD-Syntax verwenden, verwendet XSchema dieselbe Syntax wie normale XML-Dokumente.*

Übungen

1. Ein Großteil des XML-Schema-Vorschlags basiert auf einem Namensraumzeiger, bei dem es sich um eine URL handelt. Zumindest theoretisch sollte der Zeiger auf ein Vokabular verweisen, das die zu dem Namensraum gehörigen Elemente beschreibt. URLs – wie Sie aus dem Web wissen – ändern sich häufig, ohne daß es zuvor Benachrichtigungen darüber gibt, und es gibt keinen Weiterleitungsmechanismus. Das ist für einen Namensraum nicht denkbar; deshalb wurden viele der URLs in PURLs (Persistent URLs) umgewandelt. Suchen Sie in einer Ihrer bevorzugten Web-Engines Informationen über PURLs, und probieren Sie, sich bei einer davon zu registrieren.

2. Treten Sie der XML-DEV-Mailing-Liste bei. Sie haben vielleicht noch nicht viel beizutragen, und die Diskussionen sind häufig recht technisch orientiert, aber es ist gut zu wissen, daß jemand zuhört, der im Ernstfall helfen könnte. Wenn Sie sich korrekt verhalten, dann werden Ihre Beiträge immer willkommen sein – achten Sie jedoch darauf, daß es sich hier nicht um eine Mailing-Liste für Anfängerfragen handelt.

Aufbereitung von XML mit CSS

Woche 3

In den vergangenen Kapiteln haben Sie erfahren, wie Sie Ihren XML-Code lesbarer machen. Es gibt verschiedene Möglichkeiten, XML-Code in HTML umzuwandeln oder ihn anderweitig zu verarbeiten, um ihn in einem Web-Browser anzuzeigen.

Die beiden führenden Browser (von Microsoft und Netscape) verfolgen zwei völlig unterschiedliche Ansätze dafür. Wenn Ihr Code für einen Browser optimiert wird, kann er im anderen sehr wahrscheinlich nicht korrekt angezeigt werden. Darüber hinaus gibt es keine Standardisierung ihrer Ansätze und keinerlei Konsistenz zwischen unterschiedlichen Versionen desselben Produkts. Beispielsweise existieren in den Versionen 4 und 5 des Internet Explorer völlig unterschiedliche Erwartungen und Anforderungen.

Sie fragen sich vielleicht, wozu man eine standardisierte Markup-Sprache braucht, wenn es keine Standards für ihre Darstellung gibt. Dasselbe haben sich schon viele andere vor Ihnen gefragt, und auch mir kriecht der Gedanke jedesmal im Kopf herum, wenn ich eine Anwendung vermisse, die XML-Code in einer anderen, proprietären Implementierung von SGML- und XML-Software anzeigt.

Um dieses und viele ähnliche Probleme zu lösen, wurden zahlreiche Versuche unternommen, eine standardisierte Darstellungssprache (Stilsprache) zu entwickeln. In diesem und den nächsten beiden Kapiteln werden mehrere dieser Stilsprachen vorgestellt. Glücklicherweise kann XML die Früchte dieser Arbeit zu großen Teilen nutzen. Sie können es mit DSSSL (*Document Style Semantics and Specification Language*) verarbeiten, weil XML seine Wurzeln immer noch in SGML und DSSSL viel von SGML übernommen hat. Außerdem können Sie CSS (*Cascading Style Sheets*) aus HTML verwenden, weil XML und HTML mit SGML einen gemeinsamen Vorfahren und im World Wide Web eine gemeinsame Umgebung haben.

Heute geht es um die folgenden Dinge:

- Sie erhalten einen kurzen Überblick über die Geschichte einiger der wichtigsten Stilsprachen.
- Sie lernen die Grundlagen von CSS (Cascading Style Sheet) und seine Anwendung auf XML-Code kennen.

Aufstieg und Niedergang der Stilsprache

In der gesamten Geschichte von SGML finden sich immer wieder Versuche, eine Stilsprache zu definieren, die den mit Markup ausgezeichneten Text auf Papier oder einen Bildschirm ausgibt.

Die erste Sprache für die Stilspezifikation war FOSI (*Formating Output Specification Instance*), die zum Teil erfolgreich im amerikanischen Verteidigungsministerium ein-

gesetzt wurde, sich aber aufgrund ihrer Komplexität nie wirklich durchsetzen konnte. (ArborText war meines Wissens das einzige Software-Unternehmen, das fast eine kommerziell angebotene Arbeitsimplementierung implementiert hat.) Parallel zur FOSI-Initiative entstand DSSSL, die *Document Style Semantics and Specification Language*, die eine vollständige Programmiersprache für die Festlegung von Stilen eines SGML-Dokuments bereitstellt. DSSSL kann auch auf XML-Dokumente angewendet werden, wie Sie in Kapitel 19 noch erfahren.

DSSSL ist extrem leistungsfähig und kompliziert und für die Verwendung im Web vielleicht ein bißchen zu viel. Deshalb wurde eine abgespeckte Version angeboten, *DSSSL-Lite*. DSSSL-Lite wurde schließlich zu einer Internet-Variante, DSSSL-o (für DSSSL-online). DSSSL-o wurde im Dezember 1995 veröffentlicht und im August 1996 korrigiert, um die Änderungen des endgültigen DSSSL-Standards zu berücksichtigen. DSSSL-o wurde zur Grundlage der ersten XSL-Version (XS, kurz für *xml-style*).

Im Mai 1997 stellte Jon Bosak, Mitherausgeber der XML-Sprachspezifikation, eine persönliche Skizze des Stilabschnitts für die XML-Spezifikation öffentlich im Internet zur Verfügung. (Zu diesem Zeitpunkt war die XML-Spezifikation in drei Abschnitte unterteilt: einen für die Sprache – jetzt die XML Language Recommendation –, einen für das Linking und einen für den Stil.) Der vielsagende Titel des Dokuments »XML Part 3: Style [NOT YET]« sollte darauf hinweisen, daß der Inhalt mit Vorsicht zu genießen war, weil er sich sehr wahrscheinlich noch ändern würde. Das war jedoch noch nicht abschreckend genug, und kurze Zeit später erschien die erste Software.

Im August 1997 erschien eine stark verbesserte Version von XS, jetzt unter dem Namen XSL (*Extensible Style Language*). Dabei handelte es sich um die erste XSL-Version. Ich nenne diese XSL-Version XSL1, um sie von früheren Versionen zu unterscheiden. XSL1 beinhaltete das gesamte DSSSL-o, aber auch neue Objekte und Eigenschaften, die die gesamte Funktionalität von CSS unterstützten.

Ein großer Unterschied zwischen DSSSL-o und XSL1 war die Syntax. DSSSL-o war ganz offensichtlich ein LISP-Derivat, was man an den den LISP-Programmierern so vertrauten zeilenweisen Klammern erkennt. XLS1 dagegen verwendet dieselbe Syntax wie XML. Die Auswirkungen dieser Änderungen waren tiefgreifend. Beispielsweise konnte der Stilabschnitt von XML jetzt vom Computer verarbeitet (und sogar angelegt) werden.

Während sich die Syntax der Stilsprache änderte, wurden viele der ursprünglichen DSSSL-Konzepte unverändert übernommen (wie etwa Flußobjekte). Diese Inkarnation der XML-Stilsprache war jedoch relativ kurzlebig (weniger als ein Jahr), obwohl es Software gibt, die diese Version der Sprache immer noch unterstützt.

Mehr über diese neue XSL-Version erfahren Sie in Kapitel 20. Ich bezeichne diese XSL-Version hier als XSL2, um sie von der früheren Version zu unterscheiden.

Während die Entwicklung einer dedizierten XML-Stilsprache weitergeht, sollte man nicht vergessen, daß HTML eine eigene Stilsprache besitzt, CSS. CSS kann ebenfalls auf XML-Code angewendet werden. Und weil CSS darauf ausgelegt ist, Markup in einem Web-Browser auszugeben, ist es für die grundlegenden Ausgabefunktionen bestens geeignet. Im restlichen Kapitel erklären wir, wie wir die Web-Browser-Unterstützung von CSS und HTML nutzen können, um mit kleinstem Aufwand XML-Code anzuzeigen.

CSS – Cascading Style Sheets

Ich habe bereits erwähnt, daß XSL1 aus XS entstanden ist und dabei Funktionen aus CSS übernommen hat. XSL1 hat einige der CSS-Eigenschaften als Flußobjekte übernommen (mehr darüber erfahren Sie in Kapitel 19). Dadurch sollte es den Anwendern einfacher gemacht werden, CSS-Stylesheets in XSL-Stylesheets umzuwandeln, nicht nur durch die Code-Konvertierung, sondern auch unter Beibehaltung derselben Konzepte.

Es gibt zwei Versionen von CSS: CSS Level 1 (CSS1), wobei es sich um eine W3C-Empfehlung vom Dezember 1996 handelt, und CSS Level 2 (CSS2), das im November 1997 als Vorschlag veröffentlicht wurde und das heute in einer Arbeitsskizze vom Januar 1998 existiert. Noch unterstützen sehr wenige Web-Browser die neueste CSS-Version (und einige haben sogar noch Probleme mit der älteren Version), deshalb werde ich mich hier auf CSS1 beschränken. CSS2 bietet zahlreiche neue Funktionen, ist aber nicht abwärtskompatibel zu CSS1.

Ich werde hier nicht versuchen, Ihnen CSS1 beizubringen. Es gibt zahlreiche gute Bücher zu diesem Thema und jede Menge Informationen im Web darüber.

Die Nutzung des HTML- und CSS-Codes Dritter ist Diebstahl. Man kann Sie jedoch nicht dafür bestrafen, wenn Sie diesen Code benutzen, um daraus zu lernen.

Eine einfache Methode, den CSS-Code herunterzuladen, ist die Verwendung einer alten Version eines Web-Browsers. Geeignet sind Netscape 2 oder auch Mosaic, aber Sie sollten sicherstellen, daß es sich dabei um eine Version handelt, die keine CSS-Stylesheets unterstützt. Wenn Sie in Ihrem normalen Web-Browser ein interessantes Stylesheet sehen, durchsuchen Sie den HTML-Quellcode nach der Adresse (in Netscape View > Page Source, im Internet Explorer View > Source) des Stylesheet (sehen Sie dazu

in den META-Tags am Dokumentanfang nach). Schneiden Sie die Adresse des Stylesheet aus, laden Sie die alte Browser-Version, und tragen Sie diese Adresse in das URL-Feld ein (vergessen Sie nicht, den Pfad zu dem ursprünglichen Dokument anzugeben, und lösen Sie alle relativen Pfade auf). Jetzt sollten Sie den Inhalt des Stylesheet als ASCII-Text sehen, den Sie auf Ihrer Festplatte speichern können.

Wenn Sie einen zweiten Browser verwenden, sollten Sie DLL- oder andere Konflikte vermeiden. Sie können Ihre Lesezeichen oder Favoriten zerstören, aber auch wichtige Einstellungen. Wenn Sie eine sehr alte Version verwenden, gibt es vermutlich keine Probleme, aber Sie sollten separate Profile für die beiden Versionen einrichten.

XML, CSS und Web-Browser

Statt Ihnen CSS beizubringen, zeige ich Ihnen, wie CSS für XML-Code verwendet wird.

Alles, was ich hier erkläre, wird sich natürlich ändern, weil die drei von mir verwendeten Browser (Microsoft Internet Explorer 5 Beta Preview 2, Netscape Communicator 4.5 Preview 2 und Netscape Mozilla Build September 1998) noch keine offiziellen Produkte, sondern Testversionen sind.

Leider ist die korrekte Anzeige von XML (mit einem sinnvollen Layout) eine recht komplizierte Angelegenheit, und Sie werden viel Zeit damit verbringen, den Code zu ändern, zu laden, wieder zu ändern – bis Sie schließlich beim gewünschten Ergebnis sind oder einfach aufgeben. Verlieren Sie den Mut nicht – es gibt Tricks, die Ihnen ermöglichen, die in den Browser eingebauten HTML-Interpretationen zu nutzen, und später in diesem Kapitel werde ich Ihnen zeigen, wie man sie benutzt. Leider ändern sich aber nicht nur die korrekten Methoden, XML anzuzeigen, sondern auch die illegalen.

XML, CSS und der Internet Explorer

Microsoft treibt momentan sehr viel Aufwand mit XML (mehr über ihre Pläne erfahren Sie in Kapitel 21). Es will aber Dynamic HTML als Grundlage darstellbaren Codes beibehalten. Das bedeutet, die XML-Unterstützung im Internet Explorer 5 ist begrenzt. Sie können den XML-Code zwar anzeigen, aber Microsoft konzentriert sich

mehr darauf, dynamisch XML-Elemente in HTML-Elemente umzuwandeln (die XML-Data-Source-Objekte haben Sie bereits in Kapitel 12 kennengelernt). XML ist dabei kaum mehr als ein gutes Datenübertraungsschema – Teil der von Microsoft sogenannten »Dateninseln«.

Egal, was Microsoft mit XML vorhat – interessant ist, daß der Internet Explorer 5 momentan der einzige Web-Browser ist, der XML wirklich unterstützt. (Netscapes Mozilla-Code bietet eine beschränkte Unterstützung, wie Sie später in diesem Kapitel noch sehen werden, aber dieser Browser steht nur als Quellcode zur Verfügung, und die Kompilierung ist relativ aufwendig). Leider unterstützt IE5 XL auf relativ unkonventionelle Weise, die nicht dem Standard entspricht. Niemand kann Microsoft dafür tadeln; wenn es keine Standards gibt, kann man schlecht damit konform sein.

Bevor wir die Anzeige von XML-Code in IE5 betrachten, wollen wir das Ergebnis sehen, wie wir es uns vorstellen. Abbildung 18.1 zeigt eine einfache Webseite, in HTML geschrieben und mit einem separaten CSS-Stylesheet, das für das visuelle Layout sorgt. Den HTML-Code sehen Sie in Listing 18.1, den CSS-Code in Listing 18.2.

Listing 18.1: Der HTML-Code für das Zieldokument

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2:  <HTML>
3:  <HEAD>
4:     <LINK rel="stylesheet" href="SAMPLEXML.css">
5:     <TITLE>Sample XML Home Page</TITLE>
6:  </HEAD>
7:  <BODY>
8:  <DIV CLASS="CHAPTER">
9:  <H1>SAMPLE WEB PAGE</H1>
10: <P>Welcome to one of the world's first WWW Home pages
11: written entirely in XML. This web page has been
12: constructed using:</p>
13: <UL>
14: <LI>basic XML code (derived from HTML)</LI>
15: <LI>a CSS stylesheet to get the page to display
16: in Microsoft Internet Explorer</LI>
17: <LI>a separate CSS stylesheet to get the page
18: to display in Netscape Communicator</LI>
19: <LI>another CSS stylesheet to get the page to display
20: in Netscape Mozilla (a September 1998 build)</LI>
```

```
21: <LI>variations of the XML code embedded as pseudo-HTML
22: code inside a standard HTML web page</LI>
23: </UL>
24: <P>Explore to your heart's content.</P>
25: <P CLASS="WARNING">Note that all of this code is experimental so don't ➥be
disappointed
26: if something doesn't quite work the way you think it should.</P>
27: <CENTER>
28: <ADDRESS>
29: <HR NOSHADE WIDTH="100%">This page last updated: 21 August ➥1998</ADDRESS>
30: </CENTER>
31: </DIV>
32: </BODY>
33: </HTML>
```

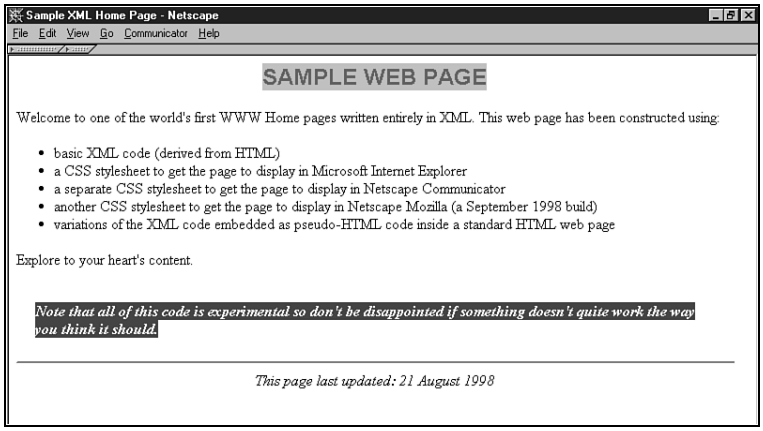


Abbildung 18.1: Das formatierte HTML-Dokument im IE5.

Bei der genaueren Betrachtung von Listing 18.1 sehen Sie, daß es sich zwar um HTML-Code handelt, aber auch (mit Ausnahme des leeren HR-Elements) um korrekten XML-Code. Er enthält eine Dokumenttypdeklaration, alle Elemente sind korrekt verschachtelt, die Attributwerte sind in Anführungszeichen eingeschlossen (»«) und es gibt für alle Elemente schließende Tags. Wenn Sie je XML-Dokumente anlegen, sollten Sie sich diese Vorgehensweise angewöhnen. (Beachten Sie jedoch, daß die HTML-4-DTD zwar eine korrekte XML-DTD ist, Sie aber nicht davon ausgehen können, daß Sie Dokumente anhand dieser DTD auswerten können.)

Listing 18.2: Der CSS-Code für das Zieldokument

```
1:  BODY {
2:     color: #000000;
3:  }
4:
5:  H1 {
6:      font-family: Arial;
7:      font-size: 18pt;
8:      font-weight: bold;
9:      color: #FF0000;
10:     background-color: #C0C0C0;
11:     text-align: center;
12:     border-top: solid;
13:     border-bottom: solid;
14:     border-left: solid;
15:     border-right: solid;
16:    }
17:
18: A:LINK {
19:      color: #0000FF;
20:      }
21:
22: A:VISITED {
23:     color: #990099;
24:     }
25:
26: .WARNING {
27:     font-style: italic;
28:     font-size: 12pt;
29:     font-weight: bold;
30:     color: #FFFFFF;
31:     background-color: #0000FF;
32:     text-decoration: blink;
33:     margin-top: 8pt;
34:     margin-bottom: 8pt;
35:     margin-left: 10pt;
36:     margin-right: 10pt;
37:     padding-top: 1%;
38:     padding-bottom: 1%;
39:     padding-left: 1%;
40:     padding-right: 1%;
41:     border-top: dashed #000000;
42:     border-bottom: dashed #000000;
43:     border-left: dashed #000000;
44:     border-right: dashed #000000;
45: }
```

Beachten Sie, daß der CSS-Code in Listing 18.2 durch die Kombination einiger Eigenschaften auch hätte verkürzt werden können. Beachten Sie außerdem, daß ich die Eigenschaften für viele der Elemente einfach nicht deklariert habe. Ich kann es dem Web-Browser überlassen, seine Standardeigenschaften dafür zu verwenden. Wie Sie gleich sehen werden, müssen in XML alle Eigenschaften für alle Elemente explizit angegeben werden.

Der HTML-Code im Beispiel ist fast XML-Code. Weil es hier um die Anzeigemechanismen geht, wollen wir alles so einfach wie möglich halten. Um den HTML-Code in XML-Code umzuwandeln, benenne ich einfach einige der Elemente um. Das Dokument wird zu einem PAGE-Element, eine UL wird zu einem LIST-Element usw. Das Ergebnis sehen Sie in Listing 18.3. (Beachten Sie, daß ich ein bißchen geschummelt und das HR-Element weggelassen habe. Dem Stil für das Element könnte einfach eine horizontale Linie hinzugefügt werden, aber ich wollte das Stylesheet hier nicht unnötig verkomplizieren.)

Listing 18.3: Das Zieldokument, jetzt als XML-Code

```
<PAGE>
<TITLE>SAMPLE WEB PAGE</TITLE>
<PARA>Welcome to one of the world's first WWW Home pages
written entirely in XML. This web page has been
constructed using:</PARA>
<LIST>
<ITEM>basic XML code (derived from HTML)</ITEM>
<ITEM>a CSS stylesheet to get the page to display in
Microsoft Internet Explorer</ITEM>
<ITEM>a separate CSS stylesheet to get the page to display
in Netscape Communicator</ITEM>
<ITEM>another CSS stylesheet to get the page to display
in Netscape Mozilla (a September 1998 build)</ITEM>
<ITEM>variations of the XML code embedded as pseudo-HTML
code inside a standard HTML web page</ITEM>
</LIST>
<PARA>Explore to your heart's content.</PARA>
<WARNING>Note that all of this code is experimental so
don't be disappointed if something doesn't quite work the
```

```
20: way you think it should.</WARNING>
21: <UPDATE>This page last updated: 21 August 1998</UPDATE>
22: </PAGE>
```

Wenn Sie den Code so wie er ist anzeigen, sehen Sie eine Ausgabe wie in Abbildung 18.2 gezeigt. Der Internet Explorer zeigt eine navigierbare Darstellung der Elementstruktur im Dokument an.

Wie Sie in Abbildung 18.2 sehen, könnte das Ergebnis noch viel schlechter sein. Der IE5 zeigt hier das Markup. Aus dieser Darstellung des Inhalts können Sie leicht auf die Struktur der Information im Dokument zugreifen. Durch Anklicken der Symbole Minus (–) oder Plus (+) können Sie die Elementstruktur beliebig erweitern und wieder zusammenklappen. Offensichtlich wurden keine Stile angewendet, weil es keinen Link auf ein Stylesheet gibt, und um korrekt dargestellte XML-Elemente zu erzielen, müßten Sie ein XLS-Stylesheet bereitstellen.

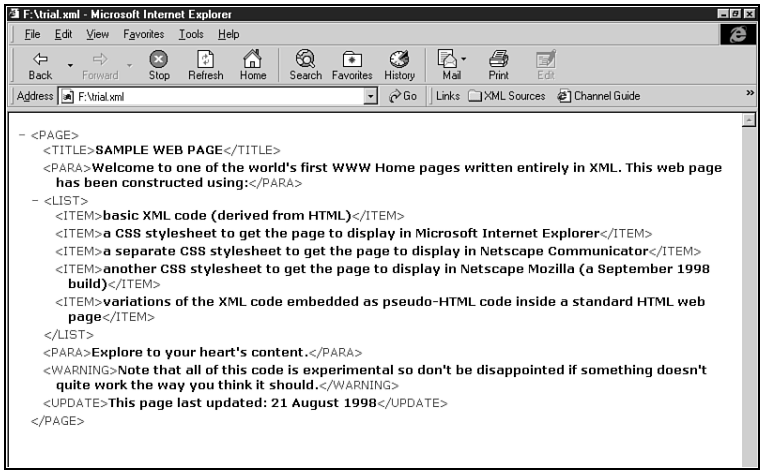


Abbildung 18.2: Das reine XML-Dokument im IE5.

IE5 kann nicht nur XML-Code darstellen, sondern natürlich auch HTML. Statt jetzt also ein vollständiges XSL-Stylesheet anzulegen, erstellen wir einfach ein CSS-Stylesheet und ordnen es dem XML-Code zu. Der IE5 weiß nicht, was er mit XML-Code machen soll, aber er weiß, daß es ihn gibt. Um die XML-Elemente mit den HTML-Ele-

menten zu kombinieren, müssen Sie einen Namensraum deklarieren und kundtun, daß die XML-Elemente diesem Namensraum angehören. Den veränderten XML-Code mit seiner HTML-Hülle sehen Sie in Listing 18.4.

Listing 18.4: Das XML-Dokument mit einem CSS-Link

```
1:  <?xml:stylesheet href="css.css" type="text/css"?>
2:  <PAGE>
3:  <TITLE>SAMPLE WEB PAGE</TITLE>
4:  <html:IMG xmlns:html="htmluri" SRC="logo.gif" ALIGN="middle"/>
5:  <PARA>Welcome to one of the world's first WWW Home
6:  pages written entirely in XML. This web page has been
7:  constructed using:</PARA>
8:  <LIST>
9:  <ITEM>basic XML code (derived from HTML)</ITEM>
10: <ITEM>a CSS stylesheet to get the page to display in
11: Microsoft Internet Explorer</ITEM>
12: <ITEM>a separate CSS stylesheet to get the page to
13: display in Netscape Communicator</ITEM>
14: <ITEM STYLE="color:green;>another CSS stylesheet to get the page to
15: display in Netscape Mozilla (a September 1998 build)</ITEM>
16: <ITEM>variations of the XML code embedded as
17: pseudo-HTML code inside a standard HTML web page</ITEM>
18: </LIST>
19: <PARA>Explore to your heart's content.</PARA>
20: <WARNING>Note that all of this code is experimental so
21: don't be disappointed if something doesn't quite work the way you
22: think it should.</WARNING>
23: <UPDATE>This page last updated: 21 August 1998</UPDATE>
24: </PAGE>
```

In Listing 18.4 verweise ich mit Hilfe einer XML-Verarbeitungsanweisung auf das CSS-Stylesheet. Bei sorgfältiger Betrachtung sehen Sie, daß meine Syntax eigentlich fehlerhaft ist. Gemäß der XML-Spezifikation sollte ich schreiben `xml-stylesheet`, aber der Internet Explorer 5 beschwert sich nicht über die Verwendung von `xml:stylesheet` (und nimmt es Ihnen auch nicht übel, wenn Sie statt dessen `XML` schreiben). Das ist überraschend, weil IE5 alle anderen Aspekte einer XML-Datei sorgfältig prüft.

Ich habe den XML-Code abgeändert und in Zeile 4 einen Verweis auf eine Grafik eingefügt. Diese Syntax entspricht nicht dem Standard, weil der verwendete Namensraum (`html`) nirgends deklariert ist. Aber zweifellos funktioniert sie.

Sie können Ihrem XML-Code auch einen Stil zuordnen, indem Sie die Stilspezifikation inline im Element-Start-Tag angeben (wie in Zeile 14 von Listing 18.4 gezeigt):

```
<ITEM STYLE="color:green;">another CSS stylesheet to get ➥the page to
```

Das CSS-Stylesheet sehen Sie in Listing 18.5.

Listing 18.5: Das CSS-Stylesheet.

```
1: PAGE {display: block; font-family:arial,sans-serif;}
2: TITLE { font-size: 24pt; font-weight: bold;}
3: PARA { display: block; padding-top: 6pt; }
4: LIST { display: block; list-style-type: lower-roman; padding-top: 6pt;}
5: ITEM { display: block; padding-top: 6pt;text-indent: 10pt;}
6: UPDATE { display: block; text-align: center;  font-style: italic;
7:        border-top: 2pt solid black; padding-top: 6pt; padding-bottom: 6pt; }
8: WARNING { display: block; margin-left:24pt; color: red; padding-top: 6pt; }
```

Beachten Sie, daß es für jedes Element im Dokument, das Sie anzeigen wollen, eine Formatspezifikation geben muß. Wenn Sie eine weglassen oder ein Element vergessen, wird nichts angezeigt.

In der vorletzten Zeile von Listing 18.5 sehen Sie auch, daß ich eine `border-top`-Spezifikation in das `UPDATE`-Element eingefügt habe, um zu kompensieren, daß ich das HTML-Element `HR` entfernt habe.

Abbildung 18.3 zeigt die vollständige XML-Datei mit dem CSS-Stylesheet im IE5.

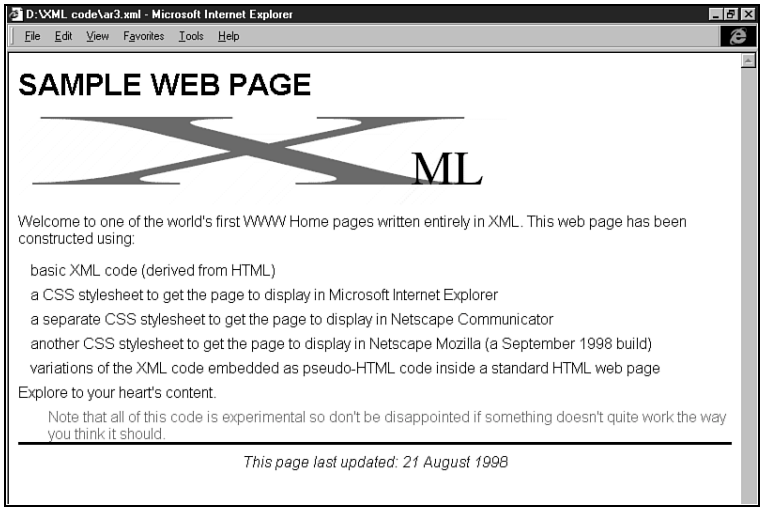


Abbildung 18.3: Das XML-Dokument mit dem CSS-Stylesheet im IE5.

Das Layout dieses XML-Dokuments kommt dem in Abbildung 18.1 gezeigten HTML-Layout sehr nahe (ich habe den Rahmen und die Schattierung für den Titel weggelassen, um den Code so einfach wie möglich zu halten). Es stimmt nicht perfekt überein, aber es kommt dem Ganzen schon sehr nahe. Leider unterstützt diese Version von IE5 CSS (noch) nicht, deshalb interpretiert der Browser die `display:list-item`-Spezifikation für das `ITEM`-Element auch nicht. Ich kann also die Listenelemente nicht mit den Bullets auszeichnen.

Die Änderungen legen den XML- und CSS-Code natürlich auf diesen Browser fest und sogar auf die verwendete Version, weil sich IE5 etwas anders verhält. Laden Sie diesen Code in den Netscape Communicator, werden Sie überhaupt nichts sehen. Wie in Abbildung 18.4 gezeigt, ergibt sich eine ganz ähnliche Anzeige zu der von IE5, wenn Sie den Code in Mozilla laden. Mozilla unterstützt einige der CSS-Eigenschaften nicht, deshalb wird die horizontale Linie unten auf der Seite nicht angezeigt.

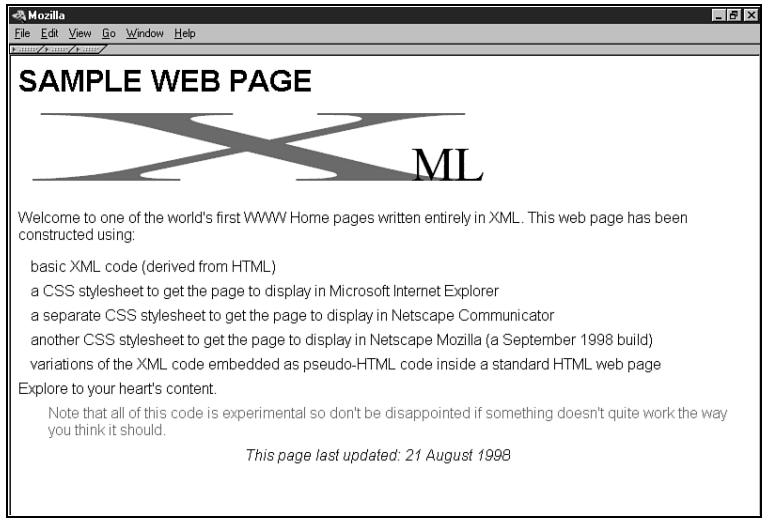


Abbildung 18.4: Das XML-Dokument aus dem IE5, angezeigt in Mozilla.

Trotz der Tatsache, daß wir XML-Code erzeugt haben, der weit vom Standard abweicht, handelt es sich dabei um eine sinnvolle Methode, XML darzustellen, ohne Code schreiben zu müssen, der ihn HTML-Elementen zuordnet (wie in Kapitel 12 beschrieben). Nachdem CSS vollständig unterstützt wird, ist die Lösung völlig korrekt. Jetzt betrachten wir die andere Seite: XML-Code in Mozilla.

XML, CSS und Mozilla

Der IE5 ist stabil und vollständig. Bei Mozilla ist das etwas ganz anderes. Netscape verfolgt die Strategie des öffentlichen Quellcodes, so daß jeder zu dem Code beitragen kann und die Ergebnisse wenig vorhersehbar sind. Die kompilierte Version im Internet und die Meinungen der Entwickler weisen jedoch darauf hin, daß Mozilla eine solidere XML-Unterstützung als Microsoft bietet.

Ich persönlich zweifle an dem Browser-Krieg zwischen Microsoft und Netscape. Ich glaube, Netscape verdient sehr viel mehr am Verkauf seiner Web-Server und der unterstützenden Software, als es je mit dem Verkauf von Browsern möglich gewesen wäre. Microsoft stellt den Internet Explorer kostenlos zur Verfügung, denn es war nicht der Browser-Markt, an dem man interessiert war – man wollte die Position von Windows als Internet-Plattform schützen. Darüber hinaus gibt es andere (und auch bessere) Browser, wie beispielsweise Opera, Amaya, Mosaic, Likse und Lynx, um nur ein paar wenige der bekannteren zu nennen. Aber auch wenn das nicht stimmt, besteht wenig Zweifel, daß Microsoft immer mehr Land gewinnt (was insbesondere in Anbetracht ihres Fehlstarts sehr überraschend ist).

Die Entwicklung des Netscape Communicator läuft parallel (der aktuelle offizielle Release ist 4.5), und niemand weiß, wann und ob Mozilla je zum offiziellen Release wird. Netscape verwendet die besten Komponenten seines öffentlichen Browsers und baut sie in das Mainstream-Produkt ein.

Wo Sie Mozilla finden

Leider ist das Mozilla-Programm ein riesiger Download, und Sie brauchen alle mögliche eigene Software, um überhaupt etwas mit dem Quellcode anfangen zu können. Aufgrund der Programmbibliotheken und aller Änderungen, die Sie am Code für Ihre jeweilige Situation vornehmen müssen, ist die Kompilierung nicht ganz einfach. Sie brauchen relativ viel Programmiererfahrung, viel Geduld, jede Menge Zeit und einen heldenhaften Willen. Keine Angst. Sie können sich auch vollständig kompilierte und ausführbare Programmdateien herunterladen (allerdings nur für Windows 95 und Windows NT). Es gibt vier Sites, die ich dazu empfehlen kann: `http://www.wynholds.com/mike/mozilla/`, `http://www.mozilla.org`, `http://www.mozillaZine.org` und `http://mozilla.hypermart.net/`.

(Die Großzügigkeit und Herzlichkeit einiger Menschen im Internet überrascht mich immer noch, nachdem ich 12 Jahre online bin.)

Die Programminstallation ist relativ einfach. Mit einigen kleineren Änderungen an den Profilen Ihres Netscape Communicator kann er glücklich neben anderen Browsern leben.

Anzeige von XML-Code in Mozilla

Ich will IE5 nicht mit Mozilla vergleichen. Wie Sie in den Kapiteln 10 und 12 bereits erfahren haben, verwenden die beiden Pakete unterschiedliche Ansätze und unterstützen unterschiedliche Funktionen. Nichtsdestotrotz ist es sinnvoll, XML-Code auf dieselbe Weise in Mozilla anzuzeigen wie in IE5. Wir beginnen mit einer HTML-Datei, die bereits ein formatiertes Aussehen aufweist, und versuchen, sie in Mozilla neu anzulegen.

Im ersten Schritt laden wir eine HTML-Datei und ein CSS-Stylesheet in Mozilla. Wir gehen dann von der resultierenden Anzeige aus (es wäre unvernünftig, vom XML-Code ein besseres Ergebnis zu erwarten als vom HTML-Code). Ich verwende dazu denselben Code wie aus der Übung für den IE5 (Listing 18.1 und 18.2). Das Ergebnis sehen Sie in Abbildung 18.5.

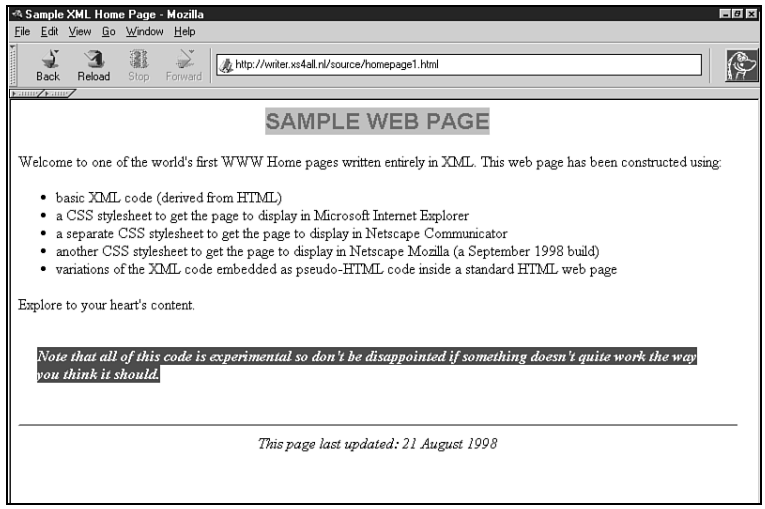


Abbildung 18.5: Die HTML-Zieldatei in Mozilla.

Wie Sie sehen, sieht derselbe HTML- und CSS-Code in Mozilla anders aus als im IE5, aber keiner davon ist zu 100 Prozent »korrekt«. Deshalb nehmen wir die Anzeige einfach hin, wie sie ist.

Im nächsten Schritt konvertieren wir den HTML-Code in XML-Code. Dazu ändern wir die Elementnamen, wie bereits im vorigen Beispiel gemacht. Anschließend laden wir diesen XML-Code in Mozilla, wie in Abbildung 18.6 gezeigt.

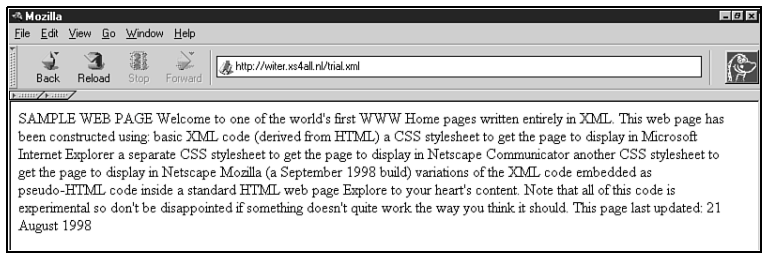


Abbildung 18.6: Die noch nicht fertige XML-Zieldatei in Mozilla

Im nächsten Schritt ordnen wir dem XML-Code Stile zu. In Mozilla ist dafür nicht im entferntesten so viel Aufwand erforderlich wie in IE5. Was Mozilla betrifft, ist ein Element ein Element (außer beim Parsing der Datei, wenn diese geladen wird), und Sie können die Formatspezifikationen für das XML-Element auf genau dieselbe Weise hinzufügen, wie Sie es für HTML-Elemente machen würden. Jetzt betrachten wir noch einmal die XML-Eingabe, die Sie in Listing 18.6 sehen.

Listing 18.6: Die XML-Rohdatei

```
1:  <PAGE>
2:  <TITLE>SAMPLE WEB PAGE</TITLE>
3:  <PARA>Welcome to one of the world's first WWW Home pages
4:  written entirely in XML. This web page has been
5:  constructed using:</PARA>
6:  <LIST>
7:  <ITEM>basic XML code (derived from HTML)</ITEM>
8:  <ITEM>a CSS stylesheet to get the page to display in
9:  Microsoft Internet Explorer</ITEM>
10: <ITEM>a separate CSS stylesheet to get the page to display
11: in Netscape Communicator</ITEM>
12: <ITEM>another CSS stylesheet to get the page to display
13: in Netscape Mozilla (a September 1998 build)</ITEM>
14: <ITEM>variations of the XML code embedded as pseudo-HTML
15: code inside a standard HTML web page</ITEM>
16: </LIST>
17: <PARA>Explore to your heart's content.</PARA>
18: <WARNING>Note that all of this code is experimental so
19: don't be disappointed if something doesn't quite work the
20: way you think it should.</WARNING>
21: <UPDATE>This page last updated: 21 August 1998</UPDATE>
22: </PAGE>
```

Sie haben vielleicht noch nicht bemerkt, daß in der XML-Rohdatei etwas fehlt (und falls doch, dann verdienen Sie meine Hochachtung!). Es gibt keine XML-Deklaration. Das ist kein Problem, wenn Sie den XML-Code in HTML-Code einhüllen, und sehr wahrscheinlich nimmt es Ihnen Mozilla nicht einmal übel, wenn Sie sie weglassen. Genauer gesagt, ist sie nicht zwingend erforderlich, und Sie brauchen sie nicht, es sei denn, Sie verwenden nicht den Standardzeichensatz – dennoch sollten Sie sich angewöhnen, sie bereitzustellen. In einem HTML-Kontext würde die XML-Deklaration als Standard-SGML-Verarbeitungsanweisung behandelt und ignoriert. Es schadet also nicht, sie anzugeben.

Jetzt fügen wir die XML-Deklaration ein, ebenso wie eine weitere Verarbeitungsanweisung, die das CSS-Stylesheet mit dem XML-Code verknüpft. Beachten Sie, daß sich die Syntax dafür von der Vorgehensweise in HTML unterscheidet, wie in Listing 18.7 gezeigt.

Listing 18.7: Die vollständige XML-Datei

```
1:  <?xml version="1.0"?>
2:  <?xml:stylesheet type="text/css2" href="xmlmoz.css"?>
3:  <PAGE>
4:  <TITLE>SAMPLE WEB PAGE</TITLE>
5:  <PARA>Welcome to one of the world's first WWW Home
6:  pages written entirely in XML. This web page has been
7:  constructed using:</PARA>
8:  <ITEMST>
9:  <ITEM>basic XML code (derived from HTML)</ITEM>
10: <ITEM>a CSS stylesheet to get the page to display in
11: Microsoft Internet Explorer</ITEM>
12: <ITEM>a separate CSS stylesheet to get the page to
13: display in Netscape Communicator</ITEM>
14: <ITEM>another CSS stylesheet to get the page to
15: display in Netscape Mozilla (a September 1998 build)</ITEM>
16: <ITEM>variations of the XML code embedded as
17: pseudo-HTML code inside a standard HTML web page</ITEM>
18: </LIST>
19: <PARA>Explore to your heart's content.</PARA>
```

```
20: <WARNING>Note that all of this code is experimental so
21: don't be disappointed if something doesn't quite work the way you
22: think it should.</WARNING>
23: <UPDATE>This page last updated: 21 August 1998</UPDATE>
24: </PAGE>
```

Wenn Sie diesen Code in Mozilla laden, sehen Sie ein ähnliches Ergebnis wie in Abbildung 18.7 gezeigt.

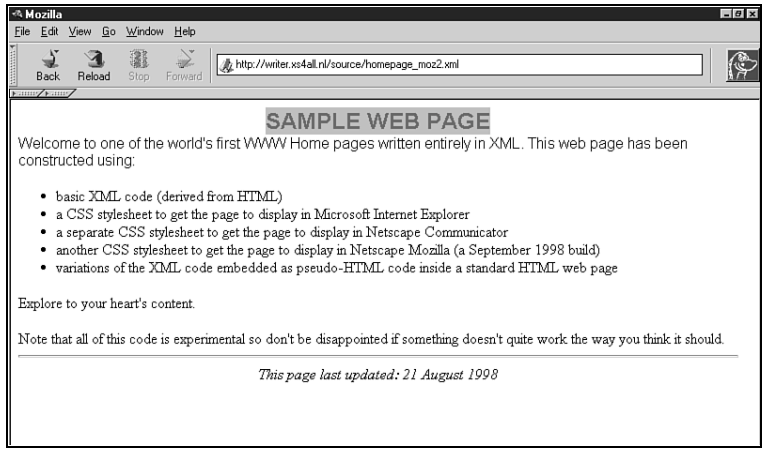


Abbildung 18.7: Die endgültige XML-Datei in Mozilla.

Wie Sie sehen, unterstützt Mozilla CSS etwas weniger als IE5. Ich glaube jedoch, daß es auch in Mozilla bald eine vollständige CSS-Unterstützung gibt (insbesondere weil der Netscape Communicator CSS unterstützt). Wenn man überlegt, wie wenige Änderungen am XML-Code erforderlich waren (die alle korrektes XML darstellen), scheint es kein weiter Weg mehr sein, mit CSS XML in einem Netscape-Browser so anzuzeigen, als handele es sich um HTML.

Tricks

Bisher habe ich Ihnen eine Methode gezeigt, Ihren XML-Code mehr oder weniger zu ändern, um ihn in einem Browser anzuzeigen, und eine andere Methode, Ihren XML-Code beizubehalten, um ihn auf andere Weise in einem anderen Browser anzuzeigen. Die Qual der Wahl!

Bevor ich das Thema der CSS-Stile verlasse und mich den »echten« Stilsprachen zuwende, will ich Ihnen einen praktischen Trick verraten, der gut funktioniert, wenn Sie bei der Namensvergabe Ihrer Elemente genügend Freiraum besitzen. Ich spreche hier von einem Trick, weil es einer ist. Ich nutze eine der Schwächen des Browsers aus, um ihm den Eindruck zu verschaffen, er sähe kein XML, sondern HTML.

Es gibt keine geheimen Codierungen und auch sonst keine komplizierten Verrenkungen – der Trick ist, die existierenden HTML-Elementnamen zu nutzen. Eine Liste ist eine Liste ist eine Liste, egal wie man sie betrachtet. Es könnte sich um eine Teilliste oder um eine Preisliste handeln, aber durch sorgfältige Überlegung, wie die Elemente heißen sollen und wie Sie sie verschachteln (und vergessen Sie nicht, Attribute zu verwenden, um Informationen über Elemente einzufügen), ist es durchaus möglich, HTML-Elemente zu verwenden. Vergleichen Sie die beiden folgenden Codeabschnitte, und zwar

```
<price.list>
<item>shoe, black</item>
<item>shoe, brown</item>
</price.list>
```

mit

```
<price.list>
<ul>
<li> shoe, black</item>
<item>shoe, brown</item>
</ul>
</price.list>
```

Ein großer Unterschied? Nicht wirklich. Statt XML-Code in HTML-Code einzuhüllen, kombinieren wir beide einfach. Jeder Browser, der HTML versteht, erkennt auch die HTML-Elemente und kann sie darstellen. Sie brauchen sich dann nur noch um die restlichen XML-Elemente zu kümmern.

Funktioniert das? Nicht wirklich, aber dann funktioniert es auch nicht schlechter als »reines« XML, und manchmal funktioniert es sehr viel besser. Betrachten Sie ein Beispiel für eine solche Mischung, wie in Listing 18.8 gezeigt.

Die Beta-Preview-Version 1 des Internet Explorer 5 unterstützte diese Hybridformen auch, aber mit Einführung der besseren XSL-Unterstützung ist diese Funktion aus Preview Version 2 verschwunden.

Listing 18.8: Eine Hybrid-XML-Datei für Mozilla

```
1:  <?xml version="1.0"?>
2:  <?xml:stylesheet type="text/css2" href="samplexmlmoz1.css"?>
3:  <page>
4:  <maintitle>SAMPLE WEB PAGE</maintitle>
5:  <p>Welcome to one of the world's first WWW Home pages
6:  written entirely in XML. This web page has been
7:  constructed using:</p>
8:  <UL>
9:  <LI>basic XML code (derived from HTML)</LI>
10: <LI>a CSS stylesheet to get the page to display in
11: Microsoft Internet Explorer</LI>
12: <LI>a separate CSS stylesheet to get the page to
13: display in Netscape Communicator</LI>
14: <LI>another CSS stylesheet to get the page to
15: display in Netscape Mozilla (a September 1998 build)</LI>
16: <LI>variations of the XML code embedded as
17: pseudo-HTML code inside a standard HTML web page</LI>
18: </UL>
19: <P>Explore to your heart's content.</P>
20: <warning>Note that all of this code is
21: experimental so don't be disappointed if something doesn't quite
22: work the way you think it should.</warning>
23: <UPDATE>This page last updated: 21 August 1998</UPDATE>
24: </page>
```

Der Code unterscheidet sich nicht wesentlich. Sie genießen große Vorteile, weil Sie in den CSS-Stylesheets nur das Format für die XML-Elemente und die HTML-Elemente festlegen müssen, für die das Standardlayout nicht tauglich ist. Sie können sich einfach auf den Browser verlassen, daß er diese HTML-Elemente so gut wie möglich darstellt.

Diese Mischung der HTML-Elemente ist eine praktische Lösung, wenn der Browser eine bestimmte CSS-Eigenschaft nicht unterstützt, wie es auch in Mozilla der Fall ist.

So weit zum Code – jetzt wollen wir Bilder sehen! Abbildung 18.8 ist die Mozilla-Hybrid-Datei, dargestellt in Mozilla. Urteilen Sie selbst, und vergleichen Sie die Ergebnisse mit Abbildung 18.3 (reines XML in IE5) und Abbildung 18.7 (reines XML in Mozilla).

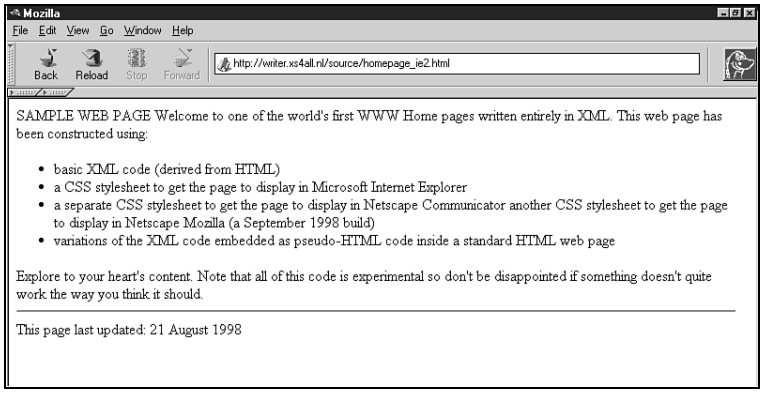


Abbildung 18.8: Die Hybrid-XML-Datei in Mozilla.

Wenn Sie die Ausgaben vergleichen, sehen Sie, wie weit Sie XML beibehalten können, wenn Ihr Hauptanliegen ist, eine Anzeige in einem Web-Browser vorzunehmen. Größtenteils liegt es an der Leistungsfähigkeit des Browsers und an Ihrer Entschlossenheit, eine gute Darstellung zu erzielen.

Manchmal glaube ich, man könnte sehr viel mit HTML machen, ohne daß man XML überhaupt braucht. HTML (insbesondere in der neuesten Version, 4) bietet Funktionen wie DIV-Elemente, Elementklassen und Attribute, die Ihnen erlauben, fast so viel beschreibende Information einzufügen, wie es in XML mit den Elementen möglich ist. Betrachten Sie die reine (semantische) HTML-Variante der XML-Datei, mit der wir in diesem Kapitel gearbeitet haben (siehe Listing 18.9).

Listing 18.9: Semantischer HTML-Code

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2:  <HTML>
3:  <HEAD>
4:     <LINK rel="stylesheet" href="SAMPLEXML.css">
```

```
5:     <TITLE>Sample XML Home Page</TITLE>
6:  </HEAD>
7:  <BODY>
8:  <DIV CLASS="PAGE">
9:  <H1 CLASS="PageTitle">SAMPLE WEB PAGE</H1>
10: <P CLASS="first">Welcome to one of the world's first
11: WWW Home pages written entirely in XML. This web page has
12: been constructed using:</p>
13: <UL CLASS="list">
14: <LI CLASS="item1">basic XML code (derived from HTML)</LI>
15: <LI CLASS="item2">a CSS stylesheet to get the page to display
16: in Microsoft Internet Explorer</LI>
17: <LI CLASS="item3">a separate CSS stylesheet to get the page to
18: display in Netscape Communicator</LI>
19: <LI CLASS="item4">another CSS stylesheet to get the page to
20: display in Netscape Mozilla (a September 1998 build)</LI>
21: <LI CLASS="last">variations of the XML code embedded as
22: pseudo-HTML code inside a standard HTML web page</LI>
23: </UL>
24: <P>Explore to your heart's content.</P>
25: <P CLASS="WARNING">Note that all of this code is experimental
26: so don't be disappointed if something doesn't quite work the
27: way you think it should.</P>
28: <HR NOSHADE WIDTH="100%">
29: <P CLASS="update">This page last updated: 21 August 1998</P>
30: </DIV>
31: </BODY>
32: </HTML>
```

Das ist kein wirklich fairer Vergleich. Das CSS-Stylesheet für eine solche Darstellung müßte riesig sein, und natürlich würde ich dadurch alle zusätzlichen Funktionen von XML aufgeben, beispielsweise Linking, Adressierung und Auswertung. Wenn Sie jedoch nur semantische Information brauchen, kann HTML fast genauso praktisch wie XML sein.

Einbettung von CSS in XSL

In Kapitel 20 erfahren Sie mehr über XSL. Zuvor möchte ich Ihnen jedoch zeigen, wie Sie mit einem Minimum an XSL-Kenntnissen CSS-Eigenschaften in ein XSL-Stylesheet einbetten können.

Es gibt zwar eine Anmerkung vom W3C (`http://www.w3.org/TR/NOTE-XSL-` und `-CSS.html`), die vorschlägt, wie CSS- und XSL-Code kombiniert werden sollen, aber im Internet Explorer 5 Beta Preview 2 ist bereits eine andere Methode implementiert.

Listing 18.10 zeigt das oben vorgestellte XML-Codebeispiel, das so abgeändert wurde, daß es ein XSL-Stylesheet verwendet.

Listing 18.10: Der XML-Code wurde so abgeändert, daß er ein XSL-Stylesheet verwendet

```
1:  <?xml:stylesheet href="css.xsl" type="text/xsl"?>
2:  <PAGE>
3:  <TITLE>SAMPLE WEB PAGE</TITLE>
4:  <html:IMG xmlns:html="htmluri" SRC="logo.gif" ALIGN="middle"/>
5:  <PARA>Welcome to one of the world's first WWW Home
6:  pages written entirely in XML. This web page has been
7:  constructed using:</PARA>
8:  <LIST>
9:  <ITEM>basic XML code (derived from HTML)</ITEM>
10: <ITEM>a CSS stylesheet to get the page to display in
11: Microsoft Internet Explorer</ITEM>
12: <ITEM>a separate CSS stylesheet to get the page to
13: display in Netscape Communicator</ITEM>
14: <ITEM>another CSS stylesheet to get the page to
15: display in Netscape Mozilla (a September 1998 build)</ITEM>
16: <ITEM>variations of the XML code embedded as
17: pseudo-HTML code inside a standard HTML web page</ITEM>
18: </LIST>
19: <PARA>Explore to your heart's content.</PARA>
20: <WARNING>Note that all of this code is experimental so
21: don't be disappointed if something doesn't quite work the way you
22: think it should.</WARNING>
23: <UPDATE>This page last updated: 21 August 1998</UPDATE>
24: </PAGE>
```

Alle Änderungen erfolgen im Stylesheet, wie in Listing 18.11 gezeigt.

Listing 18.11: Das XSL-Stylesheet mit dem CSS-Code

```
1: <?xml version="1.0"?>
2: <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
3:
4:   <xsl:template>
5:     <xsl:value-of/>
```

```
6:   </xsl:template>
7:
8:   <xsl:template match="/">
9:     <HTML>
10:      <HEAD>
11:        <TITLE>
12:          Sample Web Page
13:        </TITLE>
14:        <STYLE>
15:          <![CDATA[
16:            LI  { margin-left:  24pt; margin-right: 24pt }
17:          ]]>
18:        </STYLE>
19:      </HEAD>
20:      <BODY BGCOLOR="#FFFFFF" TEXT="#000000">
21:         <xsl:for-each select="PAGE">
22:          <xsl:apply-templates/>
23:        </xsl:for-each>
24:      </BODY>
25:    </HTML>
26:  </xsl:template>
27:
28:  <xsl:template match="PAGE/TITLE">
29:    <H1>
30:      <xsl:apply-templates/>
31:    </H1>
32:  </xsl:template>
33:
34:  <xsl:template match="PARA">
35:    <P>
36:      <xsl:apply-templates/>
37:    </P>
38:  </xsl:template>
39:
40:  <xsl:template match="LIST">
41:    <UL>
42:      <xsl:for-each select="ITEM">
43:        <LI>
44:          <xsl:apply-templates/>
45:        </LI>
46:      </xsl:for-each>
47:    </UL>
48:  </xsl:template>
49:
50:  <xsl:template match="WARNING">
51:    <P STYLE="margin-left:24pt; color: red;">
```

```
52:      <xsl:apply-templates/>
53:    </P>
54:  </xsl:template>
55:
56:  <xsl:template match="UPDATE">
57:    <HR/>
58:      <ADDRESS STYLE="text-align: center">
59:        <xsl:apply-templates/>
60:      </ADDRESS>
61:  </xsl:template>
62:
63:</xsl:stylesheet>
```

Beachten Sie, daß es sich bei XSL um ein XML-Dokument handelt, es muß also wohlgeformt sein, damit der Browser es akzeptiert.

Im Laufe dieses Buchs werden Sie noch mehr über die XSL-Syntax erfahren. Hier brauchen Sie nur zu wissen, daß Sie für jedes darzustellende XML-Element mehrere XSL-Anweisungen brauchen, die es dem entsprechenden HTML-Element zuweisen:

```
<xsl:template match="PARA">
  <P>
    <xsl:apply-templates/>
  </P>
</xsl:template>
```

Wenn Sie ein Element in einem anderen Element explizit auswählen wollen, verwenden Sie die Eltern/Kind-Methode, so daß `LIST/ITEM` sich nur auf `ITEM`-Elemente bezieht, die innerhalb von `LIST`-Elementen erscheinen. XSL verwendet immer die genaueste Übereinstimmung, so daß keines der anderen `ITEM`-Elemente davon beeinflußt wird.

Wenn Sie einem Element Stileigenschaften zuweisen wollen, gehen Sie dafür genau wie im »normalen« CSS-Code vor, indem Sie sie in das Element-Start-Tag schreiben, wie in den Zeilen 51 und 58 gezeigt.

Wenn Sie globale Stile anwenden wollen, legen Sie sie in einem CDATA-Abschnitt an, wo der Code vor dem XML-Prozessor verborgen wird, wie in den Zeilen 14 bis 18 gezeigt.

Das Ergebnis ist fast perfekt, wie in Abbildung 18.9 gezeigt. Weil wir jedes XML-Element einem HTML-Element zuordnen, erlaubt Ihnen der Browser, auf die von ihm unterstützten CSS-Eigenschaften zuzugreifen.

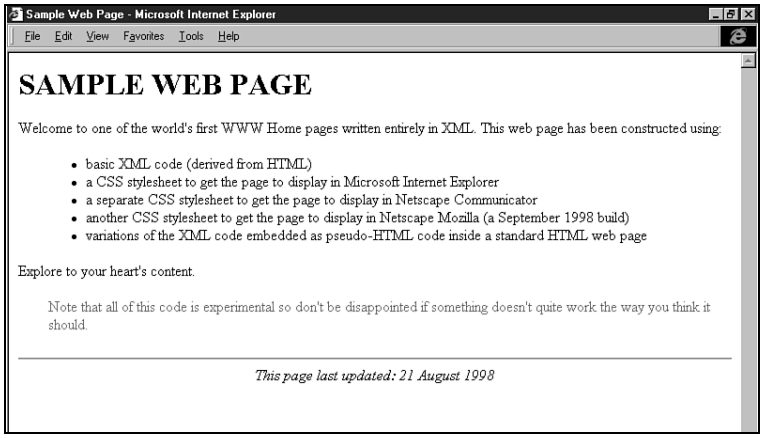


Abbildung 18.9: Die XML-Datei in IE5 verwendet CSS in XSL.

Der in Listing 18.11 gezeigte XSL-Code ist nicht derselbe wie der im XSL-Proposal beschriebene. Offensichtlich hat Microsoft bereits Änderungen der XSL-Syntax implementiert, die noch gar nicht veröffentlicht wurden. Diese Syntax wird aktuell in IE5 Beta Preview 2 unterstützt, aber man kann nicht vorhersagen, wie sich die XSL-Syntax letztlich ändert und ob sie in IE5 oder einem anderen Web-Browser unterstützt wird.

CSS-Stylesheet-Eigenschaften

Sie haben bereits erfahren, wie man CSS-Stylesheets in XML einsetzt, aber ich habe noch nicht viel über CSS selbst gesagt. Aber das werde ich auch hier nicht tun. Es gibt viele gute Bücher zu diesem Thema. Darüber hinaus finden Sie zahllose Informationen im Internet darüber (insbesondere wenn Sie die Tips nutzen, die ich Ihnen am Anfang dieses Kapitels gegeben habe). Egal wieviel Sie über CSS lesen, Sie werden schnell erkennen, daß es keine zwei Browser gibt, die es gleich interpretieren. Der beste Kompromiß kann häufig nur durch Trial&Error ermittelt werden. Die folgenden Abschnitte sollen Ihnen als Abschluß dieses Kapitels einen kurzen Überblick über die CSS1-Eigenschaften bieten, und ich sage es noch einmal: Probieren Sie sie aus!

Einheiten

Abstands- und Größeneinheiten können als Absolutwerte angegeben werden, die unabhängig von ihrem Kontext immer gleich bleiben:

```
Width: 50px;
margin-left: 2em;
```

Die folgende Auflistung zeigt die verfügbaren absoluten Einheiten. Beachten Sie, daß zwischen Wert und Einheit kein Leerzeichen angegeben wird:

- `cm` (Zentimeter; `1cm` = `10mm`)
- `in` (Inch; `1in` = `2.54cm`)
- `mm` (Millimeter)
- `pc` (Pica; `1pc` = `12pt`)
- `pt` (Punkt; `1pt`= `1/72 in`)

Diese Einheiten können als Prozentwerte angegeben werden, um relative Größen zu spezifizieren, die abhängig von ihrem Kontext berechnet werden (die Größe des Browser-Fensters, die Größe der Tabelle usw.):

```
width: 8ß%;
```

Die folgende Auflistung beschreibt die verfügbaren relativen Einheiten. Beachten Sie auch hier, daß zwischen Wert und Einheit kein Leerzeichen angegeben wird:

- `em` (em, die Höhe der Schrift für das Element)
- `ex` (x-Höhe, die Höhe des Buchstabens x)
- `px` (Pixel, abhängig von der Bildschirmauflösung; 72 dpi ist eine durchschnittliche EGA-/VGA-Bildschirmauflösung mit 72 Pixel pro Inch)

Einige Eigenschaften verwenden Farbeinheiten zur Definition von Farben. Diese Farben werden nach demselben Hexadezimalcodesystem wie in HTML angegeben. Es besteht aus dem Pfundsymbol (#), gefolgt von hexadezimalen Zeichen (00 bis FF), `#FF0000` für Rot, `#00FF00` für Grün und `#0000FF` für Blau. Dabei ist 00 nichts und FF der maximale Farbwert. Sie können auch Prozentzahlen für Rot, Grün und Blau eingeben, aber die einfachste Methode ist, einfach die gewünschte Farbe festzulegen:

```
color: #FF0000;
color: rgb(100%,0%,0%);
```

Angabe von CSS-Eingeschaften

Eine Eigenschaft wird angegeben durch den Namen des betreffenden Elements, gefolgt von den Eigenschaftsnamen und Werten:

```
Element {eigenschaft1: wert1; eigenschaft2: wert2}
```

Ein Beispiel:

```
BODY { margin: 0;
       font-family: arial, helvetica, sans-serif;
       font-size: 14px;
       color: black; }
```

Wenn Sie für mehrere Elemente dieselben Eigenschaften gleichzeitig setzen wollen, gruppieren Sie sie innerhalb einer einzigen Stildeklaration:

```
H1, H2, H3, H5 {color: red;}
```

Wenn Sie noch weiter gehen und Eigenschaften auf ein Element anwenden wollen, falls es sich innerhalb eines anderen Elements befindet, gruppieren Sie die Elemente in einer einzigen Stildeklaration, lassen aber das trennende Komma weg, z.B.:

```
H1 B { color: red; }
```

Der Text innerhalb eines `B`-Elements wird nur dann rot angezeigt, wenn das `B`-Element im `H1`-Element enthalten ist:

```
<DOC>
<H1>This part will be <B>very red</B></H1>
<P>But this part will be <B>totally unaffected</B>
by the declaration.</P>
</DOC>
```

Klassen

Mit Hilfe von Klassenselektoren definieren Sie Elementtypen, keine benannten Elemente. Ein Klassenselektor ist eine Zeichenkette, der ein Punkt vorausgeht. Er wird unter Verwendung des STYLE-Attributs aufgerufen.

```
.warning {color: red;}
     <H1 class="warning">WARNING!</H1>
     <P class="warning">Danger, Will Robinson!</P>
```

ID-Attribute

Ein ID-Selektor wählt ein bestimmtes Element nach dem Wert seines ID-Attributs aus. Ein ID-Selektor ist der ID-Attribut-String, dem ein Pfundzeichen (#) vorausgeht. Im Wert des ID-Attributs erscheint dieses Pfundzeichen nicht.

```
#i5 {color: brown;}
     <P ID="i5">This is text with an ID of 'i5'.</P>
```

Überblick über die CSS1-Eigenschaften

Hier folgt ein kurzer Überblick über die CSS1-Eigenschaften, die Elementen zugeordnet werden können:

- `background` – die Abkürzung für alle anderen Hintergrundeigenschaften. Die Werte können in beliebiger Reihenfolge angegeben werden:

  ```
  BODY {background: white url(bg41.gif) fixed center;}
  ```

- `background-attachment` - Diese Eigenschaft gibt an, ob das Hintergrundbild mit dem Element weitergeschoben werden soll. Im allgemeinen wird es nur auf das Wurzelelement angewendet (oder das BODY-Element in einem HTML-Dokument), weil es für die meisten anderen Elemente wenig Sinn ergibt. Erlaubte Werte sind scroll und fixed.

  ```
  BODY {background-attachment: scroll;}
  ```

- `background-color` - Diese Eigenschaft setzt die Hintergrundfarbe eines Elements. Diese Hintergrundfarbe geht bis zum Rahmen des Elements.

  ```
  H4 {background-color: white;}
  ```

- `background-image` - Diese Eigenschaft legt ein Bild als Hintergrundmuster fest. In Verbindung mit den anderen Hintergrundeigenschaften kann das Bild kachelförmig angeordnet oder nur in einer Richtung wiederholt werden.

  ```
  BODY {background-image: url(../grafix/steel1.gif);}
  ```

- `background-position` - Diese Eigenschaft setzt die Anfangsposition der Hintergrundfarbe oder des Bildes. Gibt der Eigenschaftswert eine Farbe an, wird die Farbe von dieser Position aus gefüllt. Ist der Eigenschaftswert ein Bild, wird das erste Bild an der betreffenden Position angelegt. Die Wiederholrate wird durch `background-repeat` festgelegt. Mögliche Werte sind `top`, `center`, `bottom` für die vertikale Position, und `left`, `center`, `right` für die horizontale Position, eine Länge oder einen Prozentwert.

  ```
  BODY {background-position: top center;}
  ```

- `background-repeat` - Diese Eigenschaft gibt an, wie ein Hintergrundbild wiederholt wird. Erlaubte Werte sind `repeat`, `repeat-x`, `repeat-y` und `no-repeat`.

```
BODY {background-repeat: no-repeat;}
```

- `border` - Dies ist eine Abkürzung, die die Breite, (siehe border-width), Farbe (siehe border-color) und den Stil (siehe `border-style`) der Rahmen auf allen Seiten eines Elements angibt.

```
H1 {border: 2px dashed tan;}
```

- `border-bottom` - Dies ist eine Abkürzung, die die Breite, die Farbe und den Stil des unteren Rahmens eines Elements angibt.

```
UL {border-bottom: 0.5in grooved green;}
```

- `border-bottom-width` - Diese Eigenschaft setzt die Breite des unteren Rahmens eines Elements. Der Rahmen erbt den Hintergrund des Elements, kann aber einen eigenen Vordergrund haben (siehe `border-style`). Negative Werte sind nicht erlaubt. Erlaubt sind `thin`, `medium`, `thick` oder eine Länge.

```
UL {border-bottom-width: 1in;}
```

- `border-color` - Diese Eigenschaft setzt die Vordergrundfarbe des Rahmens auf allen Seiten eines Elements (siehe `border-style`). Der Rahmen erbt den Hintergrund des Elements.

```
H1 {border-color: blue; border-style: solid;}
```

- `border-left` - Dies ist eine Abkürzung, die die Breite, die Farbe und den Stil des linken Rahmens eines Elements angibt.

```
P {border-left: 3em solid gray;}
```

- `border-left-width` - Diese Eigenschaft setzt die Breite des linken Rahmens eines Elements. Der Rahmen erbt den Hintergrund des Elements, kann aber einen eigenen Vordergrund haben (siehe `border-style`). Negative Werte sind nicht erlaubt. Erlaubt sind `thin`, `medium`, `thick` oder eine Länge.

```
P {border-left-width: 3em;}
```

- `border-right` - Dies ist eine Abkürzung, die die Breite, die Farbe und den Stil des rechten Rahmens eines Elements angibt.

```
IMG {border-right: 30px dotted blue;}
```

- `border-right-width` - Diese Eigenschaft setzt die Breite des linken Rahmens eines Elements. Der Rahmen erbt den Hintergrund des Elements, kann aber einen eigenen Vordergrund haben (siehe `border-style`). Negative Werte sind nicht erlaubt. Erlaubt sind `thin`, `medium`, `thick` oder eine Länge.

```
IMG {border-right-width: .5in;}
```

- `border-style` - Diese Eigenschaft setzt den Stil des Rahmens auf allen Seiten des Elements. Erlaubte Werte sind `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset` und `outset`. Die Farbe wird durch `border-color` festgelegt.

```
H1 {border-style: solid; border-color: red;}
```

- `border-top` - Dies ist eine Abkürzung, die die Breite, die Farbe und den Stil des oberen Rahmens eines Elements angibt.

```
UL {border-top: 0.5in solid black;}
```

- `border-top-width` - Diese Eigenschaft setzt die Breite des linken Rahmens eines Elements. Der Rahmen erbt den Hintergrund des Elements, kann aber einen eigenen Vordergrund haben (siehe `border-style`). Negative Werte sind nicht erlaubt. Erlaubt sind thin, medium, thick oder eine Länge.

```
UL {border-top-width: 0.5in;}
```

- `border-width` - Diese Eigenschaft setzt die Breite aller Rahmen eines Elements. Der Rahmen erbt den Hintergrund des Elements, kann aber einen eigenen Vordergrund haben (siehe `border-style`). Negative Werte sind nicht erlaubt. Erlaubt sind thin, medium, thick oder eine Länge.

```
H1 {border-width: 2ex;}
```

- `clear` - Diese Eigenschaft gibt an, welche Floating-Elemente (falls vorhanden) neben dem Element positioniert werden können (siehe auch `float`). Mögliche Werte sind `left`, `right`, `both` und `none`.

```
H1 {clear: both;}
```

- `color` - Diese Eigenschaft setzt die Farbe eines bestimmten Elements. Für Text wird die Textfarbe gesetzt, für andere Elemente, wie beispielsweise eine horizontale Linie (HR), die Vordergrundfarbe.

```
STRONG {color: red;}
```

- `display` - Diese Eigenschaft ordnet Elemente in allgemeine Kategorien ein. Der gebräuchlichste Wert ist none, aber es ist sinnvoll, wenn er auf list-item gesetzt ist (siehe `list-style`-Eigenschaften). Erlaubte Werte sind `block`, `inline`, `list-item` und `none`.

```
.hide {display: none;}
```

- `first-letter` - Diese Eigenschaft bezieht sich auf den ersten Buchstaben in einem Element. Mit Hilfe dieser Eigenschaft können Initialien hervorgehoben werden. Sie sollte nur auf Elemente auf Blockebene angewendet werden.

```
P:first-letter {color: red;}
<P>The letter 'T' at the beginning of this element is red.</P>
```

- `first-line` - Dies ist eine Eigenschaft, die auf die erste Zeile des Texts in einem Element angewendet wird. Wenn Sie die Fenstergröße ändern und die erste Zeile in eine zweite Zeile umbrochen wird, bleiben die Zeileneigenschaften erhalten. Diese Eigenschaft sollte nur auf Elemente auf Blockebene angewendet werden.

```
P:first-line {color: red;}
<P>The first line of this element will be red.
The second and other lines will be unaffected.</P>
```

- `float` - Diese Eigenschaft setzt die Float-Charakteristik für ein Element (siehe auch `clear`). Sie wird häufig auf Bilder angewendet, so daß Text um sie herumfließen kann, sie kann aber für jedes beliebige Element gesetzt werden. Mögliche Werte sind `left`, `right` und `none`.

```
IMG {float: left;}
```

- `font` - Dies ist eine Abkürzung für alle anderen Schrifteigenschaften. Die Reiehnfolge der Werte ist relevant und muß wie folgt aussehen:

```
font {font-style font-variant font-weight font-size/
➥line-height font-family;}
```

Alle diese Eigenschaften können auch weggelassen werden. Neben einer bestimmten Schrift können Sie auch eine generische Familie angeben, `serif` (Times), `sans-serif` (Arial oder Helvetica), `cursive` (Zapf-Chancery), `fantasy` (Western) oder `monospace` (Courier).

```
P {font: bold 12pt/14pt Helvetica,sans-serif;}
```

- `font-family` - Diese Eigenschaft deklariert eine bestimmte Schrift, eine generische Schriftfamilie oder beides. Um auch Systeme abzudecken, wo eine bestimmte Schriftart möglicherweise nicht vorhanden ist, geben Sie mehrere Schriften an, deren Priorität von links nach rechts abnimmt.

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

- `font-size` - Diese Eigenschaft setzt die Größe der Schrift. Das kann eine absolute Angabe (`xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` oder `xx-large`) sein, eine relative Größe (`larger` oder `smaller`), eine Länge oder ein Prozentwert.

```
H2 {font-size: 200%;}
H3 {font-size: 36pt;}
```

- `font-style` - Diese Eigenschaft bestimmt den Zeichenstil. Mögliche Werte sind `italic`, `oblique` und `normal`.

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

- `font-variant` - In CSS1 kann diese Eigenschaft zwei Werte annehmen: `small-caps` und `normal`.

```
H3 {font-variant: small-caps;}
```

- font-weight - Diese Eigenschaft setzt die Gewichtung einer Schrift, so daß sie dicker oder dünner wird. Mögliche Werte sind normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800 und 900. Die Namen sind relativ, die numerischen Werte absolut, aber weil nicht alle Schriften neun Gewichtungen besitzen, wird möglicherweise der nächstliegende Wert verwendet.

```
B {font-weight: 500;}
```

- height - Diese Eigenschaft bestimmt die Höhe eines Elements. Sie wird im allgemeinen auf Bilder angewendet, kann aber für alle Elemente auf Blockebene oder ersetzte Elemente angewendet werden. Negative Werte sind nicht erlaubt. Die Höhe kann als Länge, Prozentwert oder mit dem Schlüsselwert auto (Standard) angegeben werden.

```
IMG.icon {height: 50px;}
```

- important - Diese Eigenschaft gibt an, daß eine Stilfestlegung wichtig ist. Wichtige Spezifikationen überschreiben alle anderen Deklarationen, unabhängig von deren Ursprung oder Beschaffenheit.

```
H1 {color: red ! important;}
```

- letter-spacing - Diese Eigenschaft gibt an, wieviel Whitespace zwischen (darstellbaren) Zeichen erscheinen soll. Mögliche Werte sind normal oder eine Länge.

```
P {letter-spacing: 0.5em;}
```

- line-height - Diese Eigenschaft bestimmt den vertikalen Abstand zwischen den Grundlinien in einem Element. Negative Werte sind nicht erlaubt. Mögliche Werte sind normal, eine Zahl (die mit der Zahlengröße der Schrift multipliziert wird, siehe font-size), eine Länge oder ein Prozentwert.

```
P {line-height: 18pt;}
H2 {line-height: 200%;}
```

- list-style - Dies ist eine Abkürzung für alle anderen list-style-Eigenschaften. Die Werte beziehen sich auf alle Elemente mit einem display-Wert von list-item (siehe display).

```
UL {list-style: square url(sqbullet.gif) outer;}
```

- list-style-image - Diese Eigenschaft gibt an, welche Kennzeichnung für eine sortierte oder nicht sortierte Liste verwendet werden soll. Dieser Wert bezieht sich auf Elemente mit dem display-Wert list-item.

```
UL {list-style-image: url(bullet3.gif);}
```

- list-style-position - Diese Eigenschaft gibt die Position der Markierung oder der Ziffer einer sortierten oder nicht sortierten Liste an. Der Wert bezieht sich auf Elemente mit einem display-Wert von list-item. Mögliche Werte sind inside

oder `outside`, wobei `outside` der Standard ist. Wird `inside` verwendet, wird die Zeile unterhalb des Markierungszeichens fortgesetzt, statt eingerückt zu erscheinen.

```
LI {list-style-position: outside;}
```

- `list-style-type` - Diese Eigenschaft gibt die Markierung oder das Numerierungssystem für nicht sortierte oder sortierte Listen an. Der Wert bezieht sich auf Elemente mit einem `display`-Wert von `list-item`. Mögliche Werte sind `disc`, `circle`, `square`, `decimal` (1,2,3), `lower-roman` (i,ii,iii) `upper-roman` (I,II,III), `lower-alpha` (a,b,c), `upper-alpha` (A,B,C) und `none`.

```
UL {list-style-type: square;}
OL {list-style-type: lower-roman;}
```

- `margin` - Diese Eigenschaft bestimmt die Größe der Ränder um ein Element. Negative Werte sind erlaubt, aber der Inhalt des Elements kann dann überdeckt werden.

```
H1 {margin: 2ex;}
```

- `margin-bottom` - Diese Eigenschaft bestimmt die Größe des Rands unterhalb eines Elements. Negative Werte sind erlaubt, aber der Inhalt des Elements kann dann überdeckt werden.

```
UL {margin-bottom: 0.5in;}
```

- `margin-left` - Diese Eigenschaft bestimmt die Größe des Rands links von einem Element. Negative Werte sind erlaubt; achten Sie jedoch darauf, das Element nicht außerhalb des Fensters zu positionieren.

```
P {margin-left: 3em;}
```

- `margin-right` - Diese Eigenschaft bestimmt die Größe des Rands rechts von einem Element. Negative Werte sind erlaubt; achten Sie jedoch darauf, das Element nicht außerhalb des Fensters zu positionieren.

```
IMG {margin-right: 30px;}
```

- `margin-top` - Diese Eigenschaft bestimmt die Größe des Rands unterhalb eines Elements. Negative Werte sind erlaubt, aber der Inhalt des Elements kann dann überdeckt werden.

```
UL {margin-top: 0.5in;}
```

- `padding` - Diese Eigenschaft bestimmt die Größe des Füllbereichs an allen Seiten eines Elements. Der Füllbereich erbt den Hintergrund des Elements. Negative Werte sind nicht erlaubt.

```
H1 {padding: 2ex;}
```

- `padding-bottom` - Diese Eigenschaft setzt die Größe des Füllbereichs unterhalb eines Elements. Der Füllbereich erbt den Hintergrund des Elements. Negative Werte sind nicht erlaubt.

```
UL {padding-bottom: 0.5in;}
```

- `padding-left` - Diese Eigenschaft setzt die Größe des Füllbereichs links von einem Element. Der Füllbereich erbt den Hintergrund des Elements. Negative Werte sind nicht erlaubt.

```
P {padding-left: 3em;}
```

- `padding-right` - Diese Eigenschaft setzt die Größe des Füllbereichs rechts von einem Element. Der Füllbereich erbt den Hintergrund des Elements. Negative Werte sind nicht erlaubt.

```
IMG {padding-right: 30px;}
```

- `padding-top` - Diese Eigenschaft setzt die Größe des Füllbereichs oberhalb eines Elements. Der Füllbereich erbt den Hintergrund des Elements. Negative Werte sind nicht erlaubt.

```
UL {padding-top: 0.5in;}
```

- `text-align` - Diese Eigenschaft bestimmt die horizontale Ausrichtung des Texts in einem Element. Mögliche Werte sind `left`, `right`, `center` und `justify`. Die Eigenschaft kann nur auf Elemente auf Blockebene angewendet werden.

```
P {text-align: justify;}
H4 {text-align: center;}
```

- `text-decoration` - Diese Eigenschaft setzt Effekte für den Text, beispielsweise Unterstreichen oder Blinken. Mögliche Werte sind `none`, `underline`, `overline`, `line-through` und `blink`. Diese Werte können kombiniert werden.

```
STRONG {text-decoration: underline;}
.old {text-decoration: line-through;}
```

- `text-indent` - Diese Eigenschaft legt die Einrückung der ersten Zeile in einem Element fest. Die Eigenschaft kann nur auf Elemente auf Blockebene angewendet werden. Negative Werte sind erlaubt.

```
P {text-indent: 5em;}
H2 {text-indent: -25px;}
```

- `text-transform` - Diese Eigenschaft ändert die Zeichendarstellung der Buchstaben in einem Element, unabhängig von der ursprünglichen Eingabe. Mögliche Werte sind `none`, `capitalize`, `uppercase` und `lowercase`.

```
H1 {text-transform: lowercase;}
.title {text-transform: capitalize;}
```

- `vertical-align` - Diese Eigenschaft bestimmt die vertikale Ausrichtung der Grundlinie eines Elements abhängig von der Linienhöhe des übergeordneten Elements. Diese Eigenschaft kann nur auf Inline-Elemente (z.B. Tabellenzellen) angewendet werden. Negative Werte sind erlaubt. Mögliche Werte sind `baseline`, `sub`, `super`, `top`, `text-top`, `middle`, `bottom`, `text-bottom` oder ein Prozentwert.

```
SUP {vertical-align: super;}
.fnote {vertical-align: 50%;}
```

- `white-space` - Diese Eigenschaft definiert, wie Whitespaces (Leerzeichen, Tabulatoren usw.) in einem Element behandelt werden sollen. Mögliche Werte sind `normal` (Mehrfachleerzeichen zu einem einzigen Leerzeichen zusammenfassen), `pre` (Mehrfachleerzeichen werden nicht zusammengefaßt), `nowrap` (keinen Zeilenumbruch ohne
-Tag erlauben).

```
DL {white-space: nowrap;}
DD {white-space: pre;}
```

- `width` - Diese Eigenschaft bestimmt die Breite eines Elements. Diese Eigenschaft bezieht sich normalerweise nur auf Bilder, kann aber auf jeder Blockebene und für jedes ersetzte Element angewendet werden. Negative Werte sind nicht erlaubt. Bei der Breite kann es sich um eine Längenangabe, einen Prozentwert oder das Wort `auto` (Standard) handeln.

```
TABLE {width: 80%;}
```

- `word-spacing` - Diese Eigenschaft bestimmt den Whitespace zwischen den Wörtern (Zeichenketten, die durch Whitespaces eingerahmt werden). Mögliche Werte sind normal oder eine Längenangabe.

```
P {word-spacing: 0.5em;}
```

Zusammenfassung

In diesem Kapitel haben Sie erfahren, wie Sie Ihr XML abändern und ihm CSS-Stylesheets zuordnen können, so daß es in den aktuellen Web-Browsern angezeigt werden kann. Diese Änderungen sind zwar eher provisorisch – die Web-Browser aber auch. Als Vorbereitung auf das, was noch kommt, haben Sie anhand des Beispielcodes gesehen, wie Sie mit ein bißchen Ausschneiden&Einfügen ganz schnell ein XML-Stylesheet anlegen, mit dem Sie Ihre XML-Elemente in HTML-Elemente umwandeln und diesen CSS-Stileigenschaften zuordnen.

Es ist zu erwarten, daß es sehr bald robustere, konformere Versionen dieser Browser geben wird und daß sie XML und auch XSL besser unterstützen. Für die Zwischenzeit sollte Ihnen dieses Kapitel jedoch genügend Information verschafft haben, selbst ein bißchen damit zu experimentieren.

F&A

F Warum braucht man überhaupt noch CSS, wenn es doch DSSSL und XSL gibt?

A *DSSSL ist eher eine Verarbeitungssprache als eine Stilsprache. XSL ist noch nicht vollständig spezifiziert und wird auch längere Zeit noch nicht unterstützt werden. CSS ist der kleinste gemeinsame Nenner für alle Stilsprachen, gleichzeitig aber auch diejenige mit der besten Unterstützung. Einfacher gesagt, die Darstellung von XML mit Hilfe von CSS ist eine sinnvolle Alternative – statt zu warten.*

F Warum XSL und CSS? Eine Stilsprache sollte doch ausreichend sein.

A *CSS ist für die Verwendung zusammen mit HTML ausgelegt. XLS ist für die Verwendung mit XML ausgelegt. XML soll HTML nicht ersetzen (was aber der Fall sein könnte), und XSL soll CSS nicht ersetzen (was ebenso der Fall sein könnte). XML muß abwärtskompatibel sein, deshalb muß es die CSS-Eigenschaften unterstützen, aber CSS bietet immer noch eine schnellere und einfachere Methode, HTML-Elementen Stil zuzuordnen. Mögliche Microsoft-Erweiterungen für CSS (wie etwa* `CSS behaviors`*) führen jedoch vielleicht dazu, daß CSS irgendwann eine eigene Nische findet.*

F Was gewinnt man damit, XML-Elemente innerhalb desselben Dokuments mit HTML-Elementen zu kombinieren?

A *Web-Browser erkennen HTML-Tags und stellen den nachfolgenden Text entsprechend dar. Sie können einen Browser austricksen, wenn Sie seine Standardformatierung auf XML-Elemente anwenden, indem Sie die Namen von HTML-Elementen verwenden. Damit ersparen Sie sich, den Stil explizit spezifizieren zu müssen.*

Übungen

1. Wählen Sie ein einfaches XML-Dokument aus. Es sollte sehr einfach sein; man unterschätzt leicht den Zeitaufwand für das Experimentieren. Legen Sie ein Stylesheet dafür an, das es im IE5 anzeigt.
2. Legen Sie für dasselbe Dokument ein Stylesheet an, um es in Mozilla anzuzeigen.
3. Betrachten Sie Ihr XML-Dokument noch einmal, und überlegen Sie, wie Sie es in einem semantischen HTML (Verwendung von Attributen zur Kennzeichnung von Elementen) einfacher hätten codieren können. Beschränken Sie sich dabei auf die Anzeigeaspekte, die einfacher gewesen wären, der XML-Code oder der HTML-Code? Welche Auswirkung hätte Ihre Wahl auf die Portierbarkeit Ihres Codes?

XML-Konvertierung mit DSSSL

Woche 3

In Kapitel 18 habe ich die Entwicklungsgeschichte der XML-Stilsprachen von CSS (der einfachsten) über DSSSL (der kompliziertesten) bis DSSL-o, XS, XSL1 und XSL2 skizziert. Sie haben erfahren, wie CSS-Stylesheets zur Darstellung von XML-Code in einem Web-Browser eingesetzt werden – der reine XML-Code, in HTML-Code eingehüllt, oder eine hybride Mischung aus XML-Elementen und HTML-Tags. Heute werden wir unsere Reise durch die Stilsprachen fortsetzen. Unter anderem geht es dabei um folgendes:

- Die Grundlagen von DSSSL und ihre Unterstützung durch jade, die Public Domain DSSSL-Engine von James Clark.
- Einrichtung einer einfachen, kostenlosen Entwicklungsumgebung, um jade auszuführen und XML-Code zu verarbeiten.
- Die Entwicklung grundlegender DSSSL-Stylesheets zur Konvertierung von XML-Code in HTML, RTF und MIF (*Frame Maker Interchange Format*)

Wo DSSSL ins Spiel kommt

Man kann ohne Übertreibung sagen, DSSSL (sprich »dissl«) sei komplex. Um irgend etwas Sinnvolles damit zu erledigen, müssen Sie nicht nur die Programmiersprache LISP beherrschen, sondern auch einen ganz bestimmten LISP-Dialekt, Scheme. Und es gibt nur sehr wenige Programmierer auf der ganzen Welt, die Scheme beherrschen.

DSSSL ist nicht ganz einfach zu erlernen, weil es außer den Entwürfen des ISO-Standards keinerlei Dokumentation gibt.

Die Komplexität von DSSSL (bedingt auch durch die Tatsache, daß es zwei Sprachen in sich vereinigt, die Transformationssprache und die Stilsprache) führte sehr schnell zu einem Versuch, eine vereinfachte Version bereitzustellen, *DSSSL-Lite*. DSSSL-Lite wiederum führte zu einer Internet-Variante von DSSSL, DSSSL-o (kurz für DSSSL-online).

Die allererste Version der XML-Stilsprache (damals mit dem Namen XS) basierte weitgehend auf DSSSL-o, und die verfügbare DSSSL-o-Software wurde langsam an XS angepaßt, was die Migration ganz einfach machte.

XS wurde angepaßt – HTML-Ergänzungen machten es kompatibler zu existierenden CSS-Stilen, und zahlreiche DSSSL-o-Funktionen wurden gestrichen. Die Syntax war völlig anders, aber das konzeptuelle Modell blieb stets das von DSSSL übernommene. Das alles führte schließlich zur ersten Version von XSL.

Im August 1998 gab es eine überarbeitete und sehr unvollständige Neuversion der Sprache, veröffentlicht als Arbeitsskizze. Diese XSL-Version benutzt immer noch dasselbe Verarbeitungsmodell wie DSSSL, aber auch hier wird eine völlig andere Syntax verwendet. Es gibt bereits neue Softwarewerkzeuge (die Sie in Kapitel 20 kennenlernen werden), aber offensichtlich dauert es noch eine Zeit, bis sich XSL stabilisiert haben wird, und noch länger, bis es von Werkzeugen und Browsern unterstützt wird.

XSL unterliegt einer ständigen Weiterentwicklung. DSSSL ist nicht verschwunden, und das jade-Paket ist so gut wie schon immer. Falls überhaupt, dann herrscht ein Kräftegleichgewicht, weil der neueste Release (1.2) ein `mif`-Backend enthält, das SGML- und XML-Code (und damit laut Definition auch HTML-Code) in das Maker Interchange Format umwandeln kann. Das Maker Interchange Format kann von den Profi-DTP-Softwarepaketen Adobe FrameMaker und Frame+SGML direkt gelesen werden.

Eine DSSSL-Entwicklungsumgebung

Ich habe jade bereits erwähnt, die DSSSL-Engine von James Clark. Wenn Sie Microsoft Windows einsetzen, können Sie mit Hilfe der folgenden Softwarepakete eine eigene, fast kostenlose Entwicklungsumgebung einrichten (für andere Plattformen müssen Sie den jade-Quellcode möglicherweise kompilieren, der ebenfalls kostenlos erhältlich ist).

- jade – Dieses Paket können Sie von der Web-Site von James Clark unter `http://www.jclark.com/jade` herunterladen.
- PFE (*Programmer's File Editor*) – Dieses Paket kann von Alan Philips Site bei der Universität von Lancester heruntergeladen werden: `http://www.lancs.ac.uk/people/cpaap/pfe/pfefiles.htm`.

  PFE ist viel mehr als ein Texteditor (obwohl er natürlich auch auf diesem Gebiet beste Dienste leistet). Er bietet die praktischsten Möglichkeiten, wie beispielsweise die Aufzeichnung von Tastencodes, die Überprüfung korrekter Klammernpaarung, die Ausführung eines DOS-Befehls und die Ausgabe des Ergebnisses in ein Fenster – um nur ein paar zu nennen.

Als ich mit dem DSSSL-Code experimentiert habe (und nur so lernt man ihn kennen!), habe ich den Quellcode (XML, SGML oder sogar HTML) in ein Textfenster heruntergeladen, anschließend das DSSSL-Stylesheet in ein zweites Fenster und die jade-Fehlerausgabedatei in ein drittes Fenster. Anschließend öffnete ich ein Befehlszeilenfenster zur Eingabe der jade-Befehle (siehe Abbildung 19.1). Ich verwende 4NT, weil es viel leistungsfähiger als alle seine Konkurrenten ist (Sie finden 4NT über `http://www.jpsoft.com`), aber Sie können genausogut einfach die MS-DOS-Eingabeaufforde-

rung verwenden. Beachten Sie, daß dieser Screenshot aus technischen Gründen nur mit 800 x 600 Pixel angefertigt werden konnte. Meine normale Auflösung von 1152 x 864 Pixel zeigt viel mehr von dem Dateiinhalt an.

PFE erkennt, wenn sich die jade-Fehlerprotokolldatei ändert. Ich habe den Standardnamen `error.log` für diese Datei beibehalten, um spätere Aufräumarbeiten zu vereinfachen. Nach der Ausführung eines jade-Befehls bewirkt das Umschalten zu PFE die Aufforderung, die Fehlerprotokolldatei neu zu laden. Sie können die Fehler ganz einfach verfolgen, indem Sie in der PFE-Datei die Zeilennummern der fehlerhaften Datei anzeigen.

Natürlich ist dies nicht die einzige Umgebung. Wenn Sie mehr Zeit haben, können Sie auch mit EMACS im DSSSL-Modus arbeiten. Allerdings sollten Sie davon absehen, wenn Sie keine starken Nerven haben – obwohl er einige sehr leistungsfähige Funktionen aufweist.

Installation von jade

Es gibt nicht viel zu installieren. Im ersten Schritt entpacken Sie das ZIP-Archiv (vorausgesetzt, Sie laden eine ausführbare Datei herunter; falls Sie den Quellcode laden, lesen Sie die Build-Anweisungen in der jade-Dokumentation nach) in ein separates Verzeichnis und fügen Ihrem Pfad `jade.exe` hinzu.

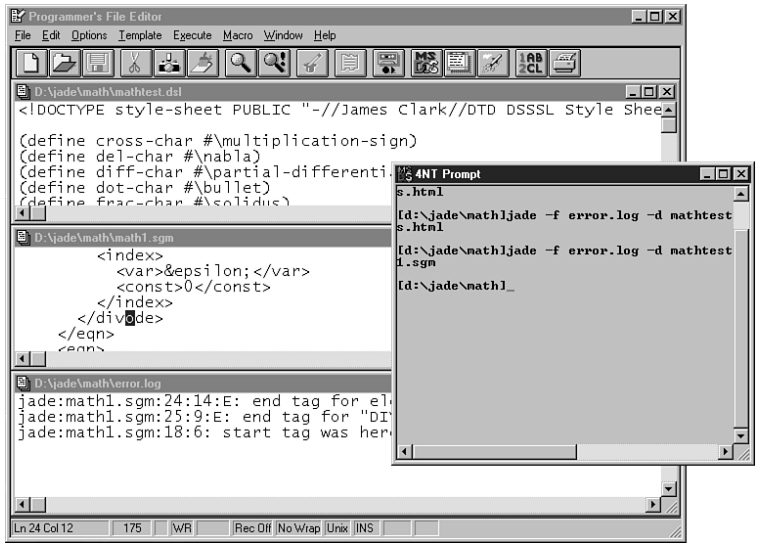


Abbildung 19.1: Eine DSSSL-Entwicklungsumgebung, in der Jade, PFE und 4NT verwendet werden.

Ausführung von jade

jade ist, wie Sie aus Abbildung 19.1 ersehen, ein befehlszeilenorientiertes Programm. Es hat keine Benutzeroberfläche mit Menüs und Dialogfeldern, in die alle Parameter eingegeben werden können.

Bevor Sie jade ausführen, sollten Sie die `catalog`-Datei (aus der jade-Distribution) kopieren und die Einträge entsprechend bearbeiten, um die Position der jade-Support-Dateien auf Ihrem Computer anzugeben.

Wenn eine der von Ihnen verarbeiteten SGML- oder XML-Dateien mittels eines öffentlichen Bezeichners auf eine DTD verweist, müssen Sie auch Einträge dafür anlegen. Ein Beispiel für eine auf diese Weise abgeänderte Katalogdatei finden Sie in Listing 19.1.

Listing 19.1: Beispiel für eine Katalogdatei in jade

```
1: PUBLIC "-//James Clark//DTD DSSSL Flow Object Tree//EN"
2:       "d:\jade\fot.dtd"
3: PUBLIC "ISO/IEC 10179:1996//DTD DSSSL Architecture//EN"
4:      "d:\jade\dsssl.dtd"
5: PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN"
6:       "d:\jade\style-sheet.dtd"
7: PUBLIC "-//IETF//DTD HTML Strict//EN" "d:\jade\html-s.dtd"
8: PUBLIC "-//W3C//DTD HTML 3.2//EN" "d:\jade\html32.dtd"
9: PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML"
10:      "d:\jade\ISOLat1.sgm"
11: PUBLIC "-//Free Text Project//DTD Play//EN"
12:       "d:\jade\work\play.dtd"
```

Die ersten drei Zeilen der Katalogdatei sind wesentlich; ohne diese Zeilen gibt jade nichts außer einer langen Fehlerliste aus. Die anderen Zeilen sind optional – eine für jede `PUBLIC`-DTD-Deklaration, die Sie verwenden (damit ersparen Sie sich den Ärger, jede Datei zu bearbeiten und die DTD in einen `SYSTEM`-Bezeichner abzuändern).

Die einfachste Methode, richtige Einträge in der Katalogdatei sicherzustellen, ist, die betreffende XML-Datei zu öffnen, den Teil der DTD, der mit dem Schlüsselwort `PUBLIC` beginnt, zu kopieren und ihn dann direkt in die Katalogdatei zu kopieren. Anschließend können Sie den Pfad zu der Datei mit der DTD einfügen.

Hier einige typische Befehle, die Sie für die Ausführung von jade brauchen:

Der Befehl wandelt eine HTML-Datei in RTF um:

```
jade -f error.log -t rtf -d html32hc.dsl -o simon.rtf simon.html
```

Das DSSSL-Stylesheet `html32hc.dsl` wird zusammen mit jade bereitgestellt. Es ist gut für die allgemeine Konvertierung von HTML in RTF geeignet. Es fügt der RTF-Datei sogar ein Inhaltsverzeichnis hinzu (um die Seitennummerneinträge zu aktualisieren, drücken Sie Strg+Ende, Strg+A und F9).

Dieser Befehl wandelt eine XML-Datei in MIF (*FrameMaker Interchange Format*) um:

```
jade -f error.log -t mif -wxml -d simon.dsl -o simon.rtf simon.xml
```

Beachten Sie den zusätzlichen Parameter `-wxml`. jade ist auf SGML-Dateien ausgelegt und beschwert sich über die fehlenden Tag-Minimierungszeichen in der XML-DTD, falls Sie diesen Parameter nicht angeben. Im allgemeinen sind diese Fehler nicht weiter schlimm und können ignoriert werden, aber gehen Sie lieber sicher. Dieser Parameter wird nur dann angewendet, wenn es sich um ein gültiges XML-Dokument handelt (es gibt einen SGML-Parser in jade). Falls Ihr XML-Dokument nur wohlgeformt, aber nicht notwendigerweise gültig ist oder wenn Sie die Auswertung umgehen möchten, geben Sie statt dessen den Parameter `-wno-valid` an.

Dieser Befehl wandelt eine XML-Datei in HTML um:

```
jade -f error.log -t sgml -wno-valid -d simon.dsl
simon.xml > simon.html
```

Beachten Sie, daß Sie jetzt die Ausgabe über eine Pipe an die Ausgabedatei schicken müssen (mit dem Zeichen >).

Dieser Befehl wandelt eine HTML-Datei in XML um:

```
jade -f error.log -t xml -d simon.dsl simon.html > simon.xml
```

Nachdem Sie die Grundlagen kennen, wollen wir die offizielle Bedeutung dieser Befehlszeilenparameter betrachten (die auch in der jade-Dokumentation erklärt sind):

- `-c katalogdatei` – Eine Alternative zum Kopieren der Katalogdatei (wie bereits beschrieben) in das Verzeichnis, von dem aus Sie jade ausführen, ist die Angabe des Pfads zur Katalogdatei in diesem Parameter.
- `-d dsssl_spezifikation` – Dieser Parameter gibt den Pfad zur der verwendeten DSSSL-Stilspezifikation (Stylesheet) an.

Offiziell ist die Datei mit den DSSSL-Formatanweisungen ein Stylesheet, das ein oder mehrere DSSSL-Stilspezifikationen enthalten kann. In jade handelt es sich bei dem Stylesheet um ein SGML-Dokument, das konform mit der Stylesheet-DTD ist.

Um Sie nicht mit zu vielen Begriffen zu verwirren, werde ich im weiteren einfach den Begriff »DSSSL-Stylesheet« verwenden. Unsere Stylesheets enthalten nur eine Stilspezifikation, und weil die DTD erlaubt, daß die Element-Start- und -Ende-Tags in einem Stylesheet weggelassen werden, brauchen Sie sich auch keine Gedanken über den Unterschied zu machen.

- `-f protokoll_datei` – Dieser Parameter gibt den Namen der Datei an, in die die Fehlermeldungen geschrieben werden sollen.
- `-G` – Dieser Schalter aktiviert den Debug-Modus. Wenn bei der Auswertung eines Ausdrucks ein Fehler auftritt, zeigt jade einen Stack-Trace an. Das wird Ihnen aber erst dann weiterhelfen, wenn Sie bereits ein bißchen Erfahrung mit jade gesammelt haben. (Es gibt außerdem einige sehr leistungsfähige DSSSL-Debugging-Funktionen im Web, die Ihnen viel praktischere Informationen bereitstellen.)
- `-t ausgabetyp` – Dieser Parameter gibt den Typ der zu erzeugenden Ausgabe an, wobei `ausgabetyp` eines der folgenden Schlüsselwörter sein kann:
- `fot` – Eine XML-Darstellung des Flußobjektbaums.
- `Rtf oder rtf-95`: Microsoft Rich Text Format – `rtf-95` erzeugt Ausgaben, die für Word 95 statt für Word 97 optimiert sind.
- `tex` – TeX.
- `sgml` – SGML (für SGML-SGML-Transformationen).
- `xml` – XML (für SGML-in-XML-Transformationen).
- `-o ausgabedatei` – Dieser Parameter weist jade an, seine Ausgaben in die Datei `ausgabedatei` zu schreiben statt auf die Standardausgabe. Der Standarddateiname ist der Name der letzten Eingabedatei, wobei die Dateinamenerweiterung durch den Ausgabetyp ersetzt wird (`rtf`, `mif` usw.). Falls es keinen Eingabedateinamen gibt, wird die Erweiterung dem Namen `jade-out` hinzugefügt.
- `-V variable` – Dieser Parameter erlaubt Ihnen, den Wert der benannten Variablen auf `true` zu setzen. In dem DSSSL-Stylesheet zu Jon Bosaks Veröffentlichung von Shakespeares Schauspielen in XML-Form (kostenlos unter `http://www.hypermedic.com/style/shakespeare/index.htm` zu laden) können Sie beispielsweise `-V` angeben, um das `fm`-Element darzustellen; dieser Inhalt wird normalerweise übersprungen.

- `-w` – dieser Parameter erlaubt Ihnen, XML-Verarbeitungsoptionen anzugeben:
- `-wxml` schaltet jade in den XML-Verarbeitungsmodus um.
- `-wno-valid` teilt jade mit, daß das XML-Dokument nicht gültig sein muß (das Dokument muß jedoch immer wohlgeformt sein).

Fehlermeldungen in jade

jade tut sein bestes, die erkannten Fehler kundzutun. Listing 19.2 zeigt ein Beispiel für einige dieser Fehlermeldungen.

Listing 19.2: Fehlermeldungen in jade

```
1:  jade:test.xml:7:0:E: DTD did not contain element
➥declaration for document type name
2:  jade:test.xml:9:5:E: element "test" undefined
3:  jade:test.xml:10:10:E: element "a" undefined
4:  jade:test.xml:11:7:E: ID "a1" already defined
5:  jade:test.xml:10:7: ID "a1" first defined here
6:  jade:test.xml:11:10:E: element "b" undefined
7:  jade:test.xml:12:2:E: required attribute "id" not specified
8:  jade:test.xml:12:2:E: element "b" undefined
9:  jade:test.xml:12:28:E: end tag for "b" omitted, but OMITTAG NO
➥was specified
10: jade:test.xml:12:0: start tag was here
11: jade:test.xml:12:28:E: end tag for "b" omitted, but OMITTAG NO
➥was specified
12: jade:test.xml:11:0: start tag was here
13: jade:test.xml:12:28:E: end tag for "a" omitted, but OMITTAG NO
➥was specified
14: jade:test.xml:10:0: start tag was here
15: jade:test.xml:12:28:E: end tag for "test" omitted, but OMITTAG NO
➥was specified
16: jade:test.xml:9:0: start tag was here
```

Die Fehlermeldungen (siehe Listing 19.2) teilen Ihnen folgendes mit:

- Das Programm (in diesem Fall normalerweise jade), das den Fehler erkannt hat.
- Den Namen der Datei, die den Fehler enthält.
- Die Nummer der Zeile, die den Fehler enthält (das ist nicht immer richtig – manchmal kann jade nicht erkennen, wo ein Fehler angefangen hat – aber die Fehlerbeschreibung hilft Ihnen möglicherweise, ihn zu finden).
- Einen Fehlercode: E für einen Fehler (die Verarbeitung wird beendet), W für eine Warnung (die Verarbeitung wird fortgesetzt, aber möglicherweise ist die Ausgabe unbrauchbar), I für eine Information (Sie haben etwas falsch gemacht, aber dies ist nur zusätzliche Information; Warnungen werden häufig von Informationen begleitet), X für einen Ausführungsfehler (in jade selbst ist ein ernsthaftes Problem aufgetreten) und nichts (zeigt an, daß etwas Fehlerhaftes erkannt wurde, was aber nicht ernsthaft genug ist, um die Verarbeitung zu unterbrechen).
- Die Spaltennummer (Zeichennummer). Auch diese Angabe muß nicht immer richtig sein. Manchmal kann jade nicht erkennen, wo ein Fehler angefangen hat, aber die Fehlerbeschreibung hilft Ihnen möglicherweise, ihn zu finden.
- Eine Beschreibung des Fehlers

Anzeige von jade-Ausgabe

Offensichtlich brauchen Sie für die Umwandlung Ihres XML-Codes in ein anderes Format die entsprechende Software auf Ihrem Computer: rtf (Microsoft Word oder den kostenlosen Word-Viewer), mif (Adobe FrameMaker oder Frame+SGML), html (beliebige Web-Browser) oder tex (ein geeignetes TeX-Paket). Das könnte recht kostspielig sein und zwingt Sie, immer wieder die Ausgabedatei zu laden (Microsoft Word erlaubt Ihnen nicht einmal, eine neue RTF-Datei anzulegen, solange die alte geöffnet ist).

In den Fujitsu Labs wurde ein wunderbares Paket entwickelt, HyBrick, das all diese Probleme löst. Mit diesem Paket können Sie die Ausgaben auf Ihrem Bildschirm anzeigen (siehe Abbildung 19.2). Natürlich brauchen Sie weiterhin eine Entwicklungsumgebung. Sie finden HyBrick unter `http://collie.fujitsu.com/hybrick/`. Dabei handelt es sich um eine Beta-Version, die jedoch sehr stabil ist. Ich habe gehört, daß eine neue Version in Arbeit ist und daß es Pläne gibt, schon bald XSL zu unterstützen.

Die ersten Schritte – jade in der Praxis

Bevor wir weiter in die Theorie der DSSSL-Stile eintauchen und Sie davon überzeugen wollen, daß alles gar nicht so schwierig ist, wie auf den ersten Blick vermutet, wollen wir einige einfache Beispiele ausprobieren.

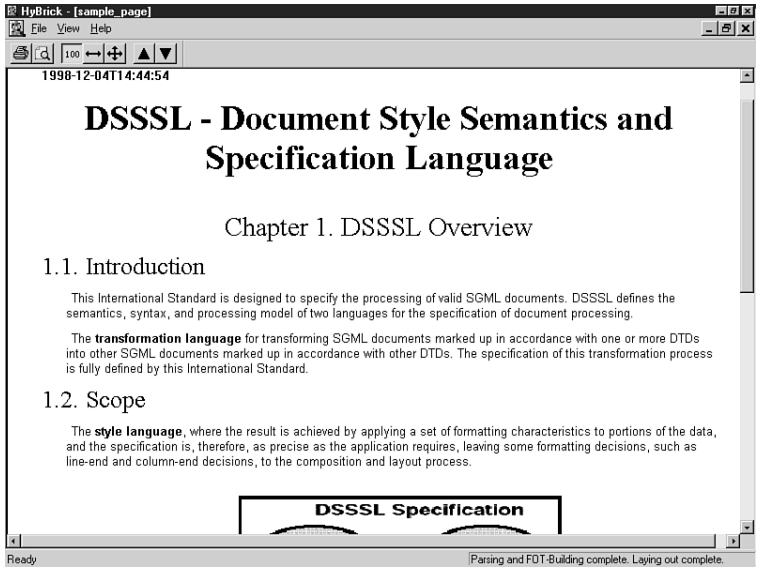


Abbildung 19.2: DSSSL-Rendering-Paket HyBrick von Fujitsu.

XML in RTF und MIF umwandeln

Zunächst wollen wir eine einfache XML-Datei in RTF umwandeln. Die XML-Eingabedatei sehen Sie in Listing 19.3.

Listing 19.3: Eine einfache XML-Datei für jade

```
1:  <?xml version="1.0"?>
2:  <!DOCTYPE test [
3:  <!ELEMENT test  (a ¦ b)*>
4:  <!ELEMENT a     (#PCDATA)>
5:  <!ELEMENT b     (#PCDATA)>
6:  ]>
7:  <test>
8:  <a>This is a test</a>
9:  <b>This is also a test</b>
10: <b>And this is another test</b>
11: </test>
```

Wie Sie sehen, ist dies wirklich eine sehr einfache XML-Datei, die eine interne DTD-Untermenge verwendet, um Probleme mit Dateinamen und Pfaden zu vermeiden. Jetzt probieren wir ein einfaches DSSSL-Stylesheet aus, das Sie in Listing 19.4 sehen.

Listing 19.4: Ein grundlegendes DSSSL-Stylesheet für die Umwandlung von XML in RTF

```
1:  <!DOCTYPE style-sheet PUBLIC
2:      "-//James Clark//DTD DSSSL Style Sheet//EN">
3:  (define *rgb-color-space*
4:    (color-space "ISO/IEC 10179:1996//Color-Space Family::Device RGB"))
5:
6:  (define *blue*
7:    (color *rgb-color-space* 0 0 1))
8:
9:  (define *red*
10:   (color *rgb-color-space* 1 0 0))
11:
12: (root                          ;set up the page geometry
13:     (make simple-page-sequence  ;as a simple page
14:    page-width:     8.5in   ;with characteristics
15:         page-height:    11in
16:         top-margin:     1in
17:         bottom-margin:  1in
18:         left-margin:    1in
19:         right-margin:   1in
20:         (process-children))) ;process the content of document
21:
22: (element test
23:     (process-children))
24:
25: (element a
26:     (make paragraph       ; make a paragraph
27:           font-size:      24pt
28:           color:          *blue*
29:           quadding:       'center
```

```
30:            space-before:   10pt
31:            line-spacing:   20pt))
32:
33: (element b
34:      (make paragraph   ; make another paragraph
35:            font-size:      18pt
36:            color:          *red*
37:            line-spacing:   30pt))
```

Dieses Stylesheet ist nicht besonders einfach, weil es Farben verwendet, und die Farbdefinitionen sind in jade etwas kompliziert. Glücklicherweise müssen Sie nicht verstehen, wie dieser Mechanismus funktioniert.

Wenn Sie einem bestimmten Element eine Farbe zuordnen wollen, deklarieren Sie den Farbraum und dann die gewünschten Farben – unter Verwendung von R- (Rot-), G- (Grün-) und B- (Blau-) Werten. Ich habe `*red*` und `*blue*` für die Namen der Farben verwendet, weil man sie durch den Stern einfacher erkennt, falls Sie das Stylesheet wechseln.

Abbildung 19.3 zeigt die Ausgabe-RTF-Datei in Microsoft Word.

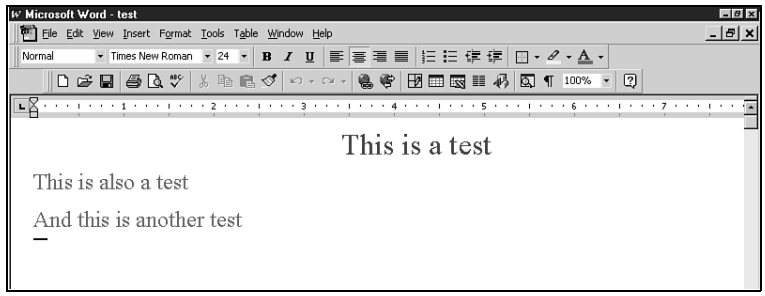


Abbildung 19.3: Die konvertierte XML-Datei in Microsoft Word.

Die Konvertierung in das MIF-Format ist ganz einfach. Man ändert den Parameter `-t rtf` in `-t mif` und lädt dann die Ausgabedatei in den FrameMaker, wie in Abbildung 19.4 gezeigt.

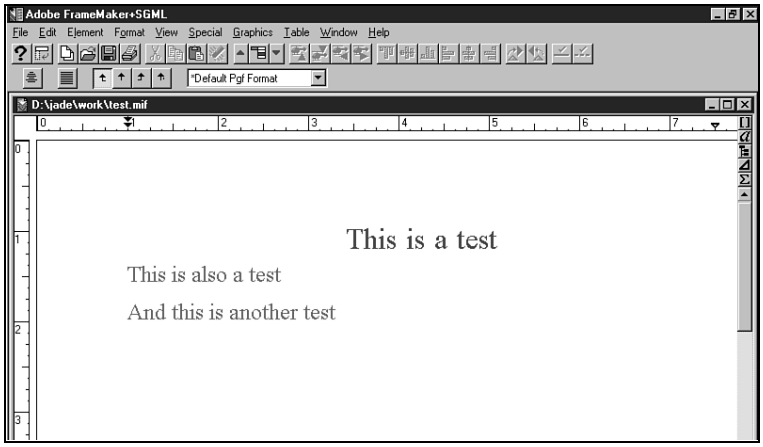


Abbildung 19.4: Die konvertierte XML-Datei im Adobe FrameMaker+SGML.

Konvertierung von XML in HTML

Die Konvertierung von XML in HTML-Code scheint zunächst recht kompliziert zu sein. Glücklicherweise müssen Sie diesen ganzen komplexen Code nicht verstehen – Sie brauchen ihn nur zu kopieren und unverändert in Ihr eigenes Stylesheet aufzunehmen. Listing 19.5 zeigt die XML-Datei, die wir konvertieren wollen.

Listing 19.5: XML-Datei für die Konvertierung in HTML

```
1:  <?xml version="1.0">
2:  <!DOCTYPE DOC [
3:  <!ELEMENT DOC  (TITLE ¦ NOTE ¦ PARA)*>
4:  <!ELEMENT TITLE (#PCDATA)>
5:  <!ELEMENT NOTE (#PCDATA)>
6:  <!ELEMENT PARA (#PCDATA)>
7:  ]>
8:  <DOC>
9:  <TITLE>Simple XML to HTML Conversion</TITLE>
10: <PARA>This sample documents demonstrates not only
```

```
11: how you can map XML elements onto HTML elements, but
12: also how you can set attributes as well.</PARA>
13: <NOTE>Note that this paragraph will be set in a
14: different color.</NOTE>
15: <PARA>This second paragraph proves that we have restricted
16: the color change to just the one note paragraph.</PARA>
17: </DOC>
```

Grundsätzlich müssen wir jedes der XML-Elemente dem entsprechenden HTML-Element zuordnen. Einige dieser Zuordnungen sind offensichtlich: `TITLE` zu `H1` und `PARA` zu `P`, aber das `NOTE`-Element gibt schon zu denken. Betrachten Sie das in Listing 19.6 gezeigte DSSSL-Stylesheet.

Listing 19.6: Ein DSSSL-Stylesheet für die Konvertierung von XML in HTML

```
1:  <!DOCTYPE style-sheet PUBLIC
2:     "-//James Clark//DTD DSSSL Style Sheet//EN"[
3:  <!ENTITY lt "&#38;#60;">
4:  <!ENTITY gt "&#62;">
5:  ]>
6:
7:  (declare-flow-object-class element
8:    "UNREGISTERED::James Clark//Flow Object Class::element")
9:  (declare-flow-object-class document-type
10:    "UNREGISTERED::James Clark//Flow Object Class::document-type")
11: (declare-flow-object-class empty-element
12:   "UNREGISTERED::James Clark//Flow Object Class::empty-element")
13: (declare-flow-object-class formatting-instruction
14: "UNREGISTERED::James Clark//Flow Object
15:             Class::formatting-instruction")
16:
17: (element DOC
18:   (sosofo-append
19:    (make document-type
20:      name:  "HTML"
21:      public-id: "-//W3C//DTD HTML 3.2//EN")
22:        (make element
```

```
23:          gi:  "HTML"
24:            (sosofo-append
25:              (make element
26:                gi:  "HEAD"
27:                  (make element
28:                    gi:  "TITLE"
29:                      (sosofo-append
30:                         (literal "Simple XML-to-HTML Conversion")
31:                      )))
32:              (make element
33:                gi:  "BODY"
34:                (process-children))))))
35:
36: (element TITLE
37:   (make element gi: "H1"))
38:
39: (element NOTE (make element gi: "P"
40:     (sosofo-append (literal "Note: "))
41:     (make element
42:       gi: "font"
43:       attributes: '(("COLOR" "RED")))))
44:
45: (element PARA  (make element gi: "P" ))
```

Sie können die vier Flußobjekt-Deklarationen ignorieren; sie sind keine Standard-DSSSL-Erweiterungen, die in jade aufgenommen wurden, um Dinge wie Konvertierungen von SGML und SGML oder von XML in HTML auszuführen. Sie können die Spezifikationen für das DOC-Element mehr oder weniger ignorieren – dieser Teil gibt einfach die DTD-Deklaration aus, die `HEAD`- und `TITLE`-Elemente, und hüllt das restliche Dokument in ein `HTML`- und ein `BODY`-Element ein. Die wichtigste Deklaration, die Sie sich hier merken sollten, ist (Web-Browser sind tolerant gegenüber schlechtem HTML, so daß Sie vermutlich akzeptable Ergebnisse erhalten, wenn Sie eine der Anweisungen für jedes der XML-Elemente einfügen und das restliche Stylesheet ignorieren):

```
(element xml.element (make element gi: "html.element"))
```

Dabei ist `xml.element` das XML-Element, das Sie konvertieren wollen, und `html.element` ist das HTML-Element, zu dem es werden soll. Diese Umwandlung kann so einfach sein! Listing 19.7 zeigt den HTML-Code nach der Konvertierung.

Listing 19.7: Die konvertierte XML-Datei

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
2:  <HTML
3:  ><HEAD
4:  ><TITLE
5:  >Simple XML-to-HTML Conversion</TITLE
6:  ></HEAD
7:  ><BODY
8:  ><H1
9:  >Simple XML to HTML Conversion</H1
10: ><P
11: >This sample documents demonstrates not only
12: how you can map XML elements onto HTML elements, but
13: also how you can set attributes as well.</P
14: ><P
15: >Note: <font
16: COLOR="RED"
17: >Note that this paragraph will be set in a different color.</font
18: ></P
19: ><P
20: >This second paragraph proves that we have restricted
21: the color change to just the one note paragraph.</P
22: ></BODY
23: ></HTML
24: >
```

Im Web-Browser sieht diese Datei wie in Abbildung 19.5 gezeigt aus.

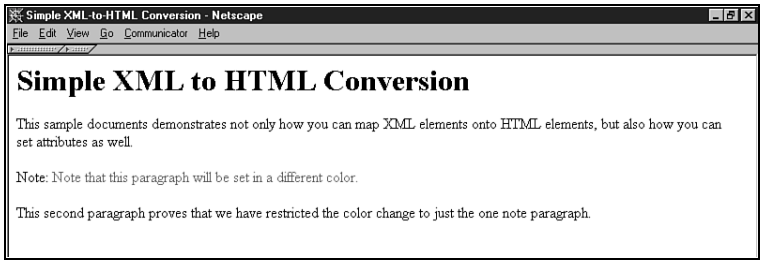


Abbildung 19.5: Die konvertierte XML-Datei im Netscape Communicator.

jade ist dafür bekannt, daß es die HTML-Tags über mehrere Zeilen verteilt, aber das hat überhaupt keine Auswirkung darauf, wie ein Web-Browser die Datei behandelt. Beachten Sie, wie das `NOTE`-Element aus der XML-Datei in ein `P`-Element in HTML umgewandelt wird, komplett mit Präfixtext und Farbänderung.

Grundlegendes DSSSL

Sie haben DSSSL bereits in der Praxis kennengelernt. Jetzt wollen wir die Theorie betrachten, die sehr schwer zu verstehen und noch schwieriger zu erklären sein kann. Aber auch wenn Sie das folgende nur teilweise verstehen, werden Sie genug wissen, um extrem leistungsfähige Stylesheets anzulegen. Wenn Ihnen das alles zu abstrakt erscheint, machen Sie sich ebenfalls keine Gedanken – am Ende dieses Kapitels gibt es praktische Beispiele, die Sie nachempfinden oder auch einfach nur kopieren können.

Flußobjekte

Die Grundbausteine der DSSSL-Stylesheets sind die sogenannten Flußobjekte. Flußobjekte sind darstellende Objekte, die schließlich das Ausgabedokument erzeugen. Diese Flußobjekte können beispielsweise Absätze, Seiten, Zeichenfolgen oder Elemente und Attribute sein.

Jedes Flußobjekt hat eine Anzeigeeigenschaft, die festlegt, ob es angezeigt wird (in einer neuen Zeile beginnt) oder inline ist (in den Container eines vorhergehenden Elements gehört). Abhängig von ihrer Aufgabe können Flußobjekte auch noch andere Eigenschaften aufweisen, beispielsweise Seitenränder, Schriftgrößen, Höhen und Breiten.

Flußobjekte sind international, so daß ein einziges Stylesheet genutzt werden kann, um Dokumente in natürlicher Sprache mit unterschiedlichen Schreibrichtungen anzulegen. Die Einrückungen werden deshalb immer mit Hilfe von `start`- und `end`-Werten spezifiziert, nicht durch `left`- oder `right`-Werte, weil ja nicht jeder von links nach rechts schreibt.

jade unterstützt nicht alle DSSSL-Flußobjektklassen, sondern nur die folgenden:

- `scroll` – Eine `scroll`-Flußobjektklasse wird als oberstes Flußobjekt für eine Online-Anzeige verwendet, die die Ausgabe nicht in Seiten unterteilt (wie ein HTML-Dokument). Sie nimmt angezeigte Flußobjekte auf. Die Anzeigeumgebung bestimmt die horizontale sowie die vertikale Größe des Flußobjekts.

- `paragraph` – Eine `paragraph`-Objektklasse repräsentiert einen Absatz. Ihr Inhalt kann inline oder angezeigt sein. Inline-Flußobjekte werden so formatiert, daß sie Zeilenbereiche bilden. Angezeigte Flußobjekte spezifizieren implizit einen Zeilenumbruch, und ihre Flächen werden zu der resultierenden Bereichsfolge addiert. Diese Flußobjektklasse ist grundlegend für die Formatierung von Textblöcken.
- `paragraph-break` – Zusammen mit der `paragraph-break`-Flußobjektklasse kann ein `paragraph`-Flußobjekt eine Abfolge von Absätzen darstellen. Die Absätze werden durch paragraph-break-Flußobjekte voneinander getrennt, welche atomar sind. `Paragraph-break`-Flußobjekte dürfen nur innerhalb von `paragraph`-Flußobjekten auftreten. Alle Eigenschaften eines `paragraph`-Flußobjekts gelten auch für ein `paragraph-break`-Flußobjekt. Die Eigenschaften eines `paragraph-break`-Flußobjekts bestimmen die Formatierung des Teils des Inhalts des `paragraph`-Flußobjekts, der diesem `paragraph-break`-Flußobjekt bis zum nächsten `paragraph-break` folgen, falls es ein solches gibt.
- `character` – Eine `character`-Flußobjektklasse wird so formatiert, daß sie einen einzigen Inline-Bereich erzeugt (character-Flußobjekte können nicht inline sein). Der Bereich kann mit benachbarten Inline-Bereichen vermischt werden. Sie verwenden ein character-Flußobjekt, um einer Zeichenkette ein anderes Layout oder andere Eigenschaften als dem Absatz, in dem sie sich befindet, zu geben.
- `line-field` – Die `line-field`-Flußobjektklasse ist inline und hat Inline-Inhalt. Sie erzeugt einen einzelnen Inline-Bereich. Die Breite dieses Bereichs ist gleich dem Wert der `field-width`-Eigenschaft. Falls der Inhalt des `line-field`-Bereichs nicht in diese Breite paßt, wird der Bereich entsprechend angepaßt. Tritt das `line-field` innerhalb des Absatzes auf, erfolgt ein Umbruch nach dem `line-field`. Sie können diese Objekte nutzen, um Präfixe mit fester Breite für bestimmte Listeneinträge anzulegen, so daß der Anfang des ersten Zeichens nach dem Präfix nicht von der Größe des Präfix abhängig ist. Diese Flußobjektklasse ist praktisch für Dinge wie Präfixtext vor unabhängigen Absätzen.
- `external-graphic` – Die Flußobjektklasse `external-graphic` wird für Grafiken in einem externen Entity verwendet (als Verweis importierte Grafiken). Flußobjekte dieser Klasse können inline oder angezeigt sein.
- `horizontal-rule` und `vertical-rule` – Diese Flußobjektklassen spezifizieren horizontale und vertikale Linien. Sie können inline oder angezeigt sein.
- `score` – Die Flußobjektklasse `score` wird zum Unterstreichen und Durchstreichen von Zeichen verwendet.
- `embedded-text` – Die Flußobjektklasse `embedded-text` wird für die Einbettung von von rechts nach links laufendem Text, in von links nach rechts laufendem Text oder umgekehrt verwendet. Dieses Flußobjekt kann nur inline verwendet werden.

- box – Die Flußobjektklasse box wird genutzt, um einen Rahmen um mehrere Flußobjekte zu ziehen. Dieses Flußobjekt kann angezeigt oder inline sein. Wird der Rahmen angezeigt, nimmt er alle angezeigten Flußobjekte auf. Ist er inline, nimmt er alle Inline-Flußobjekte auf. Ein box-Flußobjekt kann mehrere Bereiche erzeugen, dann wird die Kante des Rahmens neben dem Umbruch weggelassen. Ist der Rahmen inline, ist dieser Rahmen senkrecht zum Schreibmodus, ist er angezeigt, ist der Rahmen parallel zum Schreibmodus.
- table – Eine table-Flußobjektklasse enthält entweder alle table-part- oder alle table-column-, table-row- oder table-cell-Flußobjekte. Falls sie table-column-Flußobjekte enthält, müssen diese vor den anderen Flußobjekten erscheinen. Ein table-Flußobjekt kann nur angezeigt werden. Eine Tabelle besitzt zwei Richtungen: Zeilen und Spalten.
- table-part – Eine table-part-Flußobjektklasse ist nur innerhalb eines table-Flußobjekts erlaubt. Ein table-part-Flußobjekt besteht aus einem Tabellenrumpf, einem Tabellenkopf und einem Tabellenfuß. Im Rumpf dürfen nur table-column-Flußobjekte auftreten. table-row und table-cell dürfen an beliebiger Stelle in einer table-part erscheinen.

Durch das Formatieren eines table-part-Flußobjekts entsteht eine Folge von Flächen. Jede Fläche besteht aus einem Kopfinhalt (es sei denn, er wurde explizit weggelassen), gefolgt von einem Teil des Inhalts aus dem Tabellenrumpf, gefolgt von dem Inhalt des Tabellenfußes (es sei denn, er wurde explizit weggelassen). Jede Zeile im Rumpf kommt genau einmal vor, und die Reihenfolge der Zeilen wird beibehalten. Die Zeilen im Tabellenkopf und Tabellenfuß werden für jeden Ergebnisbereich wiederholt (so daß der Kopf wiederholt wird, wenn eine Tabelle über mehrere Seiten geht).

- table-column – Eine table-column-Flußobjektklasse ist ein atomares Flußobjekt, das Eigenschaften angibt, die für die Tabellenzellen derselben Spalte gelten.
- table-row – Eine table-row-Flußobjektklasse gruppiert Tabellenzellen in Zeilen; alle Tabellenzellen einer table-row beginnen in derselben geometrischen Zeile. Eine table-row nimmt table-cell-Flußobjekte auf. Ein table-row-Flußobjekt kann nur als Kind eines table-part- oder table-flow-Flußobjekts auftreten.
- table-cell – Eine table-cell-Flußobjektklasse nimmt alle Flußobjekte auf, die angezeigt werden können. Ein table-cell-Flußobjekt darf nur als Kind eines table-row-, table-part- oder table-flow-Flußobjekts auftreten. Die Breite der Tabelle ist gleich der Summe der Breiten aller Zellen.
- table-border – Eine table-border-Flußobjektklasse bestimmt den Rahmen einer Tabellenzelle oder einer ganzen Tabelle. Ein table-border-Flußobjekt darf jedoch nicht im Inhalt eines Flußobjekts auftreten. Die Breite der Rahmen hat keinen Einfluß auf die Breite der Zellen, die Positionierung des Zellinhalts, die Tabellenbreite oder die Größe des von der Tabelle eingenommenen Bereichs.

- `sequence` – Eine `sequence`-Flußobjektklasse verknüpft die Bereiche, die seine Kinder erzeugen, die entweder inline oder angezeigt sein können. Ein `sequence`-Flußobjekt ist praktisch, um geerbte Eigenschaften anzugeben. Ein anderes Flußobjekt nimmt nur dann ein `sequence`-Flußobjekt auf, wenn es auch alle in dieser `sequence` enthaltenen Flußobjekte aufnehmen würde.
- `display-group` – Eine `display-group`-Flußobjektklasse verknüpft die anderen Flußobjekte auf dieselbe Weise wie ein sequence-Flußobjekt, erzeugt aber auch einen neuen Anzeigebereich in einer neuen Zeile und wird von einer neuen Zeile abgeschlossen (auch wenn sie keinen Inhalt besitzt). Diese Flußobjekte werden für die Steuerung der Positionierung von Gruppen angezeigter Flußobjekte benutzt.
- `simple-page-sequence` – Eine `simple-page-sequence`-Flußobjektklasse wird formatiert, um eine Folge von Seitenbereichen zu erzeugen. Dieses Flußobjekt nimmt beliebige angezeigte Flußobjekte auf, darf jedoch nicht innerhalb des Inhalts eines anderen Flußobjekts auftreten. Dieses Flußobjekt wird für das Wurzelelement eines Dokuments verwendet oder für ein anderes Element, das oben auf einer neuen Seite beginnen soll.

Ein `simple-page-sequence`-Flußobjekt kann nur einen Zeilenkopf und einen Zeilenfuß besitzen, mit konstantem Text (bis auf die Seitennummer). Ein Dokument kann mehrere `simple-page-sequences` enthalten. Beispielsweise könnte jedes Kapitel eines Dokuments eine einzelne `simple-page-sequence` sein, so daß der Kapiteltitel in einer Kopf- oder Fußzeile angezeigt werden kann.

Die Seite wird von oben nach unten gefüllt. Die Anzeigegröße für den Inhalt der `simple-page-sequence` ist die Seitenbreite minus dem linken und dem rechten Rand.

- `link` – Eine `link`-Flußobjektklasse repräsentiert einen Hypertext-Link, der interaktiv durchlaufen werden kann, in der Regel durch Anklicken der Bereiche, die das Flußobjekt und seinen Inhalt darstellen. Wie Sie jedoch wissen, ist es in XML nicht unbedingt nötig, einen Link anzuklicken, um ihm zu folgen. Ein Link kann Inline-Flußobjekte und auch angezeigte Flußobjekte enthalten. `Link`-Flußobjekte können verschachtelt werden. Bei einer Verschachtelung ist der innerste Link der effektive Link.

Eigenschaften von Flußobjekten

Ein Flußobjekt ist eigentlich ein Formatobjekt – eine typographische Komponente, wie beispielsweise ein Absatz, eine Tabelle, ein Bereich, eine mathematische Gleichung usw. Flußobjekte haben spezielle Eigenschaften, abhängig von ihrer Aufgabe (es ist nicht sehr sinnvoll, eine Seitengröße einer Tabelle anzugeben!). Weitere Informationen über die Eigenschaften finden Sie in der DSSSL-o-Spezifikation oder in der DSSSL-Spezifikation (beide stehen kostenlos im Web zur Verfügung). Die meisten die-

ser Eigenschaften sind jedoch offensichtlich (mit Ausnahme von *quadding*, was die Ausrichtung betrifft). Hier folgt ein Überblick über die wichtigsten Flußobjektklassen und ihre Eigenschaften:

- `Display` – space-before, space-after, keep-with-previous?, keep-with-next?
- `simple-page-sequence` – page-width, page-height, left-margin, right-margin, top-margin, bottom-margin, header-margin, footer-margin, left-header, center-header, right-header, left-footer, center-footer, right-footer
- `paragraph` – min-leading, line-spacing, first-line-start-indent, last-line-end-indent, font-family-name, font-name, font-size, font-weight, start-indent, end-indent, space-before, space-after, quadding
- `line` – `field`-field-width, field-align
- `sideline` – sideline-side, sideline-sep, color, line-cap, line-thickness
- `character` – font-family-name, font-name, font-weight, font-size
- `leader` – length
- `rule` – orientation, length, color, line-thickness, start-indent, end-indent, space-before, space-after
- `external-graphic` – scale, max-width, max-height, entity-system-id, notation-system-id, color, start-indent, end-indent, space-before, space-after, position-point-x, position-point-y
- `score` – type, color, line-thickness
- `box` – box-type, background-color, box-size-before, box-size-after, color, length, line-thickness, start-indent, end-indent, space-before, space-after
- `table` – table-width, table-border, before-row-border, after-row-border, before-column-border, after-column-border, start-indent, end-indent, space-before, space-after
- `table-part` – start-indent, end-indent, space-before, space-after
- `table-column` – column-number, `n`-columns-spanned, width, start-indent, end-indent
- `table-cell` – column-number, n-columns-spanned, n-rows-spanned, cell-before-row-margin, cell-after-row-margin, cell-before-column-margin, cell-after-column-margin, cell-row-alignment, cell-background?, background-color, cell-before-row-border, cell-after-row-border, cell-before-column-border, cell-after-column-border, starts-row?, ends-row?, line-thickness
- `table-border` – border-present?, color, line-thickness

- `scroll` – background-color, background-tile, start-margin, end-margin
- `link` – destination

Neben den »offiziellen« Flußobjektklassen führt jade auch ein paar eigene ein. Sie gehören zwar nicht zum Standard, sind aber äußerst praktisch:

- `heading-level` – Ein Wert zwischen `1` und `9`, der den Absatz als Überschrift der angegebenen Ebene darstellt. Diese Eigenschaft erlaubt Microsoft Word, sinnvolle Outline-Ansichten sowie eine Gliederung anzuzeigen.
- `page-number-format` – Bestimmt das Format der von `page-number-sosofo` und `current-page-number-sosofo` verwendeten Nummer für Verweise auf Seiten in der `simple-page-sequence`. Der Ausgangswert ist `1`. Sie wird auf `simple-page-sequence`-Flußobjekte angewendet.
- `page-number-restart?` – Ist dies `true`, beginnt das Zählen der von `page-number-sosofo` und `current-page-number-sosofo` verwendeten Seitennummern für diese `simple-page-sequence` mit `1` neu. Der Ausgangswert ist #f. Sie wird auf `simple-page-sequence`-Flußobjekte angewendet.
- `page-n-columns` – Eine stets positive ganze Zahl, die die Spaltenanzahl angibt. Der Ausgangswert ist `1`. Sie wird auf `simple-page-sequence`-Flußobjekte angewendet.
- `page-column-sep` – Die Spaltenabtrennung. Der Ausgangswert ist `.5in`. Sie wird auf `simple-page-sequence`-Flußobjekte angewendet.
- `page-balance-columns?` – Ist dieser Wert `true`, sollen die Spalten auf der letzten Seite der Seitenfolge ausgeglichen werden. Der Ausgangswert ist `#f`. Sie wird auf `simple-page-sequence`-Flußobjekte angewendet.
- `superscript-height` – Die Höhe der Grundlinie einer Hochstellung oberhalb der Grundlinie des übergeordneten Elements. Sie wird auf `superscript`- und `script`-Flußobjekte angewendet.
- `subscript-depth` – Die Tiefe der Grundlinie einer Tiefstellung unterhalb der Grundlinie des übergeordneten Elements. Sie wird auf `subscript`- und `script`-Flußobjekte angewendet.
- `over-mark-height` – Die Höhe der Grundlinie des Inhalts des `over-mark`-Bereichs eines `mark`-Flußobjekts oberhalb der Grundlinie des Inhalts im Hauptbereich. Sie steuert außerdem die Höhe des Inhalts des mittleren hochgestellten Bereichs des `script`-Flußobjekts. Sie bezieht sich auf `mark`- und `script`-Flußobjekte.
- `under-mark-depth` – Die Tiefe der Grundlinie des Inhalts des `under-mark`-Bereichs eines `mark`-Flußobjekts unterhalb der Grundlinie des Inhalts im Hauptbereich. Sie steuert außerdem die Tiefe des Inhalts des mittleren tiefgestellten Bereichs des `script`-Flußobjekts. Sie bezieht sich auf `mark`- und `script`-Flußobjekte.

- `grid-row-sep` – Abtrennung zwischen den Zeilen eines `grid`-Flußobjekts.
- `grid-column-sep` – Abtrennung zwischen den Spalten eines `grid`-Flußobjekts.

Wenn Sie eine dieser Eigenschaften nutzen wollen, brauchen Sie eine Deklaration wie die folgende in Ihrem DSSSL-Stylesheet dafür:

```
(declare-characteristic page-n-columns
"UNREGISTERED::James Clark//Procedure::page-n-columns")
```

Flußobjektbaum

Das DSSSL-Stylesheet besteht aus mehreren sogenannten Flußobjektspezifikationen. (Technisch ausgedrückt, eine *sosofo* – eine *Specification of a Sequence of Flow Object*, also die Spezifikation einer Folge von Flußobjekten). Die folgende Regel erzeugt kein Flußobjekt; sie gibt nur an, wie dieses mit einer `make`-Anweisung angelegt werden kann:

```
(make paragraph font-size: 12pt )
```

Einfach ausgedrückt, jade nimmt die sosofos und erzeugt mir ihrer Hilfe Flußobjekte. Stellen Sie sich die sosofo einfach als die Schablone vor und die Flußobjekte als das Ergebnis der Anwendung dieser Schablone auf das Quelldokument.

Aus der Kombination aller sosofos erzeugt jade den sogenannten *Flußobjektbaum* (*Flow Object Tree*, fot).

Wenn jade (oder eine andere DSSSL-Engine) ein Dokument verarbeitet, legt es letztlich eine abstrakte Darstellung der Kombination aus Formatspezifikation und Quelldokument an. Diese Darstellung wird als *Flußobjektbaum* (*Flow Object Tree*) bezeichnet und seine Knoten als *Flußobjekte*.

Abbildung 19.6 zeigt einen typischen Flußobjektbaum in jade.

Die formatierte Ausgabe, also die Ergebnisse, die ausgedruckt oder angezeigt werden, wird durch Formatierung dieses Flußobjektbaums erzeugt.

Wenn Sie das Konzept des Flußobjektbaums verstanden haben, wissen Sie genug, um die Arbeitsweise von DSSSL grundsätzlich nachvollziehen zu können. Stellen Sie sich die Wurzel des Dokumentbaums beginnend am Wurzelelement vor. Sie können ein Element im Baum auswählen und es verarbeiten, indem Sie eine bestimmte Formatspezifikation darauf anwenden. Später im Stylesheet können Sie dieses Element erneut auswählen und eine andere Formatspezifikation darauf anwenden. Diese Bäume werden dann zum Flußausgabebaum kombiniert, der beide Variationen enthalten kann.

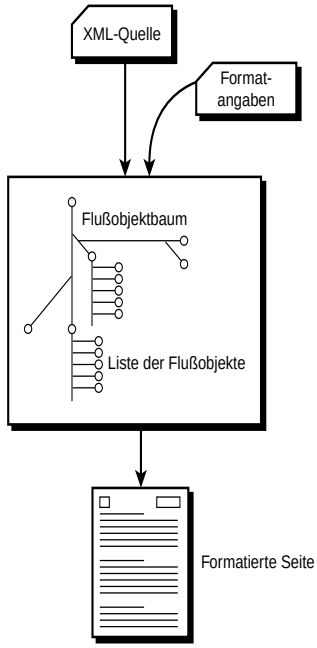


Abbildung 19.6:
Verarbeitung eines Dokuments.

Ist diese Erklärung zu abstrakt? Ich will Ihnen ein praktisches Beispiel zeigen.

Was ist ein Inhaltsverzeichnis? Sie verarbeiten das Dokument beginnend mit dem Wurzelelement und wählen alle Überschriften aus. Sie geben einen Absatz aus, der beispielsweise »Inhaltsverzeichnis« enthält, und dann alle Überschriften, die auf bestimmte Weise formatiert werden, vielleicht unter Angabe der Seitennummer (Sie könnten diese Seitennummer sogar zu einem aktiven Link machen). Wenn Sie nur ein Inhaltsverzeichnis extrahieren wollen, sind Sie bereits fertig. Das ist jedoch nicht von größerem Nutzen, wenn Sie den eigentlichen Inhalt des Dokuments formatieren wollen. Nachdem Sie das gesamte Dokument verarbeitet haben, um das Inhaltsverzeichnis anzulegen, müssen Sie jetzt zurück zum Wurzelelement (oder auch zur ersten Überschrift der zweiten Ebene, um den Titel zu überspringen) und das gesamte Dokument verarbeiten, um die eigentliche Ausgabe zu erzeugen.

Beim Anlegen des Flußausgabebaums können Sie das Quellobjekt, so oft Sie wollen, verarbeiten, auf beliebige Weise, so daß aus einem ersten XML-Dokument Inhaltsverzeichnisse, Abbildungsverzeichnisse, Indizes, Glossare, Verweislisten usw. erzeugt werden können. Und genau das macht die Leistungsfähigkeit und das Potential von DSSSL aus – es ist viel mehr als eine einfache Stilsprache. DSSSL ist eine Verarbeitungssprache, eine Transformationssprache und eine Stilsprache in einem. Gleichzeitig ist es eine vollständige Programmiersprache. Sie können Prozeduren, Funktionen, Makros, bedingte Verarbeitung, Prozeßflußsteuerung oder Ausdrücke verwenden – al-

les, was Sie von einer richtigen Programmiersprache erwarten. Das geht jedoch weit über den Rahmen dieses Buchs hinaus; deshalb wollen wir jetzt zur Praxis und der Arbeitsweise der Stylesheets zurückkehren.

Elementauswahl

Jede Formatspezifikation besteht aus einer Auswahlregel gefolgt von einer Konstruktionsregel.

Eine Auswahlregel besteht aus dem Schlüsselwort `element` und entweder dem Namen des Elements, auf das die Konstruktionsregel angewandt werden soll, oder einer Klausel, in der das richtige Element ausgewählt wird. Die Auswahlsyntax ist ganz einfach:

- `element PARA` wählt alle `PARA`-Elemente aus.
- `element (SECTION TITLE)` wählt nur `TITLE`-Elemente aus, die innerhalb von `SECTION`-Elementen angelegt sind.

Konstruktionsregeln

Eine Konstruktionsregel besteht aus einem `make`-Ausdruck gefolgt vom Namen einer Flußobjektklasse, einer Liste der Eigenschaften und ihrer Werte für diese Flußobjektinstanz sowie (optionalen) Verarbeitungsanweisungen, die bestimmte Kindelemente für die Verarbeitung auswählen.

Ein Beispiel:

```
(root  (make simple-page-sequence      (process-children))
```

Damit wird das gesamte Dokument ohne bestimmte Formatierung ausgegeben, während es sich bei folgendem um etwas ganz anderes handelt:

```
(make paragraph    space-before:     (if (or (equal? (gi (parent)) "A")
(equal? (gi (parent)) "B"))                              0pt
10pt)        (process-children-trim))
```

Die DSSSL-Syntax ist etwa gewöhnungsbedürftig, deshalb wollen wir diesen Ausdruck genauer betrachten:

- Wir haben das Schlüsselwort `if`, gefolgt von einer Auswertung:

  ```
  (or (equal? (gi (parent)) "A") (equal? (gi (parent)) "B"))
  ```

Ergibt der Ausdruck `true`, ist der Wert von `space-before` gleich `0pt`. Ist der Ausdruck `false`, ist der Wert von `space-before` gleich `10pt`.

- Es handelt sich dabei um einen `oder`-Ausdruck mit zwei Optionen:

  ```
  (equal? (gi (parent)) "A")
  ```

  und

  ```
  (equal? (gi (parent)) »B«)
  ```

- Der erste Ausdruck ist

  ```
  (equal? (gi (parent)) "A")
  ```

In natürlicher Sprache: »Ist das Elternelement dieses Elements ein A-Element?«

- Der zweite Ausdruck ist

  ```
  (equal? (gi (parent)) »B«)
  ```

In natürlicher Sprache: »Ist das Elternelement dieses Elements ein B-Element?«

Ist das Elternelement dieses Elements also entweder ein A-Element oder ein B-Element, ist der Wert von `space-before` gleich `0pt`. Andernfalls ist der Wert von `space-before` gleich `10pt`.

Neben numerischen Ausdrücken können Sie auch Zeichen, Zeichenketten, logische Ausdrücke und sogar Prozeduren verwenden. Ausdrücke in DSSSL können leistungsfähige Werkzeuge für die Auswahl von Elementen sein, und Sie können sie unterstützen, indem Sie das erste Element, das letze Element oder ein anderes Element an einer bestimmten Position im Baum oder Attribute oder Attributwerte auswählen. Ich werde hier nicht ins Detail gehen. Eine ausgezeichnete Erklärung finden Sie im Web unter der Adresse `http://www.mulberrytech.com/dsssl/dsssldoc`. Außerdem empfehle ich Ihnen, Beispiele zu betrachten und zu versuchen, sie zu verstehen.

Einfache Rezepte

Ich glaube, am besten lernt man durch Ausprobieren. In diesem Sinne werde ich Ihnen im restlichen Kapitel einige grundlegende Beispiele für häufige Anwendungen von DSSSL-Stylesheets vorstellen.

Präfixe für Elemente anlegen

In Listing 19.5 haben Sie bereits gesehen, daß Sie `"(sosofo-append (literal ""))"` verwenden können, um einen Textstring vor dem Inhalt eines Elements zu plazieren. Wäre es nicht praktisch, wenn man das Präfix eines Elements abhängig von dessen Namen bereitstellen könnte? Warnungen könnte der Text »Warning« vorangestellt werden usw. Betrachten Sie den XML-Code in Listing 19.8.

Listing 19.8: XML-Rezept-Datei 1 – Elementen ihren Namen als Präfix voranstellen

```
1:  <?xml version="1.0">
2:  <!DOCTYPE DOC [
3:  <!ELEMENT DOC     (TITLE ¦ CAUTION ¦ NOTE ¦ PARA)*>
4:  <!ELEMENT TITLE   (#PCDATA)>
5:  <!ELEMENT NOTE    (#PCDATA)>
6:  <!ELEMENT PARA    (#PCDATA)>
7:  <!ELEMENT CAUTION (#PCDATA)>
8:  ]>
9:  <DOC>
10: <TITLE>Simple XML to HTML Conversion</TITLE>
11: <PARA>This sample document demonstrates how you can
12: convert the tag name into a piece of text inside an
13: element.</PARA>
14: <NOTE>This is a note.</NOTE>
15: <PARA>This second paragraph proves that normal text is still
16: untouched.</PARA>
17: <CAUTION>This is a caution.</CAUTION>
18: </DOC>
```

Listing 19.9 ist das DSSSL-Stylesheet für Listing 19.7.

Listing 19.9: DSSSL-Rezept-Datei 1, Text als Präfix für ein Element

```
1:  <!DOCTYPE style-sheet PUBLIC
2:       "-//James Clark//DTD DSSSL Style Sheet//EN"[
3:  <!ENTITY lt "&#38;#60;">
4:  <!ENTITY gt "&#62;">
5:  ]>
6:  (declare-flow-object-class element
7:    "UNREGISTERED::James Clark//Flow Object Class::element")
8:  (declare-flow-object-class document-type
9:    "UNREGISTERED::James Clark//Flow Object Class::document-type")
10:  (declare-flow-object-class empty-element
```

```
11:   "UNREGISTERED::James Clark//Flow Object Class::empty-element")
12: (declare-flow-object-class formatting-instruction
13:   "UNREGISTERED::James Clark//Flow Object
14:     Class::formatting-instruction")
15:
16: (define debug
17: (external-procedure "UNREGISTERED::James Clark//Procedure::debug"))
18:
19: (define (make-special-para)
20:   (make sequence
21:     (make element
22:       gi: "P"
23:     (make element
24:       gi: "B"
25:       (literal (string-append (gi) ":"))))
26:     (make element
27:       gi: "BLOCKQUOTE"
28:       (process-children))))
29:
30: (element DOC
31:   (sosofo-append
32:     (make document-type
33:       name:  "HTML"
34:       public-id: "-//W3C//DTD HTML 3.2//EN")
35:         (make element
36:           gi:  "HTML"
37:             (sosofo-append
38:               (make element
39:                 gi:  "HEAD"
40:                   (make element
41:                     gi:  "TITLE"
42:                       (sosofo-append
43:                          (literal "Simple XML-to-HTML Conversion"))))
44:               (make element gi:  "BODY"
45:                (process-children))))))
46:
47: (element TITLE (make element gi: "H1" ))
48:
49: (element NOTE (make-special-para))
50: (element CAUTION (make-special-para))
51:
52: (element PARA (make element gi: "P" ))
```

Ignorieren Sie die Elemente `DOC`, `TITLE` und `P`. Das `DOC`-Element ist komplex, aber wenn Sie es schrittweise verfolgen, ganz einfach; es erzeugt die Ausgabe des korrekten HTML-Codes, wenn es auf ein `DOC`-Element trifft. Die Elemente `TITLE` und `P` sind nur einfache Zuordnungen.

Die Elemente `NOTE` und `CAUTION` sind interessanter. Weil ich sie als `special` behandeln will, habe ich die Prozedur `make-special-para` definiert. Nach dem Entfernen der Einrückungen (die mir hilft, alle öffnenden und schließenden Klammern zu verwalten), sieht diese Prozedur wie folgt aus:

```
(define (make-special-para)
  (make sequence (make element gi: "P" (make element gi: "B"
      (literal (string-append (gi) ":"))))
    (make element gi: "BLOCKQUOTE" (process-children))))
```

Ich lege eine Sequenz an (`make sequence`). Diese Sequenz besteht aus einem Element (`P`), deshalb wird alles folgende in ein `P`-Element eingehüllt. Es folgt ein Literal aus dem Elementnamen (`gi`) mit einem Doppelpunkt (:) (daher `string-append`). Außerdem folgt eine `make-element`-Anweisung, jetzt für ein `BLOCKQUOTE`-Element, und dann der Inhalt des Elements (`process-children`).

Betrachten wir den HTML-Code, der daraus entsteht – Listing 19.10.

Listing 19.10: HTML-Rezept-Datei 1, Ausgabe mit Präfixen für die Elemente

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
2:  <HTML>
3:  <HEAD>
4:  <TITLE>Simple XML-to-HTML Conversion</TITLE>
5:  </HEAD>
6:  <BODY>
7:  <H1>Simple XML to HTML Conversion</H1>
8:  <P>This sample document demonstrates how you can
9:  convert the tag name into a piece of text inside an
10: element.</P>
11: <P><B>NOTE:</B></P>
12: <BLOCKQUOTE>This is a note.</BLOCKQUOTE>
13: <P>This second paragraph proves that normal text is still
```

```
14: untouched.</P>
15: <P><B>CAUTION:</B></P>
16: <BLOCKQUOTE>This is a caution.</BLOCKQUOTE>
17: </BODY>
18: </HTML>
```

Abbildung 19.7 zeigt, wie die HTML-Datei in einem Web-Browser aussieht.

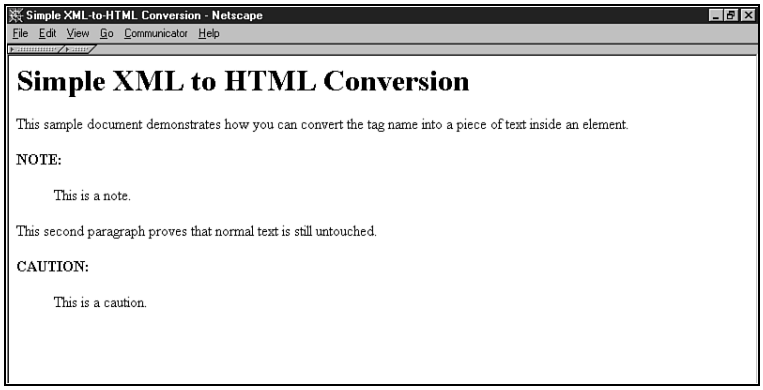


Abbildung 19.7: Das Element mit dem Präfix in HTML.

Komplexe Präfixe

Im vorigen Beispiel haben wir den Namen des Elements verwendet, um diesem ein Präfix zuzuordnen. Wir wollen noch einen Schritt weitergehen und etwas ganz anderes mit diesen `special`-Elementen tun. Wir verwenden dieselbe XML-Datei (Listing 19.5), aber ein anderes Stylesheet. Um das Listing zu kürzen (und Ihnen nicht immer wieder dieselben `DOC`-Elementspezifikationen zu zeigen), werde ich nur den Teil der Datei zeigen, der hier relevant ist.

Listing 19.11: DSSSL-Rezept-Datei 2, komplexe Präfixe

```
1:  <!DOCTYPE style-sheet PUBLIC
2:             "-//James Clark//DTD DSSSL Style Sheet//EN"[
3:  <!ENTITY lt "&#38;#60;">
```

```
4:  <!ENTITY gt "&#62;">
5:  ]>
6:
7:  <![CDATA[
8:  (define RED-ON (make formatting-instruction
9:              data: "<FONT COLOR='RED'>"))
10:  (define RED-OFF (make formatting-instruction data: "</FONT>"))
11:  (define RULE  (make formatting-instruction
12:              data: "<HR SIZE=5 NOSHADE WIDTH=300>"))
13: ]]>
14:
15: (define (make-special-para)
16:   (make sequence
17:     RULE
18:     (make element
19:       gi: "P"
20:       attributes: '(("ALIGN" "center"))
21:     (make element
22:       gi: "B"
23:       (literal (string-append (gi) ":"))))
24:     (make element
25:       gi: "BLOCKQUOTE"
26:       attributes: '(("ALIGN" "center"))
27:       (process-children)
28:        ) RULE ))
29:
30: (element NOTE (make-special-para))
31:
32: (element CAUTION (make sequence RED-ON (make-special-para)
33:     RED-OFF))
34:
35:
36:  (element PARA (make element gi: "P" ))
```

Jetzt verwende ich dieselbe `make-special-para`-Prozedur, allerdings in erweiterter Form. Als erstes habe ich eine Formatanweisung definiert:

```
(define RULE  (make formatting-instruction
data: "<HR SIZE=5 NOSHADE WIDTH=300>"))
```

Das bedeutet, ich kann `RULE` als Abkürzung für diesen ganzen HTML-Code verwenden.

Beachten Sie, daß ich das Schlüsselwort `data` verwende, um zu verhindern, daß das Markup interpretiert wird. (Sie wissen, daß das Stylesheet ein SGML-Dokument ist. Das bedeutet, Sie müssen das gesamte Markup, das jade nicht sehen soll, verbergen.)

Auf dieselbe Weise bin ich für die Anweisungen `RED-ON` und `RED-OFF` vorgegangen, aber hier habe ich das Markup verborgen, indem ich beide in einen CDATA-Abschnitt geschrieben habe.

Ein `NOTE`-Element erhält jetzt unten und oben eine Linie, wird zentriert und mit dem Text »NOTE:« darüber versehen. Ein `CAUTION`-Element wird jetzt erweitert, so daß ich eine Sequenz von »`RED-ON make-special-para RED-OFF`« anlegen kann, wodurch Warnungen rot dargestellt werden, wie in Abbildung 19.8 gezeigt.

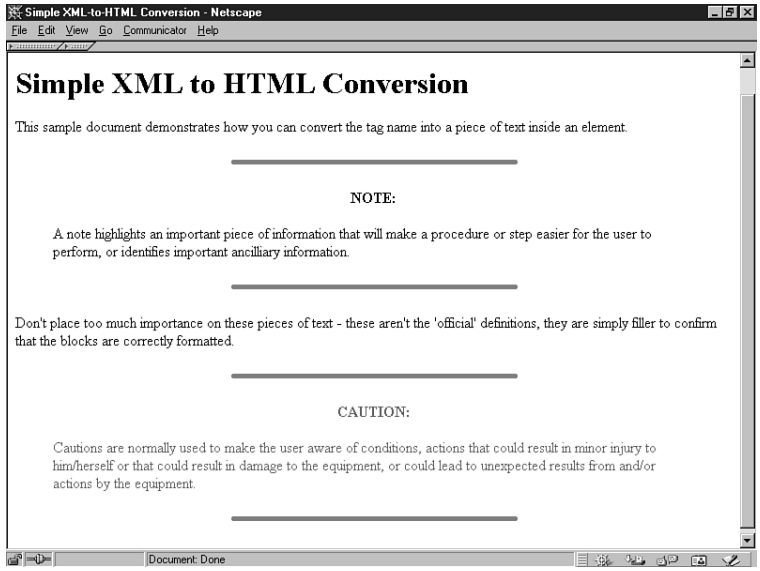


Abbildung 19.8: Komplexere Eigenschaften im HTML-Code.

Tabellen

Die beiden ersten Beispiele haben XML in HTML umgewandelt. Jetzt will ich eine XML-Datei in RTF umwandeln. In diesem Beispiel zeige ich etwas, was häufig Schwierigkeiten bereitet: eine Tabelle. Das XML-Eingabedokument sehen Sie in Listing 19.12, das DSSSL-Stylesheet in Listing 19.13.

Listing 19.12: XML-Rezept-Datei 3 (Tabellen)

```
1:  <?xml version="1.0">
2:  <!DOCTYPE DOC [
3:  <!ELEMENT DOC  (TITLE ¦ TABLE ¦ PARA)*>
4:  <!ELEMENT TITLE (#PCDATA)>
5:  <!ELEMENT TABLE (ROW)* >
6:  <!ELEMENT ROW   (CELL)*>
7:  <!ELEMENT CELL (#PCDATA)>
8:  <!ELEMENT PARA (#PCDATA ¦ TITLE)*>
9:  <!ELEMENT CAUTION (#PCDATA)>
10: ]>
11: <DOC>
12: <TITLE>Simple XML to RTF Conversion</TITLE>
13: <PARA>This sample document demonstrates how you can
14: create tables.</PARA>
15:
16: <TABLE>
17: <ROW><CELL>Cell 1.1</CELL><CELL>Cell 1.2</CELL>
18:     <CELL>Cell 1.3</CELL></ROW>
19: <ROW><CELL>Cell 2.1</CELL><CELL>Cell 2.2</CELL>
20:     <CELL>Cell 2.3</CELL></ROW>
21: <ROW><CELL>Cell 3.1</CELL><CELL>Cell 3.2</CELL>
22:     <CELL>Cell 3.3</CELL></ROW>
23: <ROW><CELL>Cell 4.1</CELL><CELL>Cell 4.2</CELL>
24:     <CELL>Cell 4.3</CELL></ROW>
25: <ROW><CELL>Cell 5.1</CELL><CELL>Cell 5.2</CELL>
26:     <CELL>Cell 5.3</CELL></ROW>
27: <ROW><CELL>Cell 6.1</CELL><CELL>Cell 6.2</CELL>
28:     <CELL>Cell 6.3</CELL></ROW>
29: <ROW><CELL>Cell 7.1</CELL><CELL>Cell 7.2</CELL>
30:     <CELL>Cell 7.3</CELL></ROW>
31: </TABLE>
32:
33: <PARA>Now we're back to normal text again.</PARA>
34: </DOC>
```

Listing 19.13: DSSSL-Rezept-Datei 3 (Tabellen)

```
1:  <!DOCTYPE style-sheet
2:     PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">
3:  (define debug
4: (external-procedure "UNREGISTERED::James Clark//Procedure::debug"))
5:
6:  (root
7:    (make simple-page-sequence
8:     left-margin:            2cm
9:     font-size:              12pt
10:    line-spacing:           14pt
11:     right-margin:           2cm
12:     top-margin:             2cm
13:     bottom-margin:          2cm
14:    (process-children)))
15:
16: (element (DOC TITLE)
17:     (make paragraph
18:        quadding:               'center
19:        font-size:              24pt
20:        line-spacing:           36pt
21:        space-after:            12pt
22:        font-weight:            'bold
23:       keep-with-next?:        #t
24:       (process-children)))
25:
26: (element PARA
27:    (make paragraph
28:        font-size:              12pt
29:        line-spacing:           16pt
30:        (process-children)))
31:
32: (element TABLE                    ;the processing of the table element
33:    (make table                      ;a data-driven table
34:        space-before: .25in
35:        space-after:  .5in
36:        table-border:  (make table-border line-thickness: 3pt)
37:        (make table-column width: 1.5in)  ;all columns before any rows
38:        (make table-column width: 1.5in)
39:        (make table-column width: 1.5in)
```

```
40:        (process-children))) ;child elements make row flow objects
41:
42: (element ROW                          ;a row in the table
43:     (make table-row
44:         (process-children)))  ;child elements make cell flow objects
45:
46: (element CELL                          ;a column in the row
47:     (make table-cell
48:         cell-before-row-border:    #t
49:         cell-before-column-border: #t
50:     (make paragraph
51:           quadding:       'center
52:           (process-children))))  ;content of the table cell
```

Wie Sie in Listing 19.12 sehen, mußte ich keine speziellen Tricks für die Formatierung der XML-Tabelle anwenden. Es ergibt keinen Sinn, den RTF-Code anzusehen. Abbildung 19.9 zeigt die konvertierte XML-Datei in Microsoft Word.

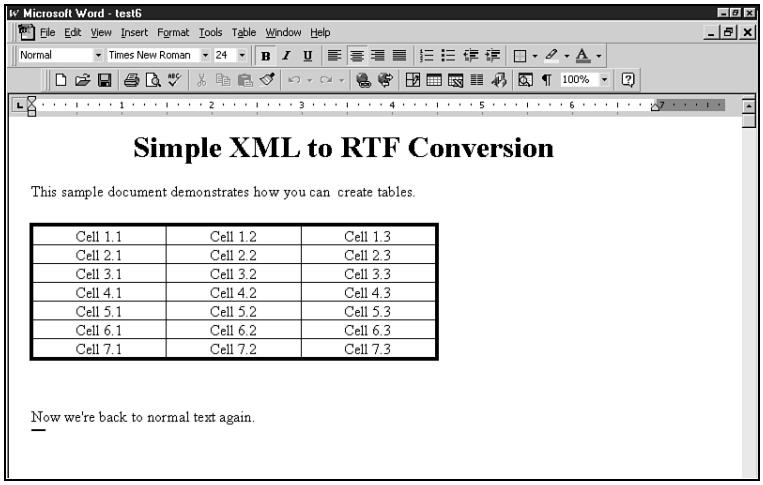


Abbildung 19.9: Die Tabellen in RTF.

Inhaltsverzeichnisse

Ich habe die Möglichkeit bereits erwähnt, ein Dokument mehrmals zu verarbeiten, um den Inhalt unterschiedlich zu formatieren. Und genau das werde ich jetzt tun. Ich habe dieses Beispiel bewußt einfach gewählt, weil ich mich auf den Modus-Mechanismus konzentrieren will, statt das ganze durch Hyperlinks vom Inhaltsverzeichnis zu den Einträgen zu verkomplizieren. Außerdem sollen Informationen in das Inhaltsverzeichnis aufgenommen werden, die nicht explizit im XML-Dokument vorhanden sind. Listing 19.14 zeigt das XML-Eingabedokument, Listing 19.15 das DSSSL-Stylesheet.

Listing 19.14: XML-Rezept-Datei 4 (Inhaltsverzeichnis)

```
1:  <?xml version="1.0">
2:  <!DOCTYPE DOC [
3:  <!ELEMENT DOC  (TITLE ¦ SECTION)*>
4:  <!ELEMENT SECTION (TITLE ¦ CAUTION ¦ NOTE ¦ PARA)*>
5:  <!ELEMENT TITLE (#PCDATA)>
6:  <!ELEMENT NOTE (#PCDATA)>
7:  <!ELEMENT PARA (#PCDATA ¦ TITLE)*>
8:  <!ELEMENT CAUTION (#PCDATA)>
9:  ]>
10: <DOC>
11: <TITLE>Simple XML to HTML Conversion</TITLE>
12: <SECTION>
13: <TITLE>Introduction</TITLE>
14: <PARA>This sample document demonstrates how you can
15: create a table of contents. This code uses two DSSSL modes. The
16: most important (toc) model allows us to process the document twice,
17: once to extract the text we need for the TOC and once for
18: the normal document formatting.
19: element.</PARA>
20: <NOTE>This is an extremely powerful feature of DSSSL that is well
21: worth learning.</NOTE>
22: <PARA>This second paragraph proves that normal text is still
23: untouched.</PARA>
24: </SECTION>
25: <SECTION><TITLE>Going Further</TITLE>
26: <CAUTION>Don't forget to use the debug features when developing
27: DSSSL style sheets with jade.</CAUTION>
28: <PARA>If you look at the code, you'll see I used two modes, not one,
```

```
29: I use a toc mode on the document to extract the TITLE elements to
30: include them in the TOC, and another mode to extract the text
31: contained within the TITLE element since I don't want
32: them to be formatted.</PARA>
33: </SECTION>
34: <SECTION><TITLE>And then?</TITLE>
35: <PARA>Even when XSL takes off, DSSSL is going to
36: be around for a long time yet (if for no other reason than
37: that it is an ISO standard). DSSSL (with jade) is still worth
38: learning.</PARA>
39: </SECTION>
40: </DOC>
```

Listing 19.15: DSSSL-Rezept-Datei 4 (Inhaltsverzeichnis)

```
1:  <!DOCTYPE style-sheet
2:       PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">
3:  <!DOCTYPE style-sheet
4:       PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">
5:  (declare-flow-object-class element
6:    "UNREGISTERED::James Clark//Flow Object Class::element")
7:  (declare-flow-object-class document-type
8:    "UNREGISTERED::James Clark//Flow Object Class::document-type")
9:  (declare-flow-object-class empty-element
10:   "UNREGISTERED::James Clark//Flow Object Class::empty-element")
11: (declare-flow-object-class formatting-instruction
12:   "UNREGISTERED::James Clark//Flow Object
13:              Class::formatting-instruction")
14: (define debug
15:   (external-procedure "UNREGISTERED::James Clark//Procedure::debug"))
16:
17: <![CDATA[
18: (define RED-ON (make formatting-instruction
19:      data: "<FONT COLOR='RED'>"))
20: (define RED-OFF (make formatting-instruction data: "</FONT>"))
21: (define RULE  (make formatting-instruction data: "<HR>"))
22: (define START (make formatting-instruction data: "<P>"))
23: (define STOP (make formatting-instruction data: "</P>"))
24: ]]>
25:
26: (define (make-special-para)
```

```
27:   (make sequence
28:     (make element
29:       gi: "P"
30:     (make element
31:       gi: "B"
32:       (literal (string-append (gi) ":"))))
33:     (make element
34:       gi: "BLOCKQUOTE"
35:       (process-children))))
36:
37: (element DOC
38:   (sosofo-append
39:     (make document-type name: "HTML"
40:           public-id: "-//W3C//DTD HTML 3.2//EN")
41:     (make element gi:  "HTML"
42:       (sosofo-append
43:         (make element gi:  "HEAD"
44:           (make element gi: "TITLE" (sosofo-append
45:                    (literal "Simple XML-to-HTML Conversion")))
46:         ))
47:       (make element gi:  "BODY"
48:         (make sequence
49:           (make element gi:  "H2"  (sosofo-append
50:                  (literal "Table of Contents")))
51:           (with-mode toc (process-matching-children 'section))
52:           (process-children))))))
53:
54: (mode extract-title-text (element (TITLE) (process-children)))
55:
56: (mode toc
57:   (element section
58:     (make sequence
59:       START
60:       (literal "Section ")
61:       (literal (format-number (child-number) "1"))
62:       (literal " ... ... ... ... ... ")
63:       (with-mode extract-title-text
64:    (process-first-descendant "TITLE"))
65:       STOP)))
66:
67: (element (DOC TITLE)
68:   (make sequence RULE (make element gi: "H2" )))
69:
70: (element (SECTION TITLE) (make element gi: "H3" ))
71:
72: (element NOTE  (make-special-para))
```

```
73:
74: (element CAUTION
75:   (make sequence
76:     RED-ON (make-special-para) RED-OFF))
77:
78: (element PARA (make element  gi: "P"))
```

Das Geheimnis hinter dem Ganzen ist ein sogenannter Modus. Ein Modus ist letztlich nichts anderes als die Verarbeitung eines Dokuments auf eine andere Weise. Zum Anlegen des Inhaltsverzeichnisses brauche ich zwei Modi.

Ein Modus ist eine benannte Menge an Verarbeitungsanweisungen. Durch die Einrichtung von Bedingungen, wann ein Modus ausgelöst wird, können Sie eine bedingte Verarbeitung implementieren. Durch die Angabe verschiedener Modi können Sie ein Dokument mehrmals, auf jeweils unterschiedliche Weise verarbeiten. Beispielsweise könnten Sie einen Modus verwenden, um den vollständigen Inhalt eines Dokuments zu verarbeiten, einen anderen zum Extrahieren und Formatieren einiger bestimmter Elemente (etwa zum Anlegen eines Inhaltsverzeichnisses oder eines Index).

Die einzige Aufgabe dieses ersten Modus ist, die Kindelemente des `title`-Elements auszuwählen. Weil diese Elemente keine Kindelemente haben, verarbeitet die `process-children`-Anweisung einfach den darin enthaltenen Text:

```
(mode extract-title-text
      (element (TITLE)
          (process-children)))
```

Jetzt brauche ich einen zweiten Modus, der angibt, wie dieser extrahierte Text behandelt werden soll. In diesem Modus lege ich das eigentliche Inhaltsverzeichnis an:

```
(mode toc
  (element section
    (make sequence
      START
      (literal "Section ")
      (literal (format-number (child-number) "1"))
      (literal " ... ... ... ... ... ")
      (with-mode extract-title-text
        (process-first-descendant "TITLE"))
      STOP)))
```

Die Formatspezifikation wird durch ein SECTION-Element ausgelöst. Anschließend erzeugte ich eine Sequenz aus einem öffnenden P-Tag durch Anwendung der START-Prozedur, literalem Text, der Abschnittsnummer, dem Text aus dem Titel und einem schließenden P-Tag durch die STOP-Prozedur. Zum Inhalt dieser Spezifikation gibt es zwei Dinge zu sagen:

- Die Nummer – Ich habe die Abschnitte nicht numeriert (was im Stylesheet ebenfalls möglich wäre), aber jedes Element hat seinen Platz im Baum: die `child-number`. Ich wandle die `child-number` in eine `format-number` um, die angibt wie diese Nummer angezeigt werden soll. Das könnte 1 sein (Dezimalzahlen: 1, 2, 3), »A« (Großbuchstaben: A, B, C), »a« (Kleinbuchstaben: a, b, c), »I« (große römische Ziffern: I, II, III, IC) oder »i« (kleine römische Ziffern: i, ii, iii, iv).
- Die Verwendung des Modus – Ich teile der DSSSL-Engine den Modus mit Hilfe des Schlüsselworts `with-mode` mit und weise sie dann an, nur das `TITLE`-Element des ersten Kindelements zu verarbeiten. Es gibt gar keine anderen Kind-`TITLE`-Elemente, aber ich will nicht den ganzen Abschnittstext berücksichtigen (was der Fall wäre, hätte ich `process-children` angegeben).

Dies ist eine XML-in-HTML-Umwandlung, deshalb können wir den Ausgabecode anzeigen, den Sie in Listing 19.14 sehen. Abbildung 19.10 zeigt die Datei im Web-Browser.

Querverweise

Im letzten dieser Beispiele soll wieder eine XML-in-RTF-Umwandlung gezeigt werden.

Bisher habe ich nur Elemente verwendet, außer im letzten Beispiel die Position eines Elements im Elementbaum, um zusätzliche Informationen zu gewinnen. Diese zusätzliche Information ist einer Attributinformation ganz ähnlich.

In diesem Beispiel werde ich noch einen Schritt weiter gehen und die in Attributen enthaltene Information nutzen. Betrachten Sie das XML-Dokument in Listing 19.16.

Listing 19.16: XML-Rezept-Datei 5 (Querverweise)

```
1:  <?xml version="1.0">
2:  <!ELEMENT doc (TITLE ¦ SECTION)*>
3:  <!ELEMENT SECTION (TITLE ¦ PARA)*>
4:  <!ELEMENT TITLE (#PCDATA)>
5:  <!ELEMENT PARA (#PCDATA ¦ NOTE ¦ XREF)*>
```

```
6:  <!ELEMENT NOTE (#PCDATA)>
7:  <!ATTLIST NOTE ID ID #IMPLIED>
8:
9:  <!ELEMENT XREF EMPTY>
10: <!ATTLIST XREF REF CDATA #IMPLIED>
11: ]>
12: <doc>
13: <TITLE>Test Document</TITLE>
14: <SECTION><TITLE>Starting</TITLE>
15: <PARA>This paragraph simply contains some text
16: (<NOTE ID="N1">Note 1 </NOTE>
17: this is an embedded note) that we will use to wrap the
18: note we're going to cross-reference.</PARA></SECTION>
19: <SECTION><TITLE>Going On</TITLE>
20: <PARA>In this second paragraph, we'll include a cross-reference
21: <XREF REF="N1"> to the previous note.</PARA>
22: </SECTION>
23: </DOC>
```

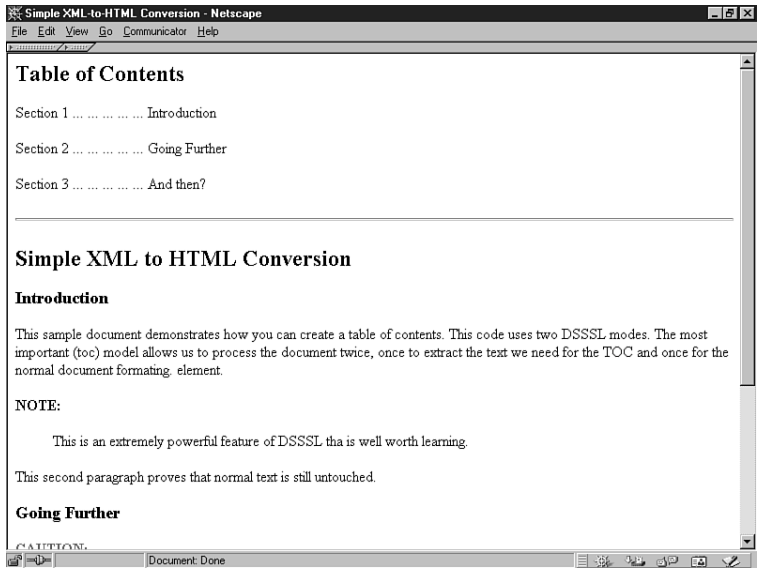


Abbildung 19.10: Das Inhaltsverzeichnis.

Dieser XML-Code enthält keine Überraschungen, aber beachten Sie, wie ich dem `NOTE`-Element in der DTD ein `ID`-Attribut hinzugefügt und ein neues `XREF`-Element mit `REF`-Attribut angelegt habe. Im eigentlichen Dokument hat das `NOTE`-Element den ID-

Attributwert NI, der auch dem REF-Attribut des XREF-Elements zugewiesen wird. Im DSSSL-Stylesheet verwende ich diese Zuordnung, um den in NOTE enthaltenen Text zu extrahieren. Listing 19.17 zeigt das DSSSL-Stylesheet.

Listing 19.17: DSSSL-Rezept-Datei 5 (Querverweise)

```
1:  <!DOCTYPE style-sheet
2       PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">
3:  <!DOCTYPE style-sheet
4:      PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">
5:
6:  (define debug
7:   (external-procedure "UNREGISTERED::James Clark//Procedure::debug"))
8:
9:  (element doc
10:    (make simple-page-sequence
11:     page-width: 15cm
12:     page-height: 20cm
13:     left-margin: .5cm
14:     right-margin: .5cm
15:     top-margin: 2cm
16:     bottom-margin: 2cm
17:     header-margin: 1cm
18:     footer-margin: 1cm
19:     center-footer:  (make sequence
20:                       (literal "Page " )
21:                       (page-number-sosofo))
22:    left-header:  (with-mode head (make sequence
23:              font-size: 10pt
24:              line-spacing: 14pt
25:              font-posture: 'italic
26:              (process-first-descendant "TITLE")))))
27:
28: (element PARA (make paragraph font-size: 14pt space-after: 5pt))
29:
30: (element (DOC TITLE) (make paragraph font-size: 18pt
31:            space-before: 6pt space-after: 10pt))
32: (element (SECTION TITLE) (make paragraph font-size: 14pt
33:            space-before: 6pt space-after: 10pt))
34:
35: (mode head
```

```
36:   (element TITLE
37:     (make paragraph font-size: 10pt space-before: 6pt
38:           space-after: 10pt)))
39:
40: (element NOTE
41:   (make sequence
42:     font-posture: 'italic
43:     font-size: 12pt
44:     (process-children)))
45:
46: (element XREF
47:   (make sequence
48:     (literal "(see ")
49:     (with-mode #f (process-children-trim)))
50:     (process-element-with-id
51:      (attribute-string "REF"))
52:      (literal ")")))
```

Um das `XREF`-Element zu erhalten und den String `Note 1` anzuzeigen, der aus dem `NOTE`-Element mit dem entsprechenden `ID`-Attributwert extrahiert wird, verwende ich die folgende Spezifikation:

```
(element XREF
  (make sequence
    (literal "(see ")
     (with-mode #f (process-children-trim)))
     (process-element-with-id
      (attribute-string "REF"))
    (literal ")")))
```

Ich verwende einen Modus, besser gesagt, ich verwende explizit keinen Modus, indem ich `with-mode` auf `false` (`#f`) setze. Anschließend verarbeite ich das Element mit der id (process-element-with-id), die mit dem Wert im String des `REF`-Attributs übereinstimmt (`attribute-string "REF"`). Beachten Sie, daß ich statt `process-children process-children-trim` verwende. Dadurch werden zusätzliche Leerzeichen, Tabulatoren oder CRs um den extrahierten String entfernt.

Noch zwei Dinge habe ich im letzten Moment eingefügt, eine Kopf- und eine Fußzeile für die Seite.

Die Fußzeile ist extrem einfach, weil eine `simple-page-sequence` drei Fußzeilen-Eigenschaften besitzt. Ich muß nur noch die gewünschte Eigenschaft auswählen und ein weiteres fertiges Flußobjekt zuordnen, eine `page-number-sosofo`:

```
center-footer:  (make sequence
                 (literal "Page " )
                 (page-number-sosofo))
```

Der Seitenkopf sollte ein bißchen interessanter sein, deshalb habe ich einen Modus angelegt, der das Titelelement des Dokuments formatiert:

```
(mode head
  (element TITLE
    (make paragraph font-size: 10pt space-before: 6pt
space-after: 10pt)))
```

Der Modus selbst ist nichts Besonderes – einfach nur genug, um eine grundlegende Formatierung für den Text festzulegen. Die eigentliche Arbeit ist die Angabe der entsprechenden Kopfzeileneigenschaften, wo ich den Modus aufrufe:

```
left-header:  (with-mode head (make sequence
                 font-size: 10pt
                 line-spacing: 14pt
                 font-posture: 'italic
                 (process-first-descendant "TITLE")))
```

Fertig. Da es sich um eine XML-in-RTF-Umwandlung handelt, können wir uns die Anzeige des RTF-Codes sparen und betrachten jetzt das Layout in Microsoft Word (glauben Sie mir, die Kopf- und Fußzeilen sind vorhanden, auch wenn man sie nicht sieht) in Abbildung 19.11.

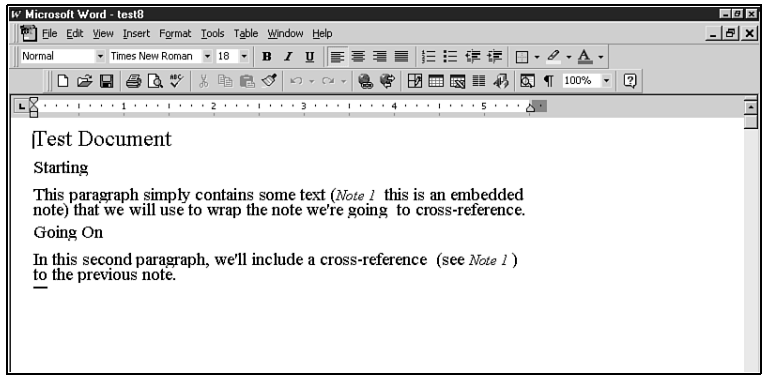


Abbildung 19.11: RTF-Rezept-Datei 5 – Anzeige (Querverweise).

Zusammenfassung

Dieses Kapitel hat sich mit XML und DSSSL beschäftigt und gezeigt, wie Sie mit jade XML-Code in RTF (zur Anzeige in Microsoft Word), MIF (zur Anzeige im Adobe FrameMaker) oder sogar in HTML umwandeln können. Ich habe hier nicht alle in jade möglichen Konvertierungen gezeigt (es gibt beispielsweise ein ausgezeichnetes TeX-Backend). Ich habe Ihnen auch nicht viel über DSSSL erzählt und die technischen Details bewußt sparsam eingesetzt. Mit Hilfe einiger einfacher Beispiele (im Web gibt es noch viel mehr davon, und die DSSSL-Mailingliste ist längst Quelle für Unterstützung und Rat der besten Experten auf diesem Gebiet) habe ich Ihnen gezeigt, daß mit DSSSL die interessantesten Dinge möglich sind – egal, was mit XSL passiert.

F&A

F Warum und wann sollte ich bei der Ausführung von jade die Parameter `-wxml` und `-wno-valid` benutzen?

A *jade ist eigentlich ein SGML-Werkzeug und erwartet, daß Sie Symbole für die Tag-Minimierung in Ihrer DTD verwenden. Um zu verhindern, daß jade sich permanent beschwert, verwenden Sie den Parameter `-wxml`.*

Normalerweise wertet jade Dokumente vor der Verarbeitung aus. Anders als in SGML müssen die XML-Dokumente dazu nicht gültig (aber wohlgeformt) sein. Um die Auswertung in jade zu unterdrücken, verwenden Sie den Parameter `-wno-valid`.

F Warum müssen Markup-Zeichen (wie <) in einem DSSSL-Stylesheet verborgen werden?

A *Bei dem von jade verwendeten DSSSL-Stylesheet handelt es sich um ein ganz normales SGML-Dokument. Deshalb müssen Sie die Katalogdatei anpassen oder anlegen, wenn Sie jade ausführen wollen, um sicherzustellen, daß es die DTD für die Stylesheets findet. Weil es sich um ein Standard-SGML-Dokument handelt, gelten die normalen Regel für das Escaping von Markup-Zeichen, die nicht von jade interpretiert werden sollen. Aber weil das Escape nicht auf die normale Weise stattfinden kann (beispielsweise mit Zeichen-Entities), müssen Sie das Markup hinter dem Schlüsselwort `data` oder in einem CDATA-Abschnitt verstecken.*

F **Warum verwendet ein DSSSL-Stylesheet** `start` **und** `end` **statt** `left` **und** `right`**?**

A *DSSSL ist international. Ein Stylesheet kann für die Formatierung von Text von links nach rechts, von rechts nach links und sogar für eine vertikale Darstellung verwendet werden. Die Angabe von »links« und »rechts« wäre in diesem Kontext sehr verwirrend, deshalb werden in den Stylesheets richtungsneutrale Begriffe verwendet, wie beispielsweise* `start`*,* `end`*,* `before` *und* `after`*.*

Übungen

1. Laden Sie jade herunter, und führen Sie das Testbeispiel der Distribution aus
2. Um sich einen Eindruck zu verschaffen, wie leistungsfähig ein DSSSL-Stylesheet sein kann, suchen Sie sich ein beliebiges HTML-Dokument aus. Laden Sie jade herunter, und suchen Sie nach dem Stylesheet `html32hc.dsl`. Mit diesem Stylesheet wandeln Sie HTML in RTF oder MIF um und zeigen das Ergebnis in einem geeigneten Softwarepaket an.
3. Nehmen Sie das Beispiel-Stylesheet für das Inhaltsverzeichnis aus unseren Rezepten (Nummer 3), und versuchen Sie, daß Stylesheet so zu erweitern, daß es einen aktiven Link zwischen dem Eintrag im Inhaltsverzeichnis und dem eigentlichen Abschnittstitel gibt. Sie brauchen dazu ein `link`-Flußobjekt und eine `current-node-page-number-sosofo`; das Stylesheet `html32.hc` kann Ihnen einige interessante Hinweise geben.
4. Versuchen Sie im Beispiel 5 (Querverweise) das HTML-Markup so zu ergänzen, daß der Querverweis zu einem Hyperlink wird.

Darstellung von XML mit XSL

Woche 3

In Kapitel 18 haben Sie CSS (*Cascading Style Sheets*) und die Verwendung von CSS-Code zur Formatierung von XML-Dokumenten kennengelernt – und das, obwohl diese eigentlich für HTML-Dokumente vorgesehen waren. In Kapitel 19 wurde DSSSL vorgestellt, mit der Möglichkeit, komplexere Layouts für XML-Dokumente damit zu schaffen. Heute werden wir unsere Betrachtung der XML-Stilsprachen mit dem eigentlichen XSL (*Extensible Style Language*) vervollständigen

In diesem Kapitel geht es um die folgenden Dinge:

- Die erste XSL-Version
- Einfache Darstellungen mit XSL2-Formatobjekten
- Element- und Attributauswahl
- Einführung in die leistungsfähige bedingte Verarbeitung von XSL

XSL1

Die erste Version von XSL (die im Oktober 1997 als inoffizielle Komitee-Skizze veröffentlicht wurde) war eine seltsame Mischung aus den konzeptionellen Modellen und Flußobjekten von DSSSL-o und ganz neuen Flußobjekten, die die HTML-Elemente nachbildeten – und das alles in XML-Syntax formuliert.

In früheren Kapiteln habe ich die Begriffe »Darstellung« und »Formatierung« recht undifferenziert verwendet. In diesem Kapitel geht es bei der *Darstellung* (Rendering) für Text um die Anzeige auf dem Bildschirm oder die Ausgabe auf dem Drucker.

Nachdem Sie die Leistungsfähigkeit von DSSSL erfahren haben (und mit seiner exotischen Syntax zurechtkommen), ist der Abstieg zu dieser ersten XSL-Version ein scheinbar akzeptabler Kompromiß. Vergleichen Sie Listing 20.1, das einen Ausschnitt aus XSL1-Code zeigt, mit Listing 20.2, das den äquivalenten Code in DSSSL-o enthält.

Listing 20.1: Ein Ausschnitt aus XSL1-Code

```
1: <rule>
2:   <target-element type="DOC"/>
3:   <scroll>
4:     <children/>
5:   </scroll>
6: </rule>
```

Listing 20.2: Das Äquivalent zu Listing 20.1 in DSSSL-o-Code

```
1: (element DOC
2:   (make scroll
3:     (process-children)
4:   )
5: )
```

Die Codeausschnitte in den Listings 20.1 und 20.2 erledigen genau dasselbe; sie wählen das `DOC`-Element aus, erzeugen ein `scroll`-Flußobjekt und verarbeiten dann die Kindelemente des `DOC`-Elements. Was die Funktionen betrifft, war XSL1 ein kleiner Schritt in die richtige Richtung. Es ist in der Lage, viele der DSSSL-o-Flußobjekte (obwohl einige wichtige weggelassen wurden) mit HTML-Flußobjekten zu kombinieren, wodurch eine gewisse Abwärtskompatibilität mit CSS entstand. Der wichtige Vorteil war, daß XSL1 in Standard-XSL-Syntax formuliert ist, so daß man nicht mehr gezwungen war, eine völlig neue Syntax zu erlernen. Damit eröffneten sich zahlreiche Möglichkeiten, weil nicht nur XML-Code, sondern auch sein Stilcode automatisch erzeugt werden konnte.

Ich werde XLS1 nicht detailliert beschreiben, weil es eigentlich nicht mehr existiert. Mit der Einführung von XSL2 ist die ältere XSL-Version völlig veraltet.

XSL2

XSL2 wurde zunächst im Juli 1998 als Entwurf veröffentlicht. Die neueste und aktuelle Version ist ein Arbeitsentwurf vom 16. Dezember 1998 (von jetzt an werde ich es nur noch als XSL bezeichnet, weil alle älteren Versionen damit veraltet sind).

Es gibt noch relativ viele unfertige Abschnitte in diesem Entwurf, und es müssen noch einige Fragen beantwortet werden, bevor dieser Vorschlag zum Standard wird. Auch beim vorhergehenden Release erfolgte die Veröffentlichung einen Monat später als die Entwurfsfertigstellung, und ich glaube, die Autoren dachten einfach, es wäre sinnvoll, wenigstens eine halbfertige Version zu präsentieren. Glücklicherweise haben sich die Software-Entwickler dadurch nicht aufhalten lassen. Innerhalb eines Monats nach dem neuesten Release gab es einen kostenlosen XSL-Prozessor (den XSLProcessor, den Sie unter `http://www.inria.fr/koala/XML/xslProcessor` finden). Es handelt sich dabei um einen Java-XSL-Prozessor, der eine XSL-Datei und eine XML-Datei entgegennimmt und daraus eine oder mehrere HTML-Dateien erzeugt.

Indelv (http://www.indelv.com) gab ein sogenanntes Technology-Introduction-Preview Release-Paket heraus, das XML-Code unter Verwendung von XSL-Code verarbeitet und anzeigt. Der aktuelle Release dieser Software weist wesentliche Einschränkungen auf (einige der Bearbeitungsfunktionen wurden deaktiviert, aber es gibt beschränkte VML-Unterstützung, was sehr eindrucksvoll ist), und mit ein bißchen Ausprobieren erhält man schließlich die Software, die man für die Verarbeitung eigenen Codes braucht.

Bei diesem Ausprobieren muß die Datei navigation.xml abgeändert werden. Fügen Sie am Dateiende folgendes ein:

```
<branch collapsed="no">
  <branch-title
     href="simon.xml">Simon's Files
  </branch-title>
  <branch>
    <branch-title
       href="simon1.xml">File 1
    </branch-title>
  </branch>
  <branch>
    <branch-title
       href="simon2.xml">File 2
    </branch-title>
  </branch>
</branch>
```

Ersetzen Sie die Dateinamen simon1.xml und simon2.xml durch die Namen Ihrer Dateien, dann wird Indelv beliebige XML-Dateien für Sie lesen.

Bevor wir weiter in die Theorie eintauchen, betrachten wir ein kurzes Beispiel für eine XML-Datei und die ihr zugeordnete XSL-Datei. Listing 20.3 zeigt ein XML-Beispiel, Listing 20.4 den XSL-Code und Abbildung 20.1 das Ergebnis in Indelv.

Listing 20.3: Beispiel für eine XML-Datei

```
1:  <?xml version="1.0"?>
2:  <?xml:stylesheet type="text/xsl" href="simon1.xsl"?>
3:  <content>
```

```
4:
5:  <hr/>
6:
7:  <title>THE MORON</title>
8:  <poem>
9:  <line>See the happy moron</line>
10: <line>He doesn't give a damn.</line>
11: <line>I wish I were a moron,</line>
12: <line><stress>My God! perhaps I am.</stress></line>
13: </poem>
14:
15: </content>
```

Beachten Sie, wie das XSL-Stylesheet durch eine XML-Verarbeitungsanweisung mit dem XML-Dokument verknüpft ist (das ist die Methode der Wahl für die Verknüpfung der beiden Dateien). Sie können einer XML-Datei mit demselben Mechanismus ein CSS-Stylesheet zuordnen (die Datei wäre dann eine CSS-Datei, und der Typ wäre `text/css`).

Der neueste Release des XSL-Entwurfs ist so neu, daß es bisher wenige Werkzeuge geschafft haben, die Syntaxänderungen von XSL zu implementieren. Im allgemeinen sollten Sie alle Vorkommen von `xsl:process-children` durch `xsl:apply-templates` ersetzen, um die Codebeispiele in diesem Kapitel an die neue Syntax anzupassen.

Beachten Sie jedoch, daß der IE5 keine Flußobjekte unterstützt, Sie werden also nicht in der Lage sein, viel von diesem Code anzuzeigen.

Listing 20.4: Die zugeordnete XSL-Datei

```
1:  <xsl:stylesheet
2:    xmlns:xsl="http://www.w3.org/TR/WD-xsl"
3:    xmlns:fo="http://www.w3.org/TR/WD-xsl/FO"
4:    result-ns="fo">
5:
6:    <xsl:template match="/">
```

```
7:      <fo:sequence
8:          font-size=”12pt”
9:          indent-end=8pt
10:         indent-start=8pt>
11:     <xsl:process-children/>
12:     </fo:sequence>
13:   </xsl:template>
14:
15:   <xsl:template match=”hr”>
16:     <fo:rule-graphic
17:         space-before-optimum=”12pt”
18:         space-after-optimum=”12pt”
19:         graphic-line-thickness='2pt'>
20:       <xsl:process-children/>
21:     </fo:rule-graphic>
22:   </xsl:template>
23:
24:   <xsl:template match=”poem”>
25:     <fo:block-level-box
26:         indent-start=”1in”
27:         indent-end=”1in”
28:         background-color='blue'
29:         graphic-line-thickness='3pt'>
30:     <xsl:process-children/>
31:     </fo:block-level-box>
32:  </xsl:template>
33:
34:  <xsl:template match=”line”>
35:     <fo:block
36:         font-style=”italic”
37:         space-before-optimum=”2pt”
38:         space-after-optimum=”2pt”>
39:       <xsl:process-children/>
40:     </fo:block>
41:   </xsl:template>
42:
43:  <xsl:template match=”title”>
44:     <fo:block
45:        font-size=”14pt”
46:        font-weight=”bold”
47:        space-before-optimum=”12pt”
48:        space-after-optimum=”8pt”>This poem is called '
49:      <xsl:process-children/>
50:        <xsl:text> '</xsl:text>
51:    </fo:block>
52:  </xsl:template>
```

```
53:
54:  <xsl:template match="stress">
55:     <fo:sequence
56:        font-size="14pt"
57:        font-weight="bold"
58:        font-style="normal"
59:        color='white'>
60:      <xsl:process-children/>
61:     </fo:sequence>
62:  </xsl:template>
63:
64: </xsl:stylesheet>
```

Beachten Sie, daß es sich bei dem XSL-Stylesheet um ein wohlgeformtes XML-Dokument handelt. Es kann analysiert werden, weil es eine Stylesheet-DTD gibt, die als Anhang des Entwurfs mitgeliefert wurde. (Das allein ist ein großer Schritt vorwärts, weil Sie jetzt prüfen können, ob die Syntax Ihrer Stylesheets korrekt ist, indem Sie dieselben Werkzeuge wie für die Prüfung Ihrer XML-Dateien anwenden – eine Möglichkeit, die es leider für CSS nicht gibt).

Das Interessante bei dieser neuen Stylesheet-Syntax ist, daß sie nicht nur wohlgeformt ist, sondern auch gültig sein sollte. Wenn Sie mit Ihrem Web-Browser den XSL-Vorschlag besuchen (`http://www.w3c.org/TR/WD-xsl`) und die Datei auf Ihrer Festplatte ablegen, können Sie die in Anhang A enthaltene DTD extrahieren und sie zum Debugging Ihrer Stylesheets heranziehen. Damit ersparen Sie sich großen Aufwand, diese langweilig einfachen Syntaxfehler zu suchen, die so leicht gemacht werden und so schwer zu erkennen sind.

Der erste Teil des Stylesheet ist mehr oder weniger Standard. Dieser Prolog ist der standardmäßige XML-Dokument-Prolog, ergänzt um einige Namensraumdeklarationen. Der XSL-Namensraum sollte immer so wie hier gezeigt verwendet werden, aber der Ergebnisnamensraum kann beliebig sein.

Es gibt noch zahlreiche Probleme in Hinblick auf Namensräume, aber hier wollen wir das nutzen, was es gibt. Theoretisch sollte bei der Deklaration eines eigenen Namensraums eine zugehörige URL bereitgestellt werden, die auf eine Erklärung der Semantik der Elemente verweist. Das wird jedoch nicht überprüft; deshalb können beliebige URLs angegeben werden, bis es einen auflösenden Mechanismus dafür gibt.

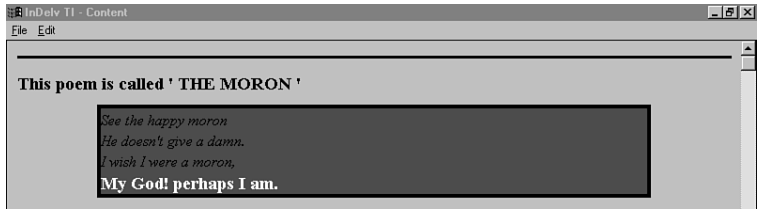


Abbildung 20.1: Eine Darstellung der XML-Datei mit Eigenschaften, die mittels einer XSL-Datei angewendet wurden.

Schablonenregeln

Der Grundbaustein der XSL-Stylesheets ist die Schablonenregel. Eine Schablonenregel beschreibt, wie ein XML-Elementknoten (dieses Element und alle darin enthaltenen Elemente) in einen XSL-Elementknoten umgewandelt wird, welcher dargestellt werden kann. Vergessen Sie nicht, daß Sie aus Teilen des Dokumentbaums neue Bäume erzeugen können (siehe Kapitel 19). Der Elementknoten könnte ein ganzer Zweig des Dokumentbaums sein, aber auch das gesamte XML-Dokument, das auf andere Weise verarbeitet wurde (beispielsweise beim Anlegen eines Inhaltsverzeichnisses).

Eine Schablonenregel besteht aus zwei Komponenten:

1. Einem Muster, das den XML-Knoten (Element) im XML-Dokument kennzeichnet
2. Einer Aktion (Darstellung oder Verarbeitung), die die Umwandlung und die Darstellung des resultierenden Knotens (des Elements oder der Elemente, das bzw. die Sie identifiziert haben)

Wenn Sie das XML-Dokument nur unverändert weitergeben wollen, wäre das folgende vielleicht ausreichend, obwohl eine andere Anweisung (xsl:copy) verwendet wird, die Sie später in diesem Kapitel noch kennenlernen, um XML-Code zu übergeben oder eine hybride XML/HTML-Mischung zu erzeugen.

```
<xsl:template match="/">
   <fo:page-sequence>
     <xsl:apply-templates/>
   </fo:page-sequence>
 </xsl:template>
```

Viele der hier gezeigten XML-Codeabschnitte basieren auf Flußobjekten, weil das die einfachste und bevorzugte Methode ist, XML-Code darzustellen. Es ist jedoch nicht die einzige Möglichkeit, und Flußobjekte werden derzeit im Internet Explorer 5 nicht unterstützt. (Und man zweifelt daran, ob sie überhaupt irgendwann unterstützt werden, weil sie für Microsofts Intention, XML-Code in HTML-Code umzuwandeln, um ihn in einem Web-Browser anzeigen zu können, keine Bedeutung haben.) Später in diesem Kapitel werden Sie einige Beispiele für XSL-Stylesheets kennenlernen, die nicht auf Flußobjekten basieren und die in IE5 genutzt werden können.

Das Element `process-children` wird in einigen Beispielen dieses Kapitels verwendet, weil es aktuell unterstützt wird. Im neuesten XSL-Entwurf wurde dieses Element in `xsl:apply-templates` geändert, und dieses Element wird in der Beta Preview 2 des Microsoft Internet Explorer 5 unterstützt.

Häufig ist es sinnvoll, alle grundlegenden Stileigenschaften in der Wurzel des Dokuments zusammenzufassen, so daß alle nachfolgenden Elemente die Eigenschaften erben können. Dazu gibt man einfach einen Schrägstrich an, der das Wurzelelement repräsentiert:

```
<xsl:template match= "/">
```

Es gibt viele Möglichkeiten, Elemente zu suchen, unter anderem:

- Nach der ID
- Nach dem Elementnamen
- Nach dem Vorgänger
- Nach den Kindern
- Nach den Attributen

Suche eines Elements nach seiner ID

Die Syntax für die Suche eines Elements nach seiner ID ist dieselbe wie die Syntax für einen XPointer:

```
<xsl:template match="id([id-wert])">
```

Dabei ist `id-wert` der spezifische Wert des ID-Attributs, nach dem Sie suchen. Der XSL-Codeabschnitt im Stylesheet

```
<xsl:template match= "id(1)">
```

würde den folgenden XML-Codeabschnitt im XML-Dokument auswählen:

```
<line id="1">See the Happy Moron!</line>
```

Suche eines Elements nach seinem Namen

Das ist die einfachste Suche – das Quellelement wird anhand des Namens im `match`-Attribut erkannt.

Der folgende XSL-Code würde alle `line`-Elemente in dem XML-Dokument auswählen:

```
<xsl:template match="line">
```

Suche eines Elements nach seinem Vorgänger

In XSL können Sie ein Element nach seinem Elternelement suchen (eine kontextabhängige Suche), zum Beispiel:

```
<xsl:template match="line/stress">
```

Damit werden alle `stress`-Elemente gesucht, die ein `line`-Element als Elternelement haben.

Statt der unmittelbaren Beziehung Eltern/Kind könnten Sie auch nach weniger präzisen Vorgängern suchen. Das folgende etwa würde nach einem `stress`-Element suchen, das irgendwo im Baum unmittelbar oberhalb ein `line`-Element besitzt (Urgroßeltern, Großeltern usw.). Beachten Sie, daß hier zwei Schrägstriche verwendet werden:

```
<xsl:template match="line//stress">
```

Sie können auch verschachtelte Auswahlen treffen, an denen mehrere Elternelemente beteiligt sind. Das folgende etwa würde ein `stress`-Element suchen, das ein `line`-Element als Elternelement hat, das wiederum ein `poem`-Element als Elternelement hat (das `poem`-Element muß also das Großelternelement des `stress`-Elements sein):

```
<xsl:template match="poem/line/stress">
```

Suche nach mehreren Elementnamen

Wenn Sie mehrere Elemente gleichzeitig auswählen wollen, trennen Sie die Elementnamen einfach durch eine vertikale Linie voneinander ab, `|`:

```
<xsl:template match="poem | line | stress">
```

Damit werden alle `poem`-, `line`- und `stress`-Elemente angesprochen.

Suche nach einem Element nach seinen Attributen

Die Syntax für die Suche nach einem Attribut lautet:

```
<xsl:template match="element-name[@attribute-name="attribute-value"]">
```

Betrachten Sie das in Listing 20.5 gezeigte XML-Dokument:

Listing 20.5: Eine einfache XML-Datei mit Attributen

```
1: <chapter>
2:   <section number="1">
3:     <title>first</title>
4:     <para type="first">Some text</para>
5:     <para>Some <em>emphasized</em> text</para>
6:     <para>Another para with no attributes</para>
7:     <para type="last">Some more text</para>
8:   </section>
9:
10:   <section number="2">
11:     <title>two</title>
12:     <para type="first">Some text</para>
13:     <para>Some <em>emphasized</em> text</para>
14:     <para>Another para with no attributes</para>
15:     <para type="last">Some more text</para>
16:   </section>
17: </chapter>
```

Die folgenden Beispiele zeigen, wie Sie Attribute und ihre Werte nutzen, um Elemente aus der XML-Datei auszuwählen, die Sie in Listing 20.5 sehen:

- `<xsl:template match="para">` – Sucht alle `para`-Elemente (mit und ohne Attribute).
- `<xsl:template match="para[@type)]">` – Sucht alle `para`-Elemente mit `type`-Attribut (egal welchen Wert dieses Attribut hat).
- <xsl:template match="para[@type='first']"> – Sucht alle para-Elemente mit dem type-Attribut »first«.
- `<xsl:template match="section[@number= '1']/para[@type='first']">` – Sucht alle `para`-Elemente mit dem `type`-Attribut »`first`« und einem Elternelement von `section` mit einem `number`-Attribut mit dem Wert `1` (das erste `para`-Element in Listing 20.5).

Suche eines Elements nach seinen Kindelementen

Elemente können auch nach ihren Kindelementen gesucht werden (den enthaltenen Elementen).

Die Syntax lautet:

```
<xsl:template match="element-name[child-name]">
```

Beispielsweise sucht der folgende Ausdruck alle `line`-Elemente, die ein `stress`-Element enthalten:

```
<xsl:template match="line[stress]">
```

Suche eines Elements nach seiner Position

Sie können Elemente auch unter Verwendung von Positionsbezeichnern suchen.

Die Syntax dafür lautet:

```
<xsl:template match="element[position-description]"
```

Es gibt die folgenden Positionsbeschreibungen:

- `first-of-any()` Das Element muß das erste Geschwisterelement sein.
- `first-of-type()` Das Element muß das erste Geschwisterelement dieses Typs sein.
- `last-of-any()` Das Element muß das letzte Geschwisterelement sein.
- `last-of-type()` Das Element muß das letzte Geschwisterelement dieses Typs sein.
- `only-of-any()` Das Element darf keine Geschwisterelemente haben.
- `only-of-type()` Das Element darf keine Geschwisterelemente desselben Typs haben.
- `not-first-of-any()` Das Element darf nicht das erste Geschwisterelement sein.
- `not-first-of-type()` Das Element darf nicht das erste Geschwisterelement dieses Typs sein.
- `not-last-of-any()` Das Element darf nicht das letzte Geschwisterelement sein.
- `not-last-of-type()` Das Element darf nicht das letzte Geschwisterelement dieses Typs sein.

- `not-only-of-any()` Das Element muß ein oder mehrere Geschwisterelemente haben.
- `not-only-of-type()` Das Element muß ein oder mehrere Geschwisterelemente desselben Typs haben.

Beispielsweise ermittelt die folgende XML-Schablone das letzte `line`-Element:

```
<xsl:template match= "line[last-of-type()]"
```

Sie können diese Bezeichner auch mit dem Schlüsselwort `not` kombinieren, so daß das obige Beispiel so umgeschrieben werden könnte, daß alles gesucht wird außer dem letzten `line`-Element:

```
<xsl:template match= "line[not last-of-type()]"
```

Wildcard-Suchen

Eine Wildcard-Suche wählt beliebige Elemente aus. Eine Wildcard-Suchregel ist sinnvoll, um eine Regel anzuwenden, wenn keine andere Regel greift. (Der XSL-Prozessor setzt eine Standardregel voraus, wenn es keine spezielle Regel gibt, wie Sie später in diesem Kapitel noch erfahren werden.)

Die Syntax für eine Wildcard-Suche lautet:

```
<xsl:template match="*">
```

Die folgende Regel würde das gesamte Dokument auswählen, wenn Sie sichergehen möchten, daß wirklich alles berücksichtigt wird:

```
<xsl:template match="/ ¦ *">
```

Auflösung von Auswahlkonflikten

Manchmal gelten mehrere Regeln für dasselbe Quellelement. In diesem Fall wird immer die spezifischere Regel herangezogen. Die folgenden Regeln beispielsweise, die dasselbe Element auswählen, sind in der Reihenfolge dargestellt, wie spezifisch sie sind.

1. `id(line1)` – Der ID-Attributwert ist der spezifischste (beachten Sie, daß ID-Werte eindeutig sein müssen).
2. `line[attribute(type)='line-no', attribute(line-no)?'1']` – Wählt alle Elemente mit einem bestimmten Attribut/Attributwert-Paar aus (weil es sich nicht um ein ID-Attribut handelt, muß der Attributwert nicht eindeutig sein).

3. `line[attribute(type)='line-no']` – Wählt alle Elemente mit einem bestimmten Attributnamen aus.
4. `line` – Wählt alle Elemente mit diesem Elementnamen aus.
5. `*` – Wählt alle Elemente aus (eine Wildcard ist die am wenigsten spezifische Angabe).

Wenn Sie mehr Kontrolle über die Anwendung einer Regel brauchen, können Sie auch ein Prioritätsattribut einführen und ihm einen positiven ganzzahligen Wert zuweisen:

```
<xsl:template match="section/title" priority="1">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Je höher der Prioritätswert, desto höher die Priorität der Schablonenregel. Der Standardwert ist 0 (wenn Sie also keinen Prioritätswert angeben).

Die Standard-Schablonenregel

Der XML-Prozessor verarbeitet den XML-Dokumentbaum rekursiv, um einen Ausgabebaum anzulegen. Um zu verhindern, daß dieser rekursive Prozeß unterbrochen wird, wenn es keine geeignete Auswahlregel gibt, setzt der XSL-Prozessor eine Standardregel voraus. Die Syntax dieser Regel lautet:

```
<xsl:template match="/¦*">
        <xsl:apply-templates/>
</xsl:template>
```

Sie können eine eigene Regel entwickeln, um die Standardregel zu überschreiben, z.B.:

```
<xsl:template match= "/¦*">
 [fehlende regel]
</xsl:template>
```

Damit wird die Verarbeitung aller Elemente verhindert, für die es keine Stilregel gibt, und ein entsprechender String [`fehlende regel`] in den Ausgabebaum geschrieben. Das kann praktisch sein, um Elemente zu kennzeichnen, die Sie vergessen haben.

Formatobjekte

Aus Kapitel 19 wissen Sie, daß DSSSL-o Flußobjekte verwendet und XLS1 eine Kombination aus DSSSL- und HTML-Flußobjekten einsetzt. Die neuste XSL-Version stellt eigene Flußobjekte bereit, bezeichnet diese aber als *Formatobjekte*. Formatobjekte sind dem Namensraum `fo` zugeordnet.

Das Formatobjekt wird auf den Ergebnisbaumknoten im Musterteil des Elements angewendet:

```
<xsl:template match="[muster]">
    <fo:[Formatobjekt] ([Stileigenschaft]="[Wert]")*>
        [Verarbeitungsanweisungen]*
    </fo:[Formatobjekt]>
</xsl:template>
```

Layouts für Formatobjekte

Der aktuelle XSL-Entwurf beschreibt nur das `page-sequence`-Objekt und das `simple-page-master`-Objekt. Später werden weitere Formatobjekte eingeführt. Bisher gibt es die folgenden Formatobjekte:

- `page-sequence` – Ein page-sequence-Objekt agiert als Vorfahre mehrerer Seiten (gedruckter oder Bildschirmseiten).
- Ein `page-sequence`-Objekt enthält ein oder mehrere Formatobjekte oder `page-sequence`-Objekte. Es kann ein oder mehrere `single-page-master`-Kinder enthalten und bis zu sechs Kind-`queues`.
- In der Praxis dient `page-sequence` als Container für Stilregeln, die von dem restlichen Dokument geerbt werden können. Beispielsweise setzt die folgende Schablonenregel,

```
<xsl:template match="/">
  <fo:page-sequence
     font-family="times new roman,serif"
     font-size="12pt"
     background-color= "white'
     color= "black">
       <xsl:apply-templates/>
  </fo:page-sequence>
</xsl:template>
```

- die Schrift und die Farbe anderer Formatobjekte im Ergebnisbaum, es sei denn, es wird etwas anderes angegeben.

- `simple-page-master` – Ein simple-page-master-Formatobjekt beschreibt eine einfache Seite, die in sechs Bereiche unterteilt werden kann. Dieses Modell kann für die Druck- oder Bildschirmausgabe herangezogen werden.

Das Element muß ein `master-name`-Attribut haben, das einen der folgenden Werte annehmen kann:

`first` – Formatierung der ersten Seite einer Seitenfolge

`odd` – beim Ausdruck die linke Seite (Verso)

`even` – beim Ausdruck die rechte Seite (Recto)

`scrolling` – die Seiten für die Bildschirmanzeige

Inhalts-Formatobjekte

Die folgenden Abschnitte beschreiben einige der gebräuchlichsten Inhalts-Formatobjekte:

- `queue` – Ein queue-Flußobjekt ist eigentlich gar kein Formatobjekt, sondern ein Container für die anderen Formatobjekte, so daß diese einem bestimmten Bereich auf der Seite (oder dem Bildschirm) zugeordnet werden können.

  Eine `queue` kann nur ein Kind einer `page-sequence` sein. Es hat einen `queue-name`-Utility-Modus mit dem Wert `title`, `header`, `body`, `footer`, `start-side` oder `end-side`. Im `body` befindet sich der Hauptinhalt des XML-Dokuments. Abbildung 20.2 zeigt eine Anordnung der sechs `queue`-Bereiche auf einer Seite. Beachten Sie, daß der `queue`-Bereich `title` nur auf Medien angewendet werden kann, für die ein Scrolling möglich ist.

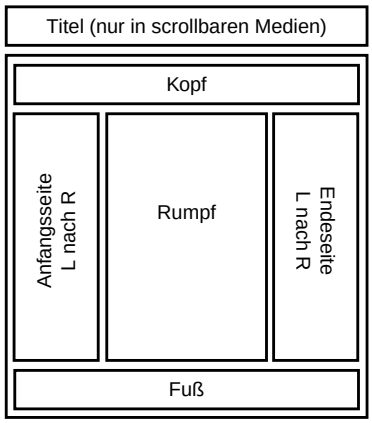


Abbildung 20.2:
Die sechs queue-Bereiche für den Aufbau einer Seite in XSL.

- `sequence` – Ein sequence-Formatobjekt gruppiert Formatobjekte, die eine gemeinsame Menge geerbter Eigenschaften gemeinsam nutzen. Der folgende Codeabschnitt stellt alle in ein em-Element eingeschlossenen Elemente kursiv dar:

```
<xsl:template match="em">
  <fo:sequence font-style="italic">
     <xsl:apply-templates/>
  </fo:sequence>
</xsl:template>
```

- `block` – Ein block-Formatobjekt setzt einfach einen Zeilenumbruch vor und nach einem Block (wie etwa einem Absatz). Das Gegenteil des block-Formatobjekts ist das inline-Formatobjekt. Listing 20.6 beispielsweise zeigt die Schablone für das block-Formatobjekt und Abbildung 20.3 das Ergebnis der Anwendung auf zwei p-Elemente.

Beachten Sie, daß ich bei der Beschreibung des XSL-Codes in den Listings 20.6 bis 20.9 `xsl:apply-templates` verwende, in den praktischen Beispielen aber `xsl:process-children`, weil das neuere `xsl_apply-templates` noch nicht von vielen Paketen unterstützt wird.

Listing 20.6: Eine Schablone für ein Block-Formatobjekt

```
1:  <xsl:template match="p">
2:    <fo:block
3:      indent-start="1.5in"
4:      indent-end="1.5in"
5:      space-before-optimum="6pt"
6:      space-after-optimum="6pt">
7:      <xsl:process-children/>
8:    </fo:block>
9:  </xsl:template>
```

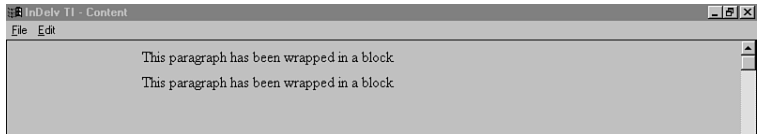


Abbildung 20.3: Die Ausgabe eines Block-Formatobjekts.

- `inline-box` – Dient als Inline-Container, mit dessen Hilfe Sie Text oder Grafik hervorheben, Rahmen und Hintergründe erzeugen oder einfach nur den Abstand um ein Element oder einen Textabschnitt festlegen können. Listing 20.7 zeigt die Schablone für ein `inline-box`-Formatobjekt, Abbildung 20.4 das Ergebnis der Anwendung auf ein `p`-Element.

Listing 20.7: Eine Schablone für ein `inline-box`-Formatobjekt

```
1:  <xsl:template match="em">
2:    <fo:inline-box
3:      background-color='yellow'
4:      graphic-line-thickness='0pt'>
5:      <xsl:process-children/>
6:    </fo:inline-box>
7:  </xsl:template>
```

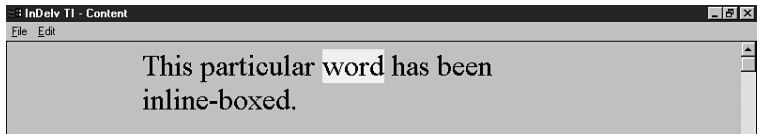


Abbildung 20.4: Ausgabe des `inline-box`-Formatobjekts.

- `list` – Ein `list`-Formatobjekt dient als Container für die Formatobjekte `list-item`, `list-item-label` und `list-item-body`. Wenn Sie die Liste verschachteln möchten, muß die zweite Liste ein Kind des Formatobjekts `list-item-body` sein.
- `rule-graphic` – Ein `rule-graphic`-Formatobjekt entspricht dem HR-Element (horizontal Rule, horizontale Linie) in HTML, kann aber Inline- oder Block-Formatobjekt sein. Listing 20.8 zeigt die Schablone für ein Blockformatobjekt, Abbildung 20.5 die Darstellung der horizontalen Linie.

Listing 20.8: Eine Schablone für ein `rule-graphic`-Formatobjekt

```
1:  <xsl:template match="hr">
2:    <fo:rule-graphic
3:      space-before-optimum="12pt"
4:      space-after-optimum="12pt"
5:      graphic-line-thickness='3pt'>
6:      <xsl:process-children/>
7:    </fo:rule-graphic>
8:  </xsl:template>
```

- graphic – Ein graphic-Formatobjekt entspricht dem IMG-Element in HTML, kann aber Inline- oder Block-Formatobjekt sein, beispielsweise:

```
<xsl:template match="display-graphic">
  <fo:graphic
     graphic-max-width="2in"
     graphic-max-height="2in"/>
</xsl:template>
```

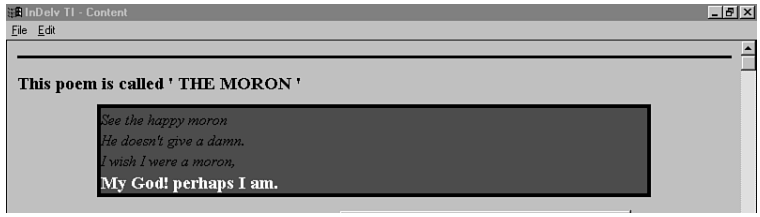


Abbildung 20.5: Darstellung des rule-graphic-Formatobjekts.

- `score` – Ein `score`-Flußbojekt ist ein Formatobjekt, das Textauszeichnungen vornimmt, beispielsweise Unterstreichen, Druchstreichen, Überstreichen usw. Ein `score` ist ein Inline-Formatobjekt.
- `block-level-box` – Eine `box` stellt Rahmen, Ränder, Hintergründe und Abstände zwischen dem Rahmen und seinem Inhalt bereit. Eine `box` kann ein `Inline`- oder Block-Formatobjekt sein. Listing 20.9 zeigt die Schablone für ein Block-Formatobjekt, Abbildung 20.6 das Ergebnis der Anwendung auf zwei `p`-Elemente.

Listing 20.9: Eine Schablone für ein `block-level-box`-Formatelement

```
1:  <xsl:template match="p">
2:    <fo:block-level-box
3:      indent-start="1in"
4:      indent-end="1in"
5:      background-color='blue'
6:      graphic-line-thickness='3pt'>
7:      <xsl:process-children/>
8:    </fo:block-level-box>
9:  </xsl:template>
```

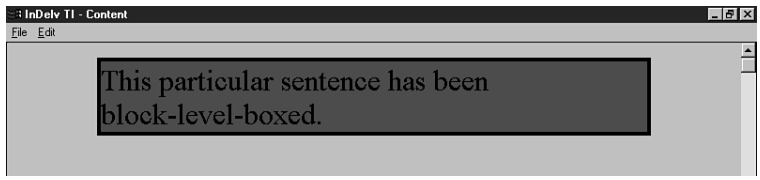


Abbildung 20.6: Darstellung des block-level-box-Formatobjekts.

- `page-numer` – Ein `page-number`-Formatobjekt weist die Rendering-Engine an, eine Seitennummer anzulegen und auszugeben.
- `link` – Ein `link` legt einen Bereich an, in dem Sie `link-end-locator`-Flußobjekte plazieren können.
- `link-end-locator` – `link-end-locator`-Flußobjekte bieten Informationen über das Ziel eines Links (äquivalent zu HREF="[URI]« in HTML).
- `character` – Erlaubt die Behandlung eines einzelnen Zeichens als Flußobjekt. Jedes Formatobjekt kann bestimmte Stileigenschaften annehmen.

Verarbeitung

Wie Sie gesehen haben, wird im ersten Teil der Aktion die Ergebnisdarstellung beschrieben. Im zweiten Teil geht es um die Verarbeitung oder den Aufbau eines Ergebnisbaum-Formatobjekts aus dem Quellbaum.

Die Standardanweisung für den XSL-Prozessor sieht wie folgt aus:

```
<xsl:apply-templates/>
```

Der XML-Prozessor verarbeitet alle Kinder des entsprechenden Quellelements.

Direkte Verarbeitung

Wenn Ihr Quell-XML-Dokument eine bekannte, regelmäßige Struktur aufweist (wie es normalerweise für Tabellen und Kataloge der Fall ist), können Sie den Baum mit Hilfe der `for-each`-Anweisung rekursiv verarbeiten. Listing 20.10 zeigt eine Beispiel-XML-Datei aus einer Datenbank mit einer Auflistung der Spurnamen einer CD-Sammlung.

Listing 20.10: Eine XML-Datei mit regelmäßiger Struktur

```
1:  <?xml version="1.0"?>
2:  <?xml:stylesheet type="text/xsl" href="cd1.xsl"?>
3:  <CDs>
4:    <CD>
5:      <title>Boys for Pele</title>
6:      <artist>Tori Amos</artist>
7:      <tracks>
8:        <track>Horses</track>
9:        <track>Blood Roses</track>
10:       <track>Father Lucifer</track>
11:       <track>Professional Widow</track>
12:       <track>Mr. Zebra</track>
13:       <track>Marianne</track>
14:       <track>Caught a Lite Sneeze</track>
15:       <track>Muhammad My Friend</track>
16:       <track>Hey Jupiter</track>
17:       <track>Way Down</track>
18:       <track>Little Amsterdam</track>
19:       <track>Talula</track>
20:       <track>Not the Red Baron</track>
21:       <track>Agent Orange</track>
22:       <track>Doughnut Song</track>
23:       <track>In the Springtime of his Voodoo</track>
24:       <track>Putting the Damage on</track>
```

```
25:       <track>Twinkle</track>
26:     </tracks>
27:   </CD>
28:   <CD>
29:     <title>The Ghosts that Haunt Me</title>
30:     <artist>Crash Test Dummies</artist>
31:     <tracks>
32:       <track>Winter Song</track>
33:       <track>Comin' Back Soon (The Bereft Man's Song)</track>
34:       <track>Superman's Song</track>
35:       <track>The Country Life</track>
36:       <track>Here on Earth (I'll have my Cake)</track>
37:       <track>The Ghosts that Haunt Me</track>
38:       <track>Thick-Necked Man</track>
39:       <track>Androgynous</track>
40:       <track>The Voyage</track>
41:       <track>At My Funeral</track>
42:     </tracks>
43:   </CD>
44: </CDs>
```

Das in Listing 20.11 gezeigte XSL-Stylesheet erzeugt ein HTML-Dokument mit einer Tabelle, die für jedes `track`-Element eine Zeile bereitstellt.

Listing 20.11: Mit `for-each` wird die XML-Datei rekursiv verarbeitet

```
1:  <?xml version="1.0"?>
2:  <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
3:
4:  <xsl:template match="/">
5:    <xsl:apply-templates/>
6:  </xsl:template>
7:
8:  <xsl:template match="track">
9:     <xsl:apply-templates/>, </xsl:template>
10:
11: <xsl:template match="track[end()]">
12:   <xsl:apply-templates/>. </xsl:template>
13:
14: <xsl:template match="textnode()">
15:   <xsl:get-value/></xsl:template>
16:
17: <xsl:template match="/">
18:   <HTML>
19:     <HEAD>
20:       <TITLE>My CD Collection</TITLE>
```

```
21:     </HEAD>
22:     <BODY>
23:       <H1>My CD Collection</H1>
24:       <TABLE BORDER="3" CELLSPACING="2" CELLPADDING="6">
25:          <col bgcolor="yellow"/>
26:         <THEAD align="left" bgcolor="silver">
27:            <TH>Artist</TH><TH>Album Title</TH><TH>Tracks</TH>
28:         </THEAD>
29:         <TBODY>
30:           <xsl:for-each select="CDs/CD">
31:           <TR>
32:              <TD><font color="red" size="5">
33:                  <B><xsl:value-of select="artist"/></B></font></TD>
34:              <TD><B><I><xsl:value-of select="title"/></I></B></TD>
35:              <TD><xsl:apply-templates select="tracks/track"/></TD>
36:           </TR>
37:           </xsl:for-each>
38:         </TBODY>
39:       </TABLE>
40:     </BODY>
41:   </HTML>
42: </xsl:template>
43:
44: </xsl:stylesheet>
```

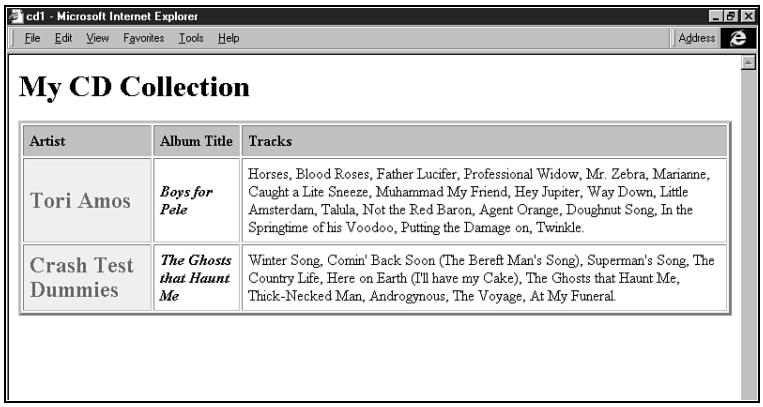


Abbildung 20.7: Der XML-Code, dargestellt als HTML in IE5.

Listing 20.11 verwendet in Zeile 11 die Anweisung `end()`. Das ist leider eine der fehlerhaften Änderungen, die Microsoft in IE5 implementiert hat, bevor die neueste Version des XSL-Syntaxentwurfs veröffentlicht wurde. Gemäß dem formalen Vorschlag sollte hier `last-of-type()` verwendet werden, aber IE5 erkennt diesen Bezeichner nicht.

Wie überraschend einfach es doch ist, XML-Code mit den XSL-Stylesheets in HTML umzuwandeln! Sie brauchen nur die gewünschten Elemente auszuwählen, das HTML-Start-Tag einzufügen, den Elementinhalt zu verarbeiten und dann das HTML-Ende-Tag einzufügen. Das ist wahrscheinlich die einfachste Methode, die Sie im gesamten Buch kennengelernt haben – ein Grund dafür, XSL zu lernen!

Die Zuordnung von XML-Elementen zu HTML-Elementen mit Hilfe von XSL ist eine einfache Methode, XML darzustellen, aber es gibt noch eine einfachere Möglichkeit, `xsl:copy`, das später in diesem Kapitel erklärt wird. Die Anweisung `xsl:copy` erlaubt Ihnen, HTML-Markup in einer XML-Datei zu verwenden und diese einfach dem Web-Browser zu übergeben.

Beschränkte Verarbeitung

Wenn Sie nur bestimmte Kindelemente eines Elements verarbeiten wollen, verwenden Sie das Element `xsl:apply-templates` und wählen die gewünschten Elemente dem Namen nach aus. Das folgende Beispiel verarbeitet alle `title`-Elemente, um daraus ein Inhaltsverzeichnis zu erstellen:

```
<xsl:template match="section">
 <fo:block font-size="12pt">
   <xsl:apply-templates select="title"/>
 </fo:block>
</xsl:template>
```

Hier kann ebenfalls die Mustersuche angewandt werden, um die Auswahl noch genauer einzugrenzen.

Bedingte Verarbeitung

Es gibt zwei Anweisungen in XLS, die Ihnen erlauben, ein Element gemäß bestimmter Bedingungen zu verarbeiten: `xsl:if` und `xsl:choose`.

Die Anweisung `xsl:if` realisiert die einfache Bedingung `if` (a) `then` (b). Die Anweisung `xsl:choose` erlaubt eine Auswahl aus mehreren Optionen.

Das Element xsl:if hat ein test-Attribut, das ein Muster vorgibt. Der Inhalt ist eine Schablone. Wenn das Muster eine nicht leere Elementliste erzeugt, wird der Inhalt instantiiert, andernfalls wird nichts angelegt.

Im folgenden Beispiel werden die Namen in einer Namensgruppe als Liste formatiert, in der die einzelnen Werte durch Kommas voneinander abgetrennt sind:

```
<xsl:template match="namelist/name">
  <xsl:apply-templates/>
  <xsl:if test=".[not(last-of-type())]">, </xsl:if>
</xsl:template>
```

Das Element xsl:choose wählt eine von mehreren Alternativen aus. Sie besteht aus mehreren xsl:when-Elementen, gefolgt von einem optionalen xsl:otherwise-Element.

Jedes xsl:when-Element hat ein test-Attribut, das ein Muster angibt, auf das hin verglichen werden soll. Das Vergleichsergebnis ist true, wenn das Muster eine nichtleere Elementliste erzeugt.

Der Inhalt der Elemente xsl:when und xsl:otherwise ist eine Schablone. Wenn ein xsl:choose-Element verarbeitet wird, wird jedes der xsl:when-Elemente ausgewertet. Der Inhalt des ersten, und nur des ersten, xsl:when-Elements, dessen Auswertung true ergibt, wird verwendet. Ist kein xsl:when-Element true, wird der Inhalt des xsl:otherwise-Elements verwendet. Ist kein xsl:when-Element true und kein xsl:otherwise-Element vorhanden, wird nichts erzeugt. Sie sollten sich also angewöhnen, immer ein xsl:otherwise-Element bereitzustellen.

Das XSL-Stylesheet, das Sie in Listing 20.12 sehen, stellt die Einträge in einer sortierten Liste unter Verwendung von arabischen Ziffern, Buchstaben und römischen Ziffern dar, abhängig von der Verschachtelungstiefe der Einträge.

Listing 20.12: Bedingte Verarbeitung in einer XSL-Datei

```
1: <xsl:template match="list/item">
 2:   <fo:list-item indent-start='12pt'>
 3:     <fo:list-item-label>
 4:       <xsl:choose>
 5:
 6:         <xsl:when test='ancestor(list/list)'>
 7:           <xsl:number format="i"/>
 8:         </xsl:when>
 9:
10:         <xsl:when test='ancestor(list)'>
11:           <xsl:number format="a"/>
12:         </xsl:when>
13:
```

```
14:         <xsl:otherwise>
15:           <xsl:number format="1"/>
16:         </xsl:otherwise>
17:
18:       </xsl:choose>
19:
20:       <xsl:text>. </xsl:text>
21:     </fo:list-item-label>
22:
23:     <fo:list-item-body>
24:       <xsl:apply-templates/>
25:     </fo:list-item-body>
26:
27:   </fo:list-item>
28:
29: </xsl:template>
```

Berechnung des Ausgabetexts

Innerhalb einer Schablone kann das `xsl:value-of`-Element genutzt werden, um den Ausgabetext zu »berechnen«, beispielsweise durch Extrahieren von Text aus dem Quellbaum oder durch Einfügen des Werts einer String-Konstanten.

Das `xsl:value-of`-Element berechnet den Text unter Verwendung eines String-Ausdrucks, der als Wert des `select`-Attributs angegeben wird. String-Ausdrücke können auch innerhalb von Attributwerten literaler Ergebniselemente verwendet werden, wenn sie in geschweifte Klammern ({}) eingeschlossen werden.

Angenommen, Sie haben den folgenden XML-Codeabschnitt:

```
<link target="http://my.home.com/~simon">My Diary</link>
```

Sie wollen das Attribut in ein Element umwandeln:

```
My Diary <address>http://my.home.com/~simon</address>
```

Listing 20.13 zeigt den XSL-Code, der das bewerkstelligt.

Listing 20.13: Die Verwendung von `xsl:value-of`

```
1:     <xsl:template match="link">
2:       <xsl:apply-templates/>
3:         <address>
4:           <xsl:value-of select="link/@target"/>
5:         </address>
6:     </xsl:template>
```

Sie können auch in der anderen Richtung vorgehen und ein Element in ein Attribut umwandeln. Angenommen, Sie haben den folgenden XML-Code:

```
<image>
  <file>my-smiling-face.gif</file>
  <size width="100"/>
</image>
```

Sie wollen ihn in folgendes umwandeln:

```
<graphic src="my-smiling-face.gif" width="100"/>
```

Listing 20.14 zeigt, wie das realisiert werden kann.

Listing 20.14: Verwendung einer Attributwertschablone

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template> <xsl:element name="graphic">
<xsl:attribute name="src">
<xsl:copy>
<xsl:value-of select="image/file"/>
</xsl:copy>
</xsl:attribute>
<xsl:attribute name="width">
<xsl:copy>
<xsl:value-of select="image/size/@width"/>
</xsl:copy>
</xsl:attribute>
</xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Textformatobjekte werden wie in Listing 20.15 gezeigt in das Verarbeitungsmodell eingefügt.

Listing 20.15: Die Verwendung von `xsl-text` zum Einfügen von Text in die Ausgabe-XML-Datei

```
1:    <xsl:template match="title">
2:        <fo:block
3:            font-size="14pt"
4:            font-weight="bold"
5:            space-before-optimum="12pt"
6:            space-after-optimum="8pt">This poem is called '
7:            <xsl:process-children/>
```

```
8:            <xsl:text> '</xsl:text>
9:        </fo:block>
10:   </xsl:template>
```

Numerierung

Die Elemente einer XML-Eingabedatei werden mit Hilfe des `xsl:number`-Elements automatisch numeriert.

Sie steuern die Numerierung mit Hilfe der Attribute `level`, `count` und `from`.

Beispielsweise numeriert die folgende XSL-Schablone die `item`-Elemente in einer Liste:

```
<xsl:template match="list/item">
            <fo:block>
                 <xsl:number/>
                 <xsl:text>. </xsl:text>
                 <xsl:process-children/>
            </fo:block>
         <xsl:template>
```

Listing 20.16 zeigt ein praktisches Beispiel für die Numerierung, wobei zwei Regeln für die Numerierung von `title`-Elementen angewendet werden. Diese Schablone ist für ein Dokument vorgesehen, das mehrere Kapitel gefolgt von mehreren Anhängen enthält, wobei sowohl Kapitel als auch Anhänge Abschnitte mit weiteren Unterabschnitten enthalten. Die Kapitel werden als 1, 2, 3 numeriert. Die Anhänge werden als A, B, C numeriert. Die Abschnitte werden als 1.1, 1.2, 1.3 numeriert und die Abschnitte in den Anhängen als A.1, A.2, A.3.

Listing 20.16: *`xsl:number`* ***zum Alegen einer mehrstufigen Numerierung***

```
1:  <xsl:template match="title">
2:    <fo:block>
3:       <xsl:number level="multi"
4:            count="chapter ¦ section ¦ subsection"
5:            format="1.1. "/>
6:       <xsl:apply-templates/>
7:    </fo:block>
8:  </xsl:template>
9:
10: <xsl:template match="appendix//title">
11:   <fo:block>
12:      <xsl:number level="multi"
13:         count="appendix ¦ section ¦ subsection"
```

```
14:         format="A.1. "/>
15:       <xsl:apply-templates/>
16:    </fo:block>
17: </xsl:template>
```

Sortieren

Das Sortieren wird durch `xsl:sort`-Elemente als Kinder `von xsl:apply-templates` oder `xsl:for-each`-Elemente festgelegt. Das erste `xsl:sort`-Kind gibt den primären Sortierschlüssel an, das zweite `xsl:sort`-Kind gibt den sekundären Sortierschlüssel an usw.

Wenn ein `xsl:apply-templates`- oder `xsl:for-each`-Element mehrere `xsl:sort`-Kinder enthält, werden die ausgewählten Elemente nicht in der Reihenfolge verarbeitet, wie sie im XML-Dokument erscheinen, sondern gemäß der durch die angegebenen Sortierschlüssel festgelegten Reihenfolge.

Ein `xsl:sort`-Element hat ein `select`-Attribut, dessen Wert ein Suchmuster darstellt. Für jedes zu verarbeitende Element wird das Suchmuster ausgewertet. Der Wert des ersten ausgewählten Elements wird als Sortierschlüssel für dieses Element verwendet. Der Standardwert des `select`-Attributs ist ».« (das aktuelle Element).

Angenommen, Sie haben eine CD-Datenbank mit XML-Auszeichnungen:

```
<CDs>
         <CD>
           <artist>Tori Amos</artist>
           <title>Boys for Pele</title>
         </CD>
</CDs>
```

Jetzt können Sie eine Liste der CDs erstellen, sortiert nach dem Künstler. Den entsprechenden XSL-Code sehen Sie in Listing 20.17.

Listing 20.17: XSL zum Sortieren von Elementen

```
1:  <xsl:template match="CDs">
2:    <ul>
3:      <xsl:apply-templates select="CD">
4:        <xsl:sort select="artist"/>
5:        <xsl:sort select="title"/>
6:      </xsl:apply-templates>
7:    </ul>
8:  </xsl:template>
9:
10: <xsl:template match="CD">
```

```
11:   <li>
12:     <xsl:value-of select="artist"/>
13:       <xsl:text> </xsl:text>
14:     <xsl:value-of select="title"/>
15:   </li>
16: </xsl:template>
```

Whitespace

Wenn der Whitespace in einem `xsl:text`-Element beibehalten werden muß, setzen Sie das `xml:space`-Attribut auf `preserve`:

```
<xsl:text xml:space="preserve">
       The spaces        in     this   t e x t
       will n o  t be       removed.
    </xsl:text>
```

Wenn Sie dagegen wissen, daß die Whitespaces in einem Element einfach nur überflüssig sind, können Sie sie automatisch entfernen, indem Sie `xml:space="strip"` deklarieren.

Um auf einer höheren Ebene alle Elemente eines bestimmten Typs zu berücksichtigen, können Sie auch separate Anweisungen in das XSL-Stylesheet aufnehmen:

- <xsl:strip-space element="para">
- <xsl:preserve-space element="code">

Makros

Makros erlauben die Wiederverwendung von Teilen Ihrer Stylesheets. Diese Stylesheets werden dazu in benannte Abschnitte zerlegt. Listing 20.18 zeigt ein einfaches Marko, das den Inhalt in einen Rahmen einfügt.

Listing 20.18: Ein einfaches XSL-Makro

```
1: <xsl:define-macro name="special-para">
2:   <fo:block-level-box
3:     space-before-optimum="8pt"
4:     space-after-optimum="8pt"
5:      graphic-line-thickness='1.5pt'>
6:      <xsl:contents/>
7:   </fo:block-level-box>
8: </xsl:define-macro>
```

Dieses Makro kann bei Bedarf aufgerufen werden, wie in Listing 20.19 gezeigt.

Listing 20.19: Aufruf eines XSL-Makros

```
1:  <xsl:template match="para">
2:    <xsl:invoke macro="special-para">
3:      <xsl:apply-templates/>
4:    </xsl:invoke>
5:  </xsl:template>
```

Formatobjekteigenschaften

Jedes Formatobjekt hat bestimmte Eigenschaften. Sie können hier nicht alle im Detail aufgeführt werden, und noch viel weniger ist es möglich, sie alle zu beschreiben. Die folgende Liste bietet einen knappen Überblick über einige der Formatobjekte mit ihren Eigenschaften und Werten. Wird nur ein Wert angegeben, ist dies der Standardwert, andernfalls wird der Standardwert fett dargestellt. Sie kennen einige dieser Formatobjekte bereits aus den Beispielen in diesem Kapitel.

Formatobjekte sind momentan die am wenigsten unterstützte Funktion von XSL; aus diesem Grund werden sie sich sehr wahrscheinlich noch ändern (deshalb werden sie vermutlich so wenig unterstützt). Die folgende Liste ist keinesfalls vollständig oder endgültig. Sie soll Ihnen nur einen Eindruck von der Vielfalt vermitteln, die XSL zu bieten hat. Den aktuellen Stand in Hinblick auf die Formatobjekte und ihre Eigenschaften finden Sie unter `http://www.w3c.org/WD/WD-xsl`).

- Simple-page-master:

Ein `simple-page-master` definiert das Layout einer Seite. `simple-page-master` werden mit Hilfe einer `sequence`-Angabe wiederholt.

```
master-name = name-specifier
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
page-height = length-specifier  ¦  auto
```

```
page-width = length-specifier ¦ auto
page-writing-mode = writing-mode-specifier  lr-tb
margin-bottom  =  36.0pt
margin-left =  36.0pt
margin-right  = 36.0pt
margin-top = 36.0pt
body-overflow  = ( visible ¦ hidden ¦ scroll ¦ auto )
body-writing-mode = a writing-mode-specifier ¦ use-page-writing-mode
end-side-overflow = ( visible ¦ hidden ¦ scroll ¦ auto )
end-side-separation (0.0pt
end-side-size 0.0pt
end-side-writing-mode = a writing-mode-specifier ¦ use-page-writing-mode
footer-overflow = ( visible ¦ hidden ¦ scroll ¦ auto )
footer-precedence = ( true ¦ false )
footer-separation 18.0pt
footer-size 36.0pt
footer-writing-mode = a writing-mode-specifier ¦ use-page-writing-mode
header-overflow = ( visible ¦ hidden ¦ scroll ¦ auto )
header-precedence = ( true ¦ false )
header-separation 18.0pt
header-size  36.0pt
header-writing-mode = a writing-mode-specifier ¦ use-page-writing-➥mode
start-side-separation 0.0pt
start-side-size 0.0pt
start-side-overflow = ( visible ¦ hidden ¦ scroll ¦ auto )
start-side-writing-mode = a writing-mode-specifier
      ¦ use-page-writing-mode
```

- Block:

Ein block-Formatobjekt erzeugt einen Bereich auf Blockebene, der Textzeilen enthält. Blöcke werden normalerweise für die Formatierung von Absätzen, Titeln, Überschriften, Abbildungs- und Tabellenunterschriften usw. verwendet. In der Regel wird damit ein rechteckiger Bereich festgelegt, der die Breite des aktuellen Bereichs hat und dessen Höhe durch den enthaltenen Text bestimmt wird.

```
language = ( none ¦ use-document ¦ an xml:lang specifier )
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
font-family = font-name or font-name-list any
font-style = ( normal ¦ italic ¦ oblique )
```

```
font-stretch = ( ultra-condensed ¦ extra-condensed ¦ condensed ¦
            semi-condensed ¦ normal ¦ semi-expanded ¦ expanded ¦
            extra-expanded ¦ ultra-expanded)
font-size 10.0pt
font-variant = ( normal ¦ small-caps )
font-weight = ( any ¦ not-applicable ¦ ultra-light ¦ extra-light ¦
              light ¦ semi-light ¦ book ¦ normal ¦ medium ¦
➥semi-bold ¦
              bold ¦ extra-bold ¦ ultra-bold  )
glyph-alignment-mode  = ( base ¦ center ¦ top ¦ bottom ¦ font )
indent-end 0.0pt
indent-start 0.0pt
indent-first-line-start 0.0pt
break-after = ( none ¦ page ¦ page-odd ¦ page-even )
break-before = ( none ¦ page ¦ page-odd ¦ page-even )
keep = ( auto ¦ no-break ¦ page )
orphans = 2
widows = 2
keep-with-next = ( true ¦ false )
keep-with-previous = ( true ¦ false )
block-line-breaking = ( wrap ¦ asis ¦ as-is-wrap ¦ asis-truncate ¦ none )
block-asis-truncate-indicator = ( none ¦ a character )
block-asis-wrap-indicator = ( none ¦ a character )
block-asis-wrap-indent 0.0pt
hyphenation-keep = ( none ¦ spread ¦ page ¦ column )
hyphenation-ladder-count  2
text-align = ( start ¦ end ¦ left ¦ right ¦ spread-inside ¦
              spread-outside ¦ page-inside ¦ page-outside ¦
              center ¦ justify ¦ justify-force )
text-align-last = ( auto ¦ start ¦ end ¦ left ¦ right ¦
                  spread-inside ¦ spread-outside ¦
                  page-inside ¦ page-outside ¦ center ¦ justify )
linespacing-strategy = ( fixed ¦ auto )
linespacing 12.0pt
space-after-maximum  0.0pt
space-after-minimum 0.0pt
space-after-optimum 0.0pt
space-before-maximum 0.0pt
space-before-minimum 0.0pt
space-before-optimum 0.0pt
writing-mode = writing-mode-specifier  lr-tb
```

- Character:

`character`-Formatobjekte werden verwendet, wenn explizit ein oder mehrere Zeichen durch ein bestimmtes Symbol überschrieben werden sollen.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
text-shadow = see the CSS proposal
text-transform = ( as-entered ¦ lower ¦ upper ¦ title ¦ (see CSS) )
char
char-kern
char-kern-mode
char-ligature
color
font-specification
glyph-alignment-mode
hyphenate
hyphenation-char
inhibit-wrap
language
position-point-shift
letterspace-after-maximum
letterspace-after-minimum
letterspace-after-optimum
wordspacing-maximum
wordspacing-minimum
wordspacing-optimum
text-shadow
text-transform: capitalize ¦ uppercase ¦ lowercase ¦ none
writing-mode
```

- List:

Erzeugt einen Bereich auf Blockebene, der eine Liste enthält. Eine Liste kann nur `list-item-label`-, `list-item`-Paare oder nur `list-item-body`-Objekte als Kindelemente enthalten.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
break-before = ( none ¦ page ¦ page-odd ¦ page-even )
```

```
break-after = ( none ¦ page ¦ page-odd ¦ page-even ¦),
indent-start 0.0pt
indent-end 0.0pt
space-before-maximum  0.0pt
space-before-minimum 0.0pt
space-before-optimum 0.0pt
space-after-maximum  0.0pt
space-after-minimum 0.0pt
space-after-optimum  0.0pt
```

- List-item:

Ein `list-item`-Flußobjekt enthält die Beschriftung und den Rumpf jedes Eintrags. Mit `list-item` werden Eigenschaften der Liste einzeln überschrieben oder geändert. Ein `list-item`-Flußobjekt kann nur in einer Liste enthalten sein. Außerdem stellt es die Hülle für ein `list-item-label` und einen `list-item-body` dar.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
indent-start 0.0pt
indent-end 0.0pt
item-space-before-maximum 0.0pt
item-space-before-minimum 0.0pt
item-space-before-optimum  0.0pt
item-space-after-maximum  0.0pt
item-space-after-minimum  0.0pt
item-space-after-optimum 0.0pt
```

- List-item-label:

Mit einem `list-item-label` wird der Rumpf eines Listeneintrags numeriert, gekennzeichnet oder hervorgehoben. Ein `list-item-label` kann nur in einer Liste enthalten sein. Es besitzt Inhalt und wird formatiert, um den Listeneintrag zu numerieren oder zu kennzeichnen.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
```

```
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
label-width 0.0pt
space-end 0.0pt
label-separator 12.0pt
```

- List-item-body:

Das `list-item-body`-Flußobjekt enthält die Komponenten (in der Regel Blöcke) für einen Listeneintrag. Es steuert die Standarddarstellungen für den Rumpf, den Abstand zwischen Zeilen und Absätzen sowie die Umbruchpriorität für Zeilen und Absätze innerhalb des Listeneintrags.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
```

- Rule-graphic:

Eine grafische Darstellung eines Liniensgements.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
color
block-level-alignment
break-after
break-before
graphic-line-thickness
indent-end
indent-start
inhibit-wrap
keep
keep-with-previous
keep-with-next
rule-graphic-length
rule-graphic-orientation
position-point-shift
space-after-maximum
```

```
space-after-minimum
space-after-optimum
space-before-maximum
space-before-minimum
space-before-optimum
writing-mode
```

- Display-graphic:

Mit diesem Objekt wird ein Bereich auf Blockebene angelegt, der eine Grafik aufnehmen kann.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
inline
block-level-alignment
break-after
break-before
color
external-graphic-id
graphic-max-height
graphic-max-width
graphic-scale
indent-end
indent-start
inhibit-wrap
keep
keep-with-previous
keep-with-next
position-point-x
position-point-y
position-preference
space-after-maximum
space-after-minimum
space-after-optimum
space-before-maximum
space-before-minimum
space-before-optimum
writing-mode
```

- Inline-box:

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
box-reserve-space
box-open-end
box-size-after
box-size-before
box-type
break-after
break-before
color
graphic-line-thickness
indent-end
indent-start
inhibit-textline-breaks?
keep
keep-with-previous?
keep-with-next?
space-after-maximum
space-after-minimum
space-after-optimum
space-before-maximum
space-before-minimum
space-before-optimum
writing-mode
```

- Block-level-box:

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
box-reserve-space
box-size-after
box-size-before
box-type
break-after
break-before
```

```
color
graphic-line-thickness
indent-end
indent-start
inhibit-textline-breaks?
keep
keep-with-previous?
keep-with-next?
space-after-maximum
space-after-minimum
space-after-optimum
space-before-maximum
space-before-minimum
space-before-optimum
writing-mode
```

- Link:

Ein Link kann ein Anzeige-Link (Blockebene) oder ein Inline-Link sein.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
merge-link-end-locators = ( true ¦ false )
```

- Link-end-locator:

Steht für das Ziel eines Links.

```
background-attachment = ( scroll ¦ fixed )
background-color = a color-specifier or transparent
background-image = a URI or none
background-position-x = ( a length-specifier ¦ left ¦ center ¦
➥right )
background-position-y = ( a length-specifier ¦ top ¦ middle ¦
➥bottom )
background-repeat = ( no-repeat ¦ repeat ¦ repeat-x ¦ repeat-y )
href = ( XPointer )
show-content = ( true ¦ false )
```

Vermeiden von Flußobjekten

In einem früheren Beispiel haben Sie gesehen, wie XML-Code mit Hilfe eines XSL-Stylesheet in HTML-Code umgewandelt wird. Eine möglicherweise nicht korrekte logische Erweiterung dieser Funktion (nicht korrekt, weil sie eigentlich dem Wesen von XSL widerspricht) ist die Verwendung von XSL-Stylesheets, um HTML-Markup mit »echtem« XML-Markup zu vermischen. Dadurch entsteht eine Hybridform aus XML/HTML, ähnlich der in früheren Kapiteln beschriebenen.

Listing 20.20 zeigt ein einfaches XML-Dokument, das bekannte HTML-Tags zur Auszeichnung einer Tabelle verwendet. Schließlich werden Tabellen in HTML zufriedenstellend dargestellt und bieten eine einfache Methode, die Tabellenformatierung in XSL anzuwenden.

Listing 20.20: Einbetten von HTML-Markup in eine XML-Datei

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="page.xsl"?>
<PAGE>
  <PARA>
    These are the "new" CDs I found in CD-Warehouse.
  </PARA>
  <PARA>
    <TABLE>
      <TITLE>List of New CDs</TITLE>
      <TR valign="top">
        <TH>Artist</TH><TH>Album Title</TH>
      </TR>
      <TR valign="top">
        <TD>Anouk</TD>
        <TD>Together Alone</TD>
      </TR>
      <TR valign="top">
        <TD>Tori Amos</TD>
        <TD>Boys for Pele</TD>
      </TR>
      <TR valign="top">
        <TD>Crash Test Dummies</TD>
        <TD>The Ghosts that Haunt Me</TD>
      </TR>
      <TR valign="top">
        <TD>Crash Test Dummies</TD>
        <TD>A Worm's Life</TD>
      </TR>
    </TABLE>
  </PARA>
</PAGE>
```

Wenn ich jetzt ein XSL-Stylesheet entwickle, das dieses XML im Internet Explorer 5 (Beta Preview 2) darstellt, kann ich diesen XML-Tabellencode mit `xsl:copy` an den Browser weitergeben. Der Browser verwendet seine Standard-HTML-Darstellung, und ich erspare mir eine Menge Arbeit. Listing 20.21 zeigt das XSL-Stylesheet. In Abbildung 20.8 sehen Sie das Ergebnis im Internet Explorer 5.

Listing 20.21: Unveränderte Weitergabe von HTML mittels eines XSL-Stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template><xsl:value-of/></xsl:template>

<xsl:template match="/">
  <HTML>
    <BODY BGCOLOR="#FFFFFF" LINK="#000066"
          VLINK="#666666" TEXT="#000000">
      <xsl:for-each select="PAGE">
        <xsl:apply-templates/>
      </xsl:for-each>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match="PAGE/PARA">
  <P><xsl:apply-templates/></P>
</xsl:template>

<xsl:template match="TABLE">
  <TABLE border="1">
    <xsl:apply-templates>
      <xsl:template>
        <xsl:copy>
          <xsl:for-each select="@*">
            <xsl:attribute>
              <xsl:value-of/>
            </xsl:attribute>
          </xsl:for-each>
          <xsl:apply-templates/>
        </xsl:copy>
      </xsl:template>
    </xsl:apply-templates>
  </TABLE>
</xsl:template>

</xsl:stylesheet>
```

Abbildung 20.8: Der XML/HTML-Tabellencode als HTML im Internet Explorer 5.

Es gibt mehrere Dinge in Listing 20.22, die angesprochen werden sollten. Ein Großteil der Arbeit wird im `xsl:copy`-Element erledigt (Zeilen 24 bis 31). Beachten Sie jedoch, daß ich ein `xsl:for-each`-Element verwendet habe (Zeilen 26 bis 30), um die Attribute der einzelnen Elemente im `TABLE`-Element zu durchlaufen. Mit `select="@*"` wähle ich die Attribute (beliebigen Typs) einzeln aus und verwende das `xsl:attribute`-Element (Zeilen 27 bis 29), um jedes Attribut und seinen Wert von dem XML-Element in das Eingabedokument zu kopieren, das im Internet Explorer angezeigt wird.

Zusammenfassung

Dieses Kapitel hat Ihnen kurz XSL1 vorgestellt, bevor wir zu der neuesten XSL-Version gekommen sind. Sie haben die Struktur eines XSL-Stylesheet und ihre Schablonen kennengelernt, und Sie haben erfahren, wie man nicht nur Elemente auswählt, sondern auch komplexe Regeln für eine bedingte Auswahl einführt. Im XSL-Entwurf gibt es zahlreiche weitere Beispiele, die interessante Einblicke bieten. Schließlich haben Sie die Formatobjekte von XSL kennengelernt, ebenso wie praktische Beispiele, wie sie zur Darstellung Ihres XML-Codes genutzt werden können.

Der Beta Preview 2 Release des Microsoft Internet Explorer 5 unterstützt einen Großteil der Transformationskomponente von XSL, aber keine Flußobjekte. Mit ein bißchen Umsicht in Hinblick auf Abweichungen, die der Internet Explorer vom Standard aufweist, und gerüstet mit den Informationen aus diesem Kapitel, können Sie jedoch anfangen, Ihre Fertigkeiten zu vertiefen.

F&A

F Was passiert, wenn ein Element mehreren Stylesheet-Regeln entspricht?

A *Alle Regeln werden ausgewertet, und nur die spezifischste davon wird verwendet. Beispielsweise hat* `select="a/b"` *Priorität vor* `select="a"`*. Sie steuern diese Verhaltensweise durch die Bereitstellung eines Prioritätsattributs für eine Regel.*

F Was passiert, wenn Sie für ein Element keine Regel angeben?

A *Die Standardverarbeitung wird angewendet. Die (implizite) Standardregel lautet für jedes Element im Element:* `<xsl:apply-templates>`*.*

F Wie können Sie Elemente in der XML-Quelldatei überspringen?

A *Es gibt zwei Methoden: Sie geben keine Regel für das Element an, oder Sie wählen das Element aus und geben explizit einen leeren String dafür aus (das letztere ist die Methode der Wahl, weil das Element-Markup beibehalten wird, so daß Sie prüfen können, ob Sie das Element wirklich verarbeitet haben).*

Übungen

1. Sie haben gesehen, wie man ein bestimmtes Element zur Verarbeitung auswählt. Schreiben Sie für den folgenden XML-Code die XSL-Schablone zur Umkehrung der Reihenfolge und Ausgabe des Strings »cccbbbaaa« (verwenden Sie dazu ein `sequence`-Formatobjekt):

```
<list>
  <itema>aaa</itema>
  <itemb>bbb</itemb>
  <itemc>ccc</itemc>
</list>
```

2. Schreiben Sie für den Code aus Übung 1 eine XSL-Schablone, die das `itemb`-Element überspringt und den String »aaaccc« ausgibt.

3. Schreiben Sie für die folgende XML-Datei ein XSL-Stylesheet, das sie in ein HTML-Dokument umwandelt:

```
<?xml version="1.0"?>
<!DOCTYPE page [
    <!ELEMENT page (#PCDATA, page)*>
    <!ELEMENT para (#PCDATA, image)*>
    <!ELEMENT image (#PCDATA)>
]>
```

```
            <page>
               <title>My Home Page</title>
               <para>This is my first basic home page
created in XML.</para>
               <para>It even includes an image:
<image>logo.gif</image>.</para>
          </page>
```

XML-Anwendungen in der Praxis

Woche 3

Ich habe in diesem Buch immer versucht, alle Informationen über XML anhand praktischer Beispiele zu erklären, was nicht immer ganz einfach war. In diesem Kapitel werde ich diese Vorgehensweise beibehalten. Ich werde einige der exotischeren Anwendungen ansprechen, mich aber auf die XML-Anwendungen konzentrieren, die bereits funktionieren. In einigen Fällen ist die Implementierung dieser Anwendungen jedoch nicht ganz perfekt.

In diesem Kapitel geht es um die folgenden Dinge:

- Die wichtigsten XML-Anwendungen
- Die Grundlagen von MathML (*Mathematics Markup Language*) und die Anzeige Ihres Codes
- Strukturierte Grafik und neue Entwicklungen
- Verhalten und die von Microsoft geplante CSS-Erweiterung
- Chrome und die beeindruckende interaktive 2D- und 3D-XML-Browsing-Umgebung von Microsoft

Der aktuelle Stand

Die Liste der möglichen Einsatzbereiche für XML ist endlos, und es gibt ständig neue Anwendungen. Die folgende Auflistung zeigt einige Beispiele, wobei die Reihenfolge nicht von ihrer Bedeutung abhängig ist. Sie ist keinesfalls vollständig, aber einige der wichtigsten oder vielversprechendsten Anwendungen werden aufgeführt:

- Die Software-Distribution und der Update der OSD (*Open Software Description*) – Microsofts Bemühungen laufen darauf hinaus, Software anzubieten, die nicht mehr installiert werden muß. Die Software wird aus dem Internet heruntergeladen, genutzt und dann wieder verworfen.

  Wenn Ihr Web-Browser ein Java-Applet herunterlädt oder JavaScript-Code ausführt, dann haben Sie letztlich Software heruntergeladen, die verworfen wird, sobald Sie den Browser schließen. Haben Sie davon schon gehört? Durch die Erweiterung von CSS (wie Sie in einem Beispiel später in diesem Kapitel noch sehen werden) wollen Microsoft und Marimba Software (sogar Programmcode) über das Internet weitergeben.

- Das Internet-OTP (*Open Trading Protocol*) – Ein XML-Protokoll für den Austausch von Informationen über Finanztransaktionen, elektronische Zahlungsweise, Kreditkarteninformation und Bankkonteninformationen.

- JSML (*Java Speech Markup Language*) – Fügt der künstlich erzeugten Sprache strukturelle Informationen hinzu, so daß sie natürlicher klingt.

- Die HL7-Initiative (*Health Level 7*) für die Formatierung elektronischer Patientendatensätze – Dabei geht es nicht nur um medizinische Datensätze, sondern auch um Informationen über die Versicherung sowie Behandlungs- und Rechnungsdaten. Das ist vielleicht eine der unbekanntesten Anwendungen, hat aber die meisten Konsequenzen. Irgendwann müssen die westlichen Länder die Kosten für ihr Gesundheitswesen exakt überwachen: ein Bereich, in dem die administrativen Kosten die tatsächlichen Behandlungskosten scheinbar übersteigen.
- OFE-Spezifikation (*Open Financial Exchange*) – Ein weiterer Vorschlag für die Formatierung und Protokollierung für E-Kommerz.
- HDML (*Handheld Device Markup Language*) – Eine Art Mini-HTML, das erlaubt, Handgeräte (PDAs wie PalmPilot, Mobiltelefone und Palmtop-Computer) für das Web-Browsing und die Kommunikation über das Internet einzusetzen.
- CDF (*Channel Definition Format*) – Einst unter dem Namen *Push Media* als Durchbruch einer neuen Form des Web-Publishing bejubelt, ist CDF zu einer praktischen Methode geworden, über den Internet Explorer Abos für Web-Sites zu bestellen, die dann regelmäßig aktualisierte Informationen liefern (Nachrichten, Aktienkurse usw.).
- VXML (*Visual XML*) – Eine Methode, Web-Sites zu beschreiben und sie in VRML (*Virtual Reality Modeling Language*) zu veröffentlichen. Damit erhält der Browser eine virtuelle 3D-Navigationsoberfläche (ähnlich dem, was die Hyperwave-Leute jahrelang betrieben haben).
- SDML (*Signed Document Markup Language*) – Eine Methode, elektronische Dokumente elektronisch zu signieren und zu analysieren (ähnlich den Verschlüsselungsmechanismen mit den öffentlichen Schlüsseln, die die Seele von PGP (*Pretty Good Privacy*) bilden).
- TMX (*Translation Memory Exchange*) – Eine Möglichkeit, Vokabular-Datenbanken zwischen (professionellen) Übersetzungs- und Lokalisierungspaketen auszutauschen.
- OpenTag – Eine Methode, XML-Tags in Text einzufügen, um zu extrahieren, zu übersetzen und die übersetzten Dinge neu einzufügen.
- P3P (*Platform for Privacy Preferences*) – Eine Methode für Web-Sites, RDF und XML zur Kennzeichnung ihres Inhalts zu verwenden (wie beispielsweise Informationen, die nur für Erwachsene vorgesehen sind).
- WebBroker – Eine Methode für verteilte Softwarekomponenten, über das Internet miteinander zu kommunizieren.
- DML (*Development Markup Language*) – Eine Methode für Organisationen der Entwicklungshilfe (z.B. WHO, UNESCO, OECD, Rockefeller Foundation oder World Bank), Informationen auszutauschen.

- XML-QL (*XML Query Language*) – Eine SQL-Implementierung in XML.
- BSML (*Bioinformatic Sequence Markup*) – Stellt Informationen über genetische Zusammenhänge dar.
- CML (*Chemical Markup Language*) – Stellt Informationen über chemische Moleküle dar.
- SMIL (*Synchronized Multimedia Integration Language*) – Integriert unabhängige Multimedia-Objekte (Video, Grafik, Audio usw.) in eine synchronisierte Multimedia-Darstellung.

Das war nur ein kurzer Blick auf die zahlreichen Anwendungen, die man für XML gefunden hat. Einige davon werden in großen Unternehmen eingesetzt (Visa, Microsoft, Oracle, um nur ein paar zu nennen), andere dagegen werden nur von einzelnen Benutzern verwendet.

Ich kann diese Anwendungen hier nicht detailliert vorstellen. Deshalb habe ich eine der interessantesten ausgewählt, MathML (*Mathematics Markup Language*), die hier genauer beschrieben werden soll. Dabei werde ich mich auf zwei Anwendungsbereiche konzentrieren (Grafik und Verhalten), wo XML vermutlich den größten Einfluß auf die nichtprofessionellen Benutzer hat.

Mathematics Markup Language

Die Computer-Satzsprache TeX ist alles andere als benutzerfreundlich, hat aber größte Verbreitung erfahren, weil es sonst kaum Möglichkeiten gab, Mathematik im Schriftsatz darzustellen.

Irgendwie haben es die Mathematik und HTML nie geschafft, auf einen Nenner zu gelangen. Die Erweiterung von HTML stand so lange aus, daß sie jetzt von XML überholt wurde. HTML wird wahrscheinlich nie für die Ausgabe von Mathematik aufgerüstet werden. Wenn Sie heute grundlegende mathematische Gleichungen auf einer Webseite darstellen möchten, können Sie entweder einen Screenshot anlegen und als Grafikdatei in Ihr Dokument importieren oder Stunden extrem frustrierender Kleinarbeit darauf verwenden, die richtigen Symbole in der richtigen Schriftart zu finden. (Hoffen Sie darauf, daß die Leser auf der anderen Seite dieselbe Geduld aufbringen, ebenfalls die richtige Schrift zu finden.)

MathML versucht, eine Lösung zu finden – und stellt eine Markup-Sprache für Mathematik bereit. Es gibt sogar einen Web-Browser, der Teile von MathML unterstützt. Sie können den Amaya-Browser von der W3C-Site herunterladen (`http://www.w3c.org/`). MathML stellt eine sehr interessante Variation eines Themas dar, das die SGML-Welt

von Anfang an beschäftigt hat: die scheinbar konkurrierenden Interessen von darstellendem und semantischem (informationsbasiertem) Markup. MathML bringt diese Interessen parallel im selben Dokument zusammen.

Darstellendes Markup hat insbesondere mit dem endgültigen Erscheinungsbild zu tun. Die HTML-Elemente <HR> und <BLINK> sind für die Darstellung vorgesehen.

Semantisches Markup ist viel mehr daran interessiert, den Informationsinhalt eines Dokuments auszuzeichnen. Damit wird es möglich, auch andere Dinge mit den Daten zu tun, beispielsweise eine hörbare Darstellung für Blinde oder (im Fall von MathML) sogar eine Weitergabe an ein Computer-Algebra-System, das Gleichungen für Sie darstellen und sogar berechnen kann.

Ein Bild sagt mehr als tausend Worte. MathML will darstellungs- und inhaltsbasierte Markup-Schemata vereinigen. Betrachten Sie die einfache Gleichung $x(x + 4) = 1$. Durch Ausmultiplizieren der Klammern erhalten Sie $x^2 + 4x - 1 = 0$, was mit rein darstellenden Codes ausgegeben werden kann, wie in Listing 21.1 gezeigt.

Listing 21.1: Ein Beispiel für das Darstellungs-Markup von MathML

```
1:  <MROW>
2:    <MROW>
3:      <MSUP>
4:        <MI>x</MI>
5:        <MN>2</MN>
6:      </MSUP>
7:      <MO>+</MO>
8:      <MROW>
9:        <MN>4</MN>
10:       <MO>&InvisibleTimes;</MO>
11:       <MI>x</MI>
12:     </MROW>
13:       <MO>-</MO>
14:       <MN>1</MN>
15:   </MROW>
16:   <MO>=</MO>
17:   <MN>0</MN>
18: </MROW>
```

Diese Codes beschreiben nur das Erscheinungsbild der Symbole auf der Seite. Vergleichen Sie das mit Listing 21.2, wo dieselbe Gleichung unter Verwendung semantischer Markup-Elemente ausgezeichnet wird.

Listing 21.2: Ein Beispiel für das semantische Markup von MathML

```
1:  <EXPR>
2:    <EXPR>
3:      <EXPR>
4:        <MI>x</MI>
5:        <POWER>
6:        <MN>2</MN>
7:      </EXPR>
8:      <PLUS/>
9:      <EXPR>
10:       <MN>4</MN>
11:       <TIMES>
12:       <MI>x</MI>
13:     </EXPR>
14:     <MINUS/>
15:     <MN>1</MN>
16:   </EXPR>
17:   <E/>
18:   <MN>0</MN>
19: </EXPR>
```

Die Listings 21.1 und 21.2 zeigen zwei Markup-Zuordnungen, die völlig unterschiedliche Ansichten desselben Objekts erzeugen. Eine davon reiht relativ bedeutungslose Symbole hintereinander auf, wo Begriffe wie Hochstellung (Element <MSUP>) und Tiefstellung (Element <MSUB>) vorherrschen. Das andere Listing enthält Markup, das semantisch sinnvolle Anweisungen darstellt, das aus Ausdrücken (Element <EXPR>) besteht.

Diese beiden Markup-Ansichten sind jedoch nicht völlig gegensätzlich und haben beide ihre Berechtigung. Das darstellende Markup ist optimal für die Anzeige geeignet, und es besteht sogar die Möglichkeit, daß ein Ausdruck Sinn ergibt, wenn er laut gelesen wird. Das inhaltsorientierte Markup dagegen repräsentiert die mathematische Bedeutung eines Ausdrucks, so daß die Anweisungen verständlich werden.

MathML hat erkannt, daß beide Markup-Arten erforderlich sind, und verwendet deshalb eine Art übergeordnetes Element – <SEMANTIC>. Dieses Element besitzt zwei Kindelemente. Das erste Kindelement ist das darstellende Markup, das zweite Kindelement ist das semantische Markup. In MathML könnte das semantische Markup als MathML-Inhalts-Tags erscheinen, was aber nicht zwingend so sein muß (es könnte sich um Code handeln, der für andere Programme sinnvoll ist). Der Inhalt des semantischen Markups könnte ein Computer-Algebra-Ausdruck sein oder sogar Quellcode eines Computerprogramms (in C oder sogar Java).

Momentan gibt es keine Unterstützung der semantischen Komponente von XML, obwohl bereits damit experimentiert wird. Der Amaya-Browser unterstützt aber die darstellende Komponente von XML. Listing 21.3 zeigt einige einfache Beispiele für eine der grundlegendsten Funktionen der Mathematik (und viel anderen Text), Hochstellungen und Tiefstellungen. Abbildung 21.1 zeigt, wie sie in Amaya aussehen.

Listing 21.3: Das darstellende Markup in MathML für Hochstellungen und Tiefstellungen

```
1:  <html>
2:  <head>
3:  <title>MathML Examples 1</title>
4:  </head>
5:  <body>
6:  <h1>Subscripts and Superscripts</h1>
7:  <hr>
8:  <h4>A Simple Subscript</h4>
9:
10: <math>
11:   <msub>
12:     <mi>x</mi>
13:     <mn>y</mn>
14:   </msub>
15: </math>
16:
17: <hr>
18: <h4>A Simple Superscript</h4>
19: <math>
20:   <msup>
21:     <mi>x</mi>
22:     <mn>a</mn>
23:   </msup>
24: </math>
25:
26: <hr>
27: <h4>A Subscript Squared</h4>
28: <math>
29:   <msup>
30:     <msub>
31:       <mi>x</mi>
```

```
32:       <mn>n</mn>
33:     </msub>
34:     <mn>2</mn>
35:   </msup>
36: </math>
37: <p></p>
38: <hr>
39: <h4>A Subscript and a Superscript Combined</h4>
40:
41: <math>
42:   <msubsup>
43:     <mi>x</mi>
44:     <mn>a</mn>
45:     <mn>b</mn>
46:   </msubsup>
47: </math>
48: <p></p>
49: <hr>
50: </body>
51: </html>
```

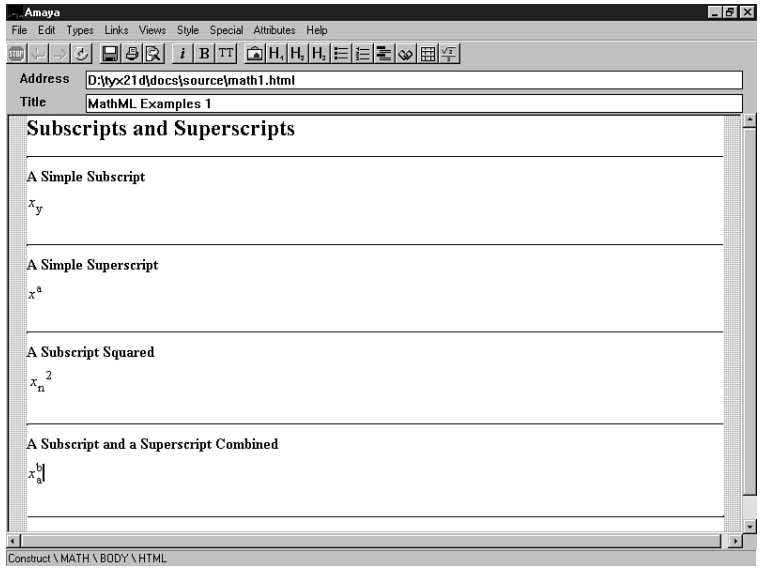


Abbildung 21.1: So zeigt Amaya den MathML-Code aus Listing 21.3 an.

Hochstellungen und Tiefstellungen sind nicht besonders kompliziert. Es gibt sogar HTML-Zeichencodes, die Ihnen ermöglichen, diese kleinen Ziffern einzufügen. Listing 21.4 zeigt etwas Komplexeres – ein Beispiel für den MathML-Code für ein paar einfache Matrizen. Abbildung 21.2 zeigt, wie der Code in Amaya dargestellt wird.

Listing 21.4: Matrizen in MathML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>
MathML Examples 2</title>
</head>
<body>
<h1>Matrices</h1>
<hr>

<h4>A Simple Matrix</h4>
<math>
  <mfenced open="[" close="]">
    <mfrac>
      <mi>a</mi>
      <mi>b</mi>
    </mfrac>
  </mfenced>
</math>
<hr>

<h4>Complex Braces (fences)</h4>
<math>
  <mfenced open="(" close="]">
    <mfrac>
      <mi>a</mi>
      <mi>b</mi>
    </mfrac>
  </mfenced>
</math>
```

```
31: <hr>
32:
33: <h4>A More Complex Matrix (math table)</h4>
34: <math>
35:   <mfenced open="(" close=")">
36:     <mtable>
37:       <mtr>
38:         <mtd>
39:           <mn>1</mn>
40:         </mtd>
41:         <mtd>
42:           <mn>0</mn>
43:         </mtd>
44:         <mtd>
45:           <mn>0</mn>
46:         </mtd>
47:       </mtr>
48:       <mtr>
49:         <mtd>
50:           <mn>0</mn>
51:         </mtd>
52:         <mtd>
53:           <mn>1</mn>
54:         </mtd>
55:         <mtd>
56:           <mn>0</mn>
57:         </mtd>
58:       </mtr>
59:       <mtr>
60:         <mtd>
61:           <mn>0</mn>
62:         </mtd>
63:         <mtd>
64:           <mn>0</mn>
65:         </mtd>
66:         <mtd>
67:           <mn>1</mn>
68:         </mtd>
69:       </mtr>
70:     </mtable>
71:   </mfenced>
72: </math>
73: <hr>
74: </body>
75: </html>
```

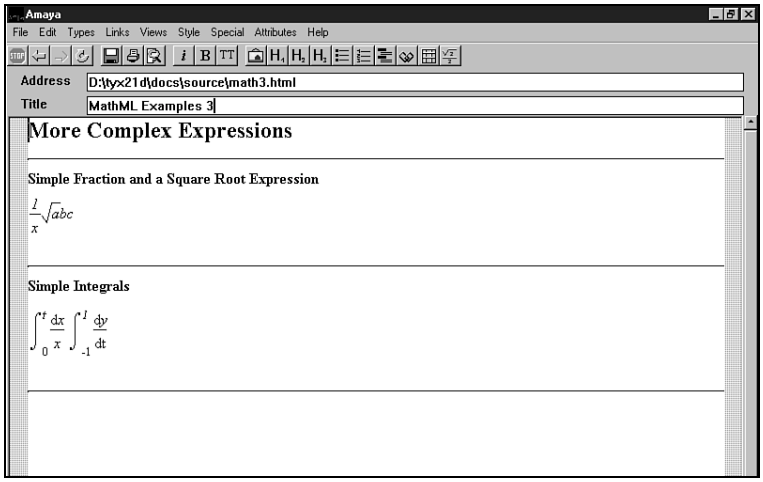


Abbildung 21.2: So zeigt Amaya den MathML-Code aus Listing 21.4 an.

Es ist immer schwierig, Matrizen korrekt darzustellen. Die Klammern müssen so angelegt werden, daß sie zum Inhalt passen. Normalerweise versucht man, sie aus separaten Zeichen für die Enden und vertikalen Linien für die Mittelstücke zusammenzusetzen. Listing 21.4 zeigt, wie einfach es sein kann, beliebige Klammern auszugeben. Sogar ihre Größe kann automatisch angepaßt werden, wie in Abbildung 21.2 gezeigt.

Matrizen allein sind schon eine Errungenschaft, aber was wäre die Mathematik ohne Integrale und all die anderen seltsamen Symbole? Listing 21.5 zeigt einige komplexe Gleichungen. Abbildung 21.3 zeigt, wie sie in Amaya ausgegeben werden.

Listing 21.5: Komplexere Mathematik mit dem darstellenden Markup von MathML

```
1:  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2:  <html>
3:  <head>
```

```
4:  <title>MathML Examples 3</title></head>
5:  <body>
6:  <h1>More Complex Expressions</h1>
7:  <hr>
8:  <h4>Simple Fraction and a Square Root Expression</h4>
9:  <math>
10:   <mfrac>
11:     <mi>1</mi>
12:     <mi>x</mi>
13:    </mfrac>
14:   <msqrt>
15:     <mi>a</mi>
16:     <mi>b</mi>
17:     <mi>c</mi>
18:   </msqrt>
19: </math>
20: <p></p>
21: <hr>
22: <h4>Simple Integrals</h4>
23: <math>
24:   <mrow>
25:     <msubsup>
26:       <mo>&int;</mo>
27:       <mn>0</mn>
28:       <mi>t</mi>
29:     </msubsup>
30:     <mfrac>
31:       <mrow>
32:         <mo>&dd;</mo>
33:         <mi>x</mi>
34:       </mrow>
35:       <mi>x</mi>
36:     </mfrac>
37:   </mrow>
38:   <mrow>
39:     <msubsup>
40:       <mo>&int;</mo>
41:       <mn>-1</mn>
42:       <mi>1</mi>
43:     </msubsup>
44:     <mfrac>
45:       <mrow>
46:         <mo>&dd;</mo>
47:         <mi>y</mi>
48:       </mrow>
```

```
49:        <mi>dt</mi>
50:      </mfrac>
51:    </mrow>
52: </math>
53: <p></p>
54: <hr>
55: </body>
56: </html>
```

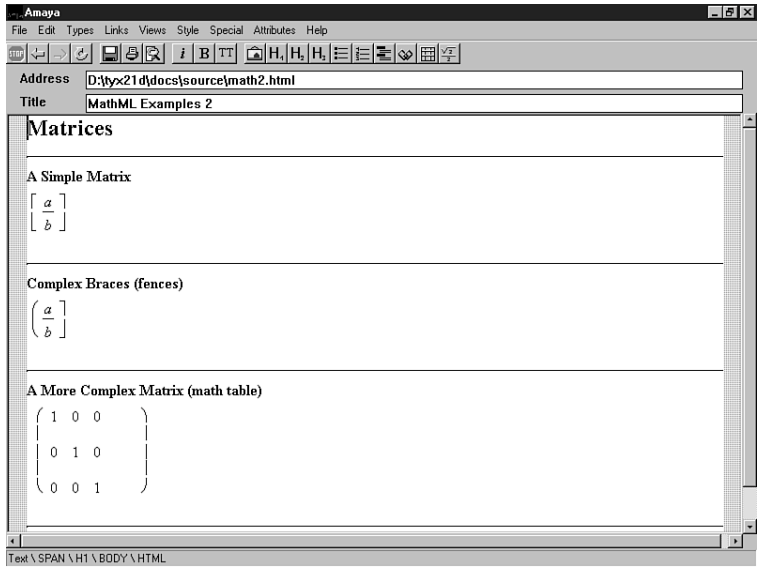


Abbildung 21.3: So zeigt Amaya den MathML-Code aus Listing 21.5 an.

Wie Sie sehen, macht es uns Amaya leicht, alle möglichen mathematischen Symbole darzustellen.

MathML ist noch lange nicht fertig – es gibt zahlreiche Funktionen, die noch ausgearbeitet werden müssen. Außerdem wird es noch längst nicht von den wichtigsten Web-Browsern unterstützt. Es gibt jedoch ein Browser-Plug-in, EzMath, das zwar einen proprietären Mathematikcode verwendet, aber der Editor kann auch MathML-Code erzeugen. Sie können diese kostenlose Software unter `http://www.w3.org/People/Raggett/EzMath.zip` herunterladen.

Strukturierte Grafik

Kaum waren die Web-Browser in der Lage, Grafik anzuzeigen, wollten die Benutzer, daß ständig irgend etwas passiert, wenn sie auf die Bilder klickten. Es folgten verknüpfte Bilder, dann Schaltflächen und dann HTML-Image Maps – zuerst die vom Server interpretierten (Server-seitig), und dann die vom Browser selbst interpretierten (Client-seitig).

Image Maps erlauben, eine Verknüpfung abhängig davon anzulegen, wo im Bild geklickt wird. Leider sind diese Image Maps einfache Listen sehr einfacher Objekte (Kreise, Quadrate und regelmäßige Polygone) mit festen Koordinaten. Listing 21.6 zeigt ein Beispiel für eine Client-seitige HTML-Image Map.

Listing 21.6: Eine typische Client-seitige HTML-Image Map

```
1: <map name="map">
2: <area shape="rect" coords="33,78,58,88" href="1.html">
3: <area shape="circle" coords="164,207,57" href="2.html">
4: <area shape="poly" coords="146,35,165,69,134,127,123,96,
5:     105,77,69,82,39,81,20,66,19,55,51,54,118,31,0,19474"
6:     href="3.html">
```

Wie jeder, der es versucht hat, Ihnen bestätigen kann, ist es ein Alptraum, diese Koordinatengrafiken zu verwalten. Immer wenn sich die Grafik ändert, müssen Sie die Image Map völlig neu anlegen. Es gibt einige Werkzeuge, die eine Image Map lesen und abändern können, aber nichtsdestotrotz muß sie in der HTML-Datei ersetzt werden.

Erschwerend kommt hinzu, daß es sich bei den meisten Grafiken heute um Bitmaps handelt. Bitmaps sind im wesentlichen binäre Daten, die die Farbe der Punkte (Pixel) auf dem Computerbildschirm darstellen. Weil sie bestimmte Lichtpunkte ansprechen, sind Bitmaps sehr empfindlich gegenüber der verwendeten Bildschirmauflösung. Ich verwende beispielsweise eine Bildschirmauflösung von 1152 x 864 Pixeln mit 65.536 Farben. Wenn ich einen Screenshot für dieses Buch aufzeichnen will, muß ich aus technischen Gründen auf 800 x 600 Pixel und 256 Farben zurückgehen. Was in höheren Auflösungen gut aussieht, kann in niedrigeren Auflösungen schrecklich wirken und umgekehrt. Die Situation ist kaum anders, wenn ich eine Grafik vergrößern will, unabhängig von der verwendeten Auflösung. Früher oder später wird Bitmap-Grafik in verschwommenen Farbrechtecken dargestellt, so daß Sie die einzelnen Pixel auf dem Bildschirm ansprechen können. Abbildung 21.4 verdeutlicht dieses Problem in der stark vergrößerten Darstellung einer digitalen Fotografie.

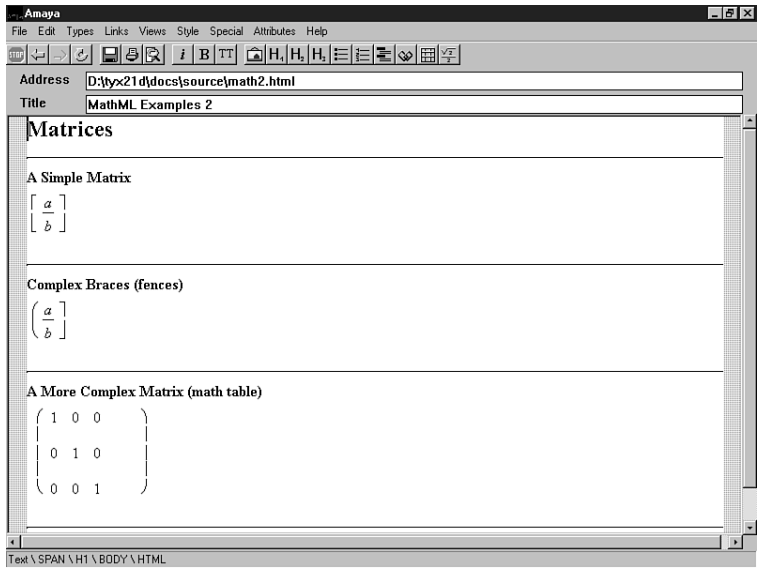


Abbildung 21.4: Ein stark vergrößertes Foto, dessen einzelne Pixel sichtbar sind.

Es gibt viele alternative Formate für Bitmaps (beispielsweise TIFF, GIF, JPEG und BMP, um nur ein paar zu nennen). Jedes dieser Formate hat Vor- und Nachteile und deshalb auch ganz bestimmte Einsatzbereiche. Für Vektorgrafiken gibt es zwei konkurrierende Formate: WMF (*Windows Metafile*) und CGM (*Computer Graphics Metafile*).

Statt einzelne Pixel zu beschreiben, verwendet die Vektorgrafik elementare Objekte, beispielsweise Linien und Bögen. Diese Objekte werden anhand ihres Anfangspunkts, ihrer Richtung und ihrer Länge (Vektoren) beschrieben. Die anzeigende Software muß diese Daten in eine Bildschirmdarstellung umwandeln, wodurch die Vektorgrafik völlig unabhängig von der Auflösung des Ausgabegeräts wird. Vektorgrafik kann nahezu beliebig vergrößert oder verkleinert werden (bis zu einem Grad, wo das, was Sie sehen, überhaupt keinen Sinn mehr ergibt).

Bitmaps sind Rasterformate, die ein Bild in ein Raster gleich großer Stücke zerlegen, sogenannte Pixel, und die Farbinformationen für jedes einzelne Pixel aufzeichnen.

Es gibt noch ein drittes Format. Meta-Formate können Vektordaten und ein Rasterbild enthalten. Eine WMF-Datei (*Windows Metafile*) beispielsweise kann eine Bitmap, Vektorinformationen und Text enthalten, wobei die Bitmap den Hauptteil des Bildes einnimmt und die Vektor- und Textdaten für Kommentare genutzt werden.

Vektorgrafiken können einfach als Text (ASCII) dargestellt werden. Damit taucht sofort die Frage auf, ob sie nicht auch in XML ausgedrückt werden können. Die Antwort auf diese Frage ist ein klares Ja, und es gibt bereits mehrere Initiativen, die diese Idee verfolgen.

WebCGM

WebCGM ist keine XML-Anwendung; deshalb werde ich mich hier nicht allzu detailliert damit beschäftigen – dennoch soll es erwähnt werden.

CGM (*Computer Graphics Metafile*) ist seit 1987 ein ISO-Standard und seit 1995 ein anerkanntes Internet-Format (MIME-Typ). Obwohl es das bevorzugte Format für technische Dokumentationen ist, hat es sich im Web nicht wirklich durchgesetzt (es gibt ein Browser-Plug-in, mit dem Sie CGM-Daten anzeigen können). Und es gibt auch keine größeren Softwarepakete, die das Format vollständig unterstützen. Die meisten Grafikpakete erlauben Ihnen, CGM-Rasterdaten anzuzeigen und möglicherweise auch zu schreiben, aber es gibt nur sehr wenige Pakete, die CGM-Vektordaten verarbeiten können. Nichtsdestotrotz haben Boeing (einer der wichtigsten Anwender) und einige der führenden Software-Unternehmen in diesem Bereich (unter anderem Intercap, Inso und ITEDO) ein CGM-Profile (WebCGM) vorgeschlagen, das Metadaten einführt, die für das Hyperlinking, die Navigation in Dokumenten und die Suche nach Bildinhalten verwendet werden können.

PGML – Precision Graphics Markup Language

PGML ist die Erweiterung des PostScript-/PFD-Formats von Adobe. Einfach ausgedrückt, übersetzt PGML PostScript-Befehle in XML-Elemente und -Attribute.

Viele Leute wissen, daß PostScript eine Seitenbeschreibungssprache ist – eine formale Beschreibung des Layouts einer Seite. Beim Ausdruck von PostScript-Dokumenten senden Sie diverse PostScript-Befehle, in die Ihre Daten eingebettet sind, an einen entsprechenden Prozessor im Drucker. Der Prozessor interpretiert diese Befehle und führt sie aus.

Listing 21.7 zeigt den PostScript-Code aus einer kleinen Testdatei, die die Zeichen einer Schrift in verschiedenen Größen ausgibt. Die resultierende Ausgabe sehen Sie in Abbildung 21.5.

Listing 21.7: Eine typische PostScript-Datei

```
1:  % Check for command line parameters:
2:  %       Name, FirstSize, Ratio, NumSizes, UseOutline.
3:
```

```
4:  /FontName where { pop } { /FontName (Palatino-Italic) def } ifelse
5:  /FirstSize where { pop } { /FirstSize 15 def } ifelse
6:  /Ratio where { pop } { /Ratio 1.6 def } ifelse
7:  /NumSizes where { pop } { /NumSizes 3 def } ifelse
8:  /UseOutline where { pop } { /UseOutline false def } ifelse
9:
10: /Strings FirstSize 20 gt
11:  { [
12:         (ABCDEFGHIJ) (KLMNOPQR) (STUVWXYZ)
13:         (abcdefghijklm) (nopqrstuvwxyz)
14:         (0123456789<=>) (:;?@  !"#$%&')
15:         (\(\)*+,-./[\\]^_) (`{¦}~)
16:  ] }
17:  { [
18:         (ABCDEFGHIJKLMNOPQRSTUVWXYZ)
19:         (abcdefghijklmnopqrstuvwxyz)
20:         (0123456789<=>:;?@  !"#$%&')
21:         (\(\)*+,-./  [\\]^_  `{¦}~)
22:  ] }
23: ifelse def
24:
25: /sshow
26:  { gsave UseOutline
27:     { { gsave ( ) dup 0 4 -1 roll put
28:         false charpath pathbbox 0 setlinewidth stroke grestore
29:         pop 8 add currentpoint exch pop moveto pop
30:       } forall
31:     }
32:     { 2 0 3 -1 roll ashow }
33:    ifelse grestore
34:  } def
35:
36: FontName findfont FirstSize scalefont setfont
37:
38: clippath pathbbox /top exch def pop pop pop newpath
39: 10 10 moveto
40: NumSizes
41:  { gsave nulldevice (Q) false charpath pathbbox grestore
42:      exch pop exch sub exch pop 1.25 mul /height exch def
43:    Strings
44:     { currentpoint exch pop top height 3 mul sub gt
45:        { showpage 10 10 height sub moveto
46:        }
47:       if
48:       dup sshow
49:       UseOutline not
```

```
50:        { 0 height rmoveto gsave 0.01 rotate sshow grestore }
51:       if
52:       0 height rmoveto
53:     } forall
54:    Ratio dup scale
55:  } repeat
56: showpage
```

Abbildung 21.5: Der PostScript-Code aus Listing 21.7 sieht bei der Anzeige als PDF-Datei (Adobe Acrobat) so aus.

Wie Sie in Listing 21.7 sehen, sind die gesendeten Befehle relativ deutlich. Bei dem PDF-Format von Adobe handelt es sich zwar um ein binäres Format, aber wenn Sie die Binärcodes herausfiltern, sehen Sie immer noch Fragmente der ursprünglichen PostScript-Befehle, wie in Listing 21.8 gezeigt.

Listing 21.8: Ein Ausschnitt einer PostScript-Datei aus Listing 21.7, nachdem diese in das PDF-Format (Adobe Acrobat) umgewandelt wurde

```
1:  <<
2:  /ProcSet [/PDF /Text ]
3:  /Font <<
4:  /F2 4 0 R
5:  >>
6:  /ExtGState <<
7:  /GS1 5 0 R
8:  >>
9:  >>
10: endobj
11: 7 0 obj
12: <<
13: /Type /Halftone
14: /HalftoneType 1
15: /HalftoneName (Default)
16: /Frequency 60
17: /Angle 45
18: /SpotFunction /Round
19: >>
20: endobj
21: 5 0 obj
22: <<
23: /Type /ExtGState
24: /SA false
25: /OP false
26: /HT /Default
27: >>
28: endobj
29: 8 0 obj
30: <<
31: /Type /FontDescriptor
32: /Ascent 733
33: /CapHeight 692
34: /Descent -276
35: /Flags 98
36: /FontBBox [-170 -276 1010 918]
37: /FontName /Palatino-Italic
38: /ItalicAngle -10
```

```
39: /StemV 84
40: /XHeight 482
41: >>
42: endobj
```

Ich will hier nicht zu sehr ins Detail gehen. PostScript verwendet einige sehr grundlegende Befehle, die den Pfad festlegen (wie etwa `moveto`, `lineto` und `curveto`), und Attribute (wie `height`, `linewidth` oder `rgbcolor`). Es ist ganz einfach, diese Anweisungen in XML-Elemente und Attribute umzuwandeln. Listing 21.9 zeigt eine sehr einfache PGML-Datei.

Listing 21.9: Eine einfache PGML-Datei

```
1:  <?xml version="1.0"?>
2:  <!DOCTYPE PGML SYSTEM "pgml1.0.dtd">
3:  <pgml boundingbox="0 0 300 300">
4:    <path fill="1" fillcolor="100 0 0">
5:      <moveto x="100" y="100"/>
6:      <lineto x="150"/>
7:      <lineto y="150"/>
8:      <lineto x="100"/>
9:      <lineto y="100"/>
10:   </path>
11: </pgml>
```

Der in Listing 21.9 gezeigte PGML-Code erzeugt ein kleines rotes Rechteck: `boundingbox` deklariert die x- und y-Koordinaten des Zeichenbereichs (0,0 als untere linke Ecke bis 300,300). Die `fillcolor`-Anweisungen setzen die Rot-, Grün- und Blauwerte (100,0,0) der Farbe, mit der das Objekt ausgefüllt wird, und `moveto` setzt den Anfangspunkt auf (100,100). Die vier `lineto`-Anweisungen zeichnen die Seitenlinien mit den Ecken an den Positionen (100,100), (150,100), (150,150) und (100,150).

Adobe war extrem mutig, und als Außenstehender kann ich sagen, daß dieser ganzheitliche Ansatz richtig war. PSGML endet nicht mit der Konvertierung von PostScript in XML; er führt die Entwicklung noch ein paar aufregende Schritte weiter. Sie können

- Objekte und Objektgruppen benennen, so daß sie durch Link- oder Such-Software ganz einfach gefunden werden.
- XML-Links in Objekte einbetten.
- Skripten in die PSGML-Datei einbetten, so daß ein Browser sich so verhält, als wären die Scripting-Befehle in der umschließenden HTML-Datei enthalten.
- Ereignis-Handler einbetten (basierend auf dem Dokumentobjektmodell, das Sie in Kapitel 16 kennengelernt haben), die Interaktion und Animation erlauben.

Es gibt noch keine Softwarepakete, die das Anlegen von PGML-Code erlauben, und es gibt auch keine Möglichkeit, Code anzuzeigen oder zu verwenden, nachdem er geschrieben wurde. Die PGML-Spezifikation ist noch nicht einmal zur Hälfte fertig. Es weist jedoch alles darauf hin, daß man es bei Adobe sehr ernst nimmt, und es kann nicht mehr lange dauern, bis das erste Paket auf den Markt kommt.

Vector Markup Language

Zusammen mit dem Internet Explorer Version 4 veröffentlichte Microsoft eine Entwicklung, die als strukturierte Grafik bezeichnet wird. Strukturierte Grafiken liegen in einem ASCII-Vektorformat zur Beschreibung von Zeichnungen vor. Dieses Format ist eigentlich nur von historischem Interesse, aber es wird in Internet Explorer Version 4 und auch in Internet Explorer Version 5 unterstützt. Es gibt ein Werkzeug von Microsoft, `wmfcnv.exe`, das WMF-Dateien (*Windows Metafile Vektorformat*) in strukturierte Grafiken umwandeln kann.

Beachten Sie, daß Sie Vektorformat-Metadateien verwenden müssen. Viele gebräuchliche Grafikpakete (wie etwa PaintShop Pro) können nur die Rasterversion dieser Dateien speichern. Es ist nicht möglich, diese Dateien zu konvertieren (`wmfcnv` ist ein sehr kleines und einfaches Utility – es wird Ihnen nur mitteilen, daß die Ausgabedatei leer ist).

Es gibt nicht viele Pakete, die diese Vektorformatdateien anlegen können; das einzige ist CorelDraw.

Wenn Sie eine fertige Quelle echter Vektorformatdateien brauchen, sehen Sie in der in Microsoft Word oder Microsoft Office enthaltenen ClipArt-Bibliothek nach, oder besuchen Sie die ClipArt-Webseiten auf der Web-Site von Microsoft (`www.microsoft.com`).

Listing 21.10 zeigt eine sehr einfache SGF-Datei, die ein in eine HTML-Datei eingebettetes Yin-Yang-Symbol anlegt.

Listing 21.10: Ein Yin-Yang-Symbol im SGF-Format

```
<html>
<head>
<title>Structured Graphic</title>
</head>
<body>
<H1>Animated Yin-Yang</H1>
<p>
<object ID="yin-yang"
CLASSID="CLSID:369303C2-D7AC-11D0-89D5-00A0C90833E6"
STYLE="height: 200; width:200; zindex:10">
<PARAM NAME="LineOOO1" VALUE="SetLineColor(0,0,0)">
<PARAM NAME="LineOOO2" VALUE="SetFillColor(255,0,0)">
<PARAM NAME="LineOOO3" VALUE="SetLineStyle(6)">
<PARAM NAME="LineOOO4" VALUE="Polygon(102,2,5,-1,3,-3,1,-8,-3,-11,
-5,-12,-8,-14,-11,-15,-14,-17,-18,-17,-20,-17,-23,-17,-26,-16,-29,
-15,-32,-14,-35,-12,-38,-9,-41,-7,-43,-5,-44,-3,-46,-1,-47,2,-47,4,
-48,6,-48,8,-48,13,-48,9,-49,7,-49,4,-49,1,-49,-2,-49,-5,-49,-7,-49,
-10,-49,-13,-48,-17,-47,-20,-46,-23,-44,-26,-43,-29,-41,-31,-39,-33,
-37,-35,-35,-37,-33,-39,-31,-41,-29,-42,-27,-44,-24,-45,-22,-46,-19,
-47,-17,-48,-14,-49,-11,-50,-8,-50,-5,-50,-2,-50,1,-50,4,-50,7,-49,11
,-49,14,-48,17,-47,19,-46,22,-45,24,-43,27,-41,29,-39,32,-38,34,-36,
36,-34,38,-32,39,-30,41,-27,43,-26,44,-24,45,-22,46,-20,46,-17,47,
-15,47,-13,47,-11,47,-8,47,-6,46,-3,45,1,44,1,43,3,41,4,40,6,
38,7,35,8,33,9,31,10,28,10,25,10,23,10,20,9,17,8,14,7,12
,6,10,4,7)">
<PARAM NAME="LineOOO5" VALUE="SetFillColor(255,255,0)">
<PARAM NAME="LineOOO6" VALUE="Polygon(102,-3,-4,0,-2,
2,0,7,4,10,6,11,9,13,12,14,15,16,18,16,21,16,24,16,27,
15,30,14,33,13,36,11,39,8,42,6,44,4,45,2,47,0,48,-3,48,
-5,49,-7,49,-9,49,-14,49,-10,49,-8,50,-5,50,-2,50,1,50,4,
50,6,50,9,50,12,49,16,48,19,47,22,45,25,44,28,42,30,40,32,
38,34,36,36,34,38,32,40,30,41,28,43,25,44,23,45,20,46,18,
47,15,48,12,49,9,49,6,49,3,50,0,49,-3,49,-6,48,-10,48,-13,
47,-16,46,-18,45,-21,44,-23,42,-26,40,-28,38,-31,37,-33,35,
-35,33,-37,31,-38,29,-40,26,-42,25,-43,23,-44,21,-45,19,-45,
16,-46,14,-46,12,-46,10,-46,7,-46,5,-45,2,-44,0,-43,-2,-42,-4,
-40,-5,-39,-7,-37,-8,-35,-9,-32,-10,-30,-11,-27,-11,-24,-11,
-22,-11,-19,-10,-16,-9,-13,-8,-11,-7,-9,-5,-6)">
</object>
</body>
</html>
```

Abbildung 21.6 zeigt die Darstellung dieser Datei im Internet Explorer 5.

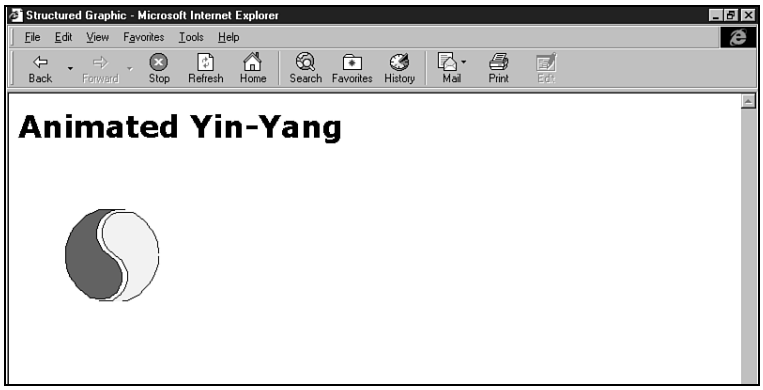


Abbildung 21.6: So sieht Listing 21.10 im Internet Explorer 5 aus.

Das SGF-Format in Listing 21.10 bietet offensichtlich nicht den hohen Informationsgehalt von PGML, funktioniert aber. Im allgemeinen ist das SGF-Format etwa 20 % kleiner als eine äquivalente Windows-Metafile-Datei, die schon sehr kompakt sein kann, wenn sie nur ein Vektorbild enthält. Je größer die Datei, desto mehr Einsparung.

Eine interessante Eigenschaft dieses Formats ist die einfache Animation. Mit nur neun zusätzlichen Zeilen kann ich diese Zeichnung so animieren, daß sie sich dreht, wenn Sie auf die Schaltfläche klicken, wie in Listing 21.11 gezeigt.

Listing 21.11: Animation des Yin-Yang-Symbols

```
1:  <html>
2:  <head>
3:  <title>Structured Graphic</title>
4:  <SCRIPT LANGUAGE="JavaScript">
```

```
5:  <!--
6:  function Rotate(degVar){
7:    widget.rotate(0,degVar,0);
8:    window.setTimeout("Rotate(5)",040,"JavaScript");
9:  }
10: -->
11: </script>
12: </head>
13: <body>
14: <H1>Animated Yin-Yang</H1>
15: <p>
16: <object
17: ID="widget"
18:
19: .
20: . put object code here
21: .
22:
23: </object>
24: <p>
25: <INPUT TYPE=button ID=rotate VALUE="ROTATE" onclick="Rotate(0)">
26: </body>
27: </html>
```

Die Abbildungen 21.7, 21.8 und 21.9 zeigen Darstellungen des gedrehten Symbols.

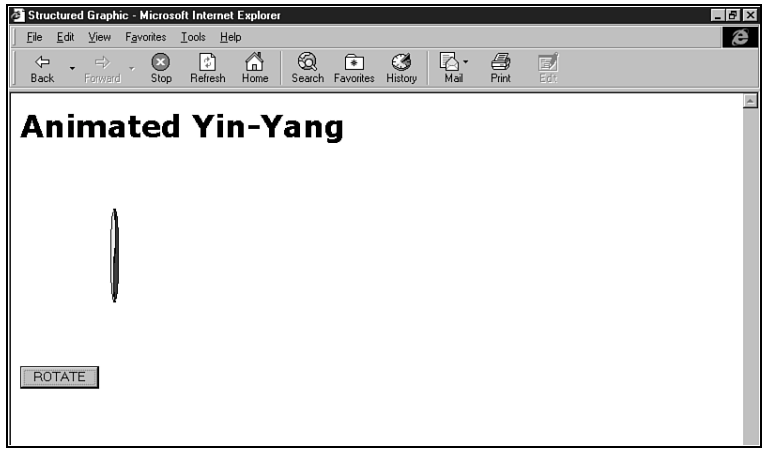


Abbildung 21.7: Eine dritte Ansicht des gedrehten Yin-Yang-Symbols im Internet Explorer 5.

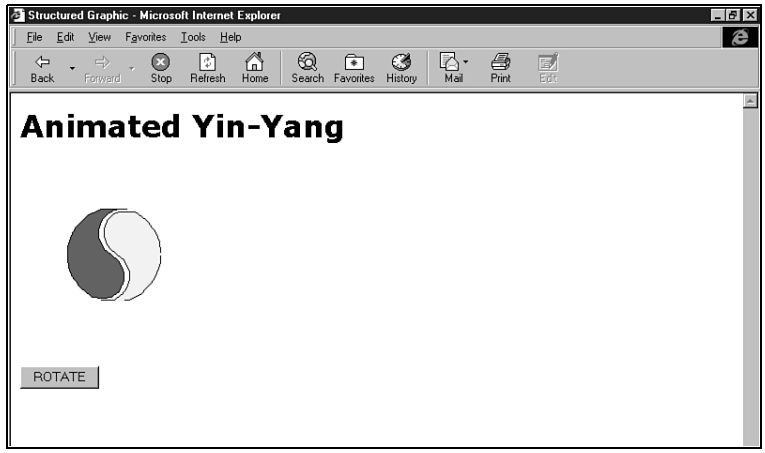


Abbildung 21.8: Ansicht des gedrehten Yin-Yang-Symbols im Internet Explorer 5.

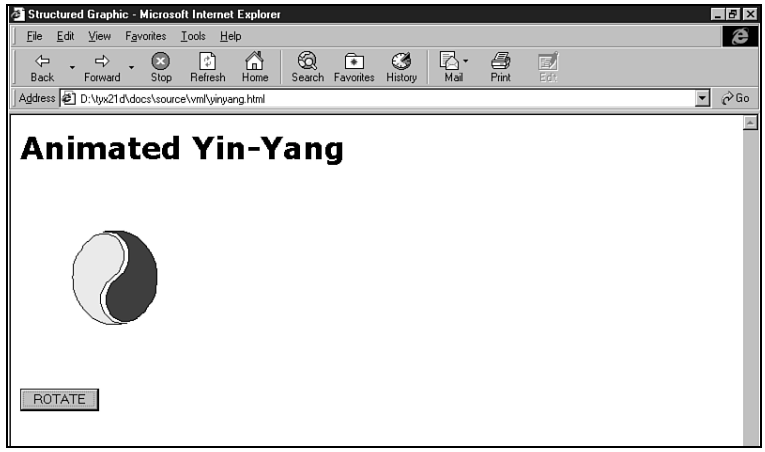


Abbildung 21.9: Eine weitere Ansicht des gedrehten Yin-Yang-Symbols im Internet Explorer 5.

Je öfter Sie auf die Schaltfläche klicken, desto schneller scheint sich das Symbol zu drehen, aber weil eine interne Verarbeitung stattfindet und keine Datei geladen werden muß, wird das Bild relativ schnell neu gezeichnet. Die strukturierte Grafik war ein Vorreiter der VML (*Vector Markup Language*) von Microsoft. VML nutzt die Grundlagen der strukturierten Grafik, baut sie aber in XML-Elemente ein (aus dem VML-Namensraum). Würde ich dafür mein Yin-Yang-Symbol aus Listing 21.10 heranziehen (was ich nicht tue, weil ich es manuell machen müßte und es keine Werkzeuge gibt, das Ergebnis anschließend zu überprüfen), könnte etwa folgendes entstehen:

```
<v:line id="Line0002" from="0 0 " to="100 100"
   style="visibility: visible"
   stroke="true" strokecolor="black">
<v:curve from="0 0" to="0 0" style="visibility: visible">
```

Verhalten

Eine der wichtigsten praktischen Errungenschaften, die SGML mit sich brachte, war die Trennung der Informationsdarstellung von der eigentlichen Information. Für viele professionelle technische Autoren (auch für mich) bedeutete das, daß sie sich nicht mehr darum kümmern mußten, ob sie die richtige Schrift verwendeten oder ob ein Produktname fett oder kursiv dargestellt werden sollte. Man muß nur noch das richtige Element auswählen, in das ein bestimmter Textabschnitt eingebunden werden soll, und die Authoring-Software sorgt für die richtige Darstellung. Außerdem können die Grafiker, die für das Layout zuständig waren, jetzt unabhängig voneinander arbeiten, parallel zu den Autoren, ohne daß sie aufeinander warten müssen.

Webseiten haben sich auf ähnliche Weise weiterentwickelt. Vor der Einführung der Stylesheets befanden sich Inhalt und Grafik unter einem Dach – obwohl viele HTML-Seiten immer noch keine Stylesheets verwenden. Die Einführung von Stylesheets bedeutete, daß Grafiker und Autoren in der Lage waren, ihre Arbeit genau zu unterteilen. Anschließend entdeckten die Benutzer die Interaktion und Animation. Sie wollten bewegten Text, der außerdem in unterschiedlichen Größen angezeigt wurde. Fast über Nacht entstand ein neuer Engpaß, weil jeder auf einen Programmierer warten mußte, der das Skript einfügte, den Code testete, die Seite testete und sie dann auf dem Server bereitstellte. Es entstand Verwirrung darüber, welches Skript verwendet werden sollte (Jscript, JavaScript, ActiveX, Java?) und wo man die Entwickler dieser Skripts finden konnte. Darüber hinaus mußte man die Wiederverwendung von Code ins Auge fassen (Herz und Seele der modernen Software-Entwicklung). Vergessen Sie es – Sie werden nicht mehr als das gute alte Ausschneiden&Einfügen schaffen!

Es war nur eine Frage der Zeit, bis jemand eine Methode fand, das Verhalten von HTML-Code nach außen zu verlagern – und sowohl Netscape als auch Microsoft bieten ihre eigenen Lösungen an.

Action-Sheets

Action-Sheets, genauer gesagt Cascading Action System (CAS), sind eine auf einem Netscape- Vorschlag basierende Spezifikation, die es gestattet, XML-Elementen Verhalten zuzuordnen.

CAS hat zwei Funktionen; für HTML wird einem Element ein Action-Sheet zugeordnet, indem eine kleine Erweiterung der CSS-Syntax (Cascading Stylesheet) genutzt wird. In den HTML-Code fügen Sie einfach ein LINK-Element ein, das auf das Stylesheet verweist, etwa so:

```
<LINK REL="ActionSheet" TYPE="text/act" HREF="flip.act">
```

Das Verhalten, das Sie der Seite (oder einem Teil davon) zuordnen wollen, ist in der externen Skriptdatei `flip.act` enthalten, die in einer beliebigen Sprache, die der Browser versteht, vorliegen kann.

Die Elemente, die dieses Verhalten haben sollen, erhalten einfach das entsprechende Klassenattribut, etwa wie folgt:

```
<P CLASS="flip">This paragraph will flip when you click on it<P>
```

Für XML schlägt Netscape vor, ein Action-Sheet als XML-Dokument anzulegen und es dann dem Ziel-XML-Dokument zuzuweisen, so wie es momentan für die CSS-Stylesheets passiert (unter Verwendung von Verarbeitungsanweisungen, wie in Kapitel 13 und 18 beschrieben). Ein typisches Action-Sheet könnte wie die in Listing 21.1 gezeigte Skizze aussehen.

Listing 21.12: Beispiel für ein Action-Sheet-Dokument

```
1:  <!DOCTYPE actionsheet SYSTEM "asheet.dtd">
2:  <actionsheet>
3:
4:  <action type="text/cas" codetype="text/javascript">
5:  @import url(http://purl.org/simonscripts/flip.cas")
6:  .flip ( onMouseover: "flip(event)"; onMouseOut: "flop(event)")
7:  </action>
8:
9:  <script type="text/javascript" src="flip.js">
10: </script>
11:
12: <script type="text/javascript">
13: function flip(e) { … }
14: function flop(e) { … }
15: </script>
16: </actionsheet>
```

Es gibt noch keine Implementierungen von Action-Sheets.

CSS-Verhalten

Microsoft bezeichnet diese Methode, ein Verhalten einzufügen, als DHTML-Verhalten. Soweit ich weiß, wurde sie dem W3C noch nicht vorgelegt, und ich bin auch nicht sicher, ob das jemals der Fall sein wird.

Das Prinzip dieses Mechanismus ist sehr einfach. Sie fügen dem XML-Element das Skript hinzu, wie in Listing 21.13 gezeigt.

Listing 21.13: Eine HTML-Datei mit dynamischem Verhalten

```
1:  <html>
2:  <head>
3:  <title>Jumping XML</title>
4:  <XML:namespace prefix="IE5"/>
5:  <style>
6:  @namespace IE5 {
7:    getsclass {
8:      font-family: courier; color: red; font-weight: bold;
9:      border-style: solid;  background-color: yellow;
10:     border-color: black; border-width: 3; width: 240;
11:     height: 140; padding: 10
12:     }
13:   }
14: </style>
15: </head>
16: <body>
17: <h1>Application of CSS Behavior to XML</h1>
18: <P>This paragraph is standard HTML code.</p>
19: <IE5:getsclass onclick="this.style.position = 'static';"
20:  style="position: absolute;
21:  top: 200; left: 200">This paragraph is XML code, with behavior
22:   attached to it by a style.
23: Click on it, and it will jump
24:  to a new position.</IE5:getsclass>
25: <P>This paragraph is also standard HTML code,
26:   it will make room for the XML
27: paragraph when it moves.</p></body>
28: </html>
```

Die Abbildungen 21.10 und 21.11 zeigen, was der Browser vor und nach dem Einfügen des Skripts in das XML-Element ausgibt.

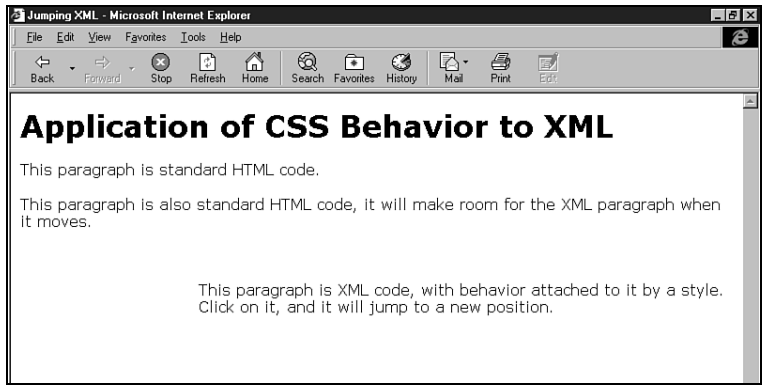


Abbildung 21.10: Der dynamische XML-Abschnitt im Internet Explorer 5 vor dem Anklicken.

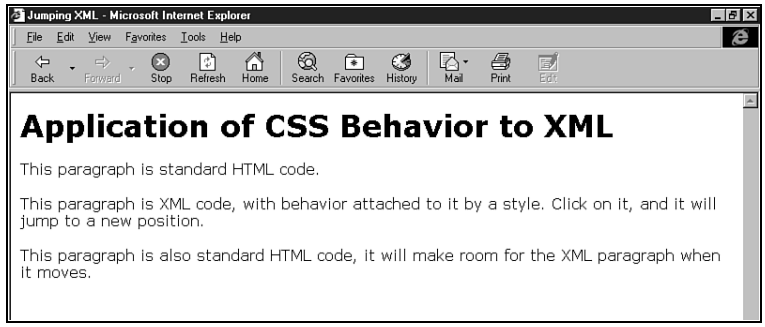


Abbildung 21.11: Der dynamische XML-Abschnitt im Internet Explorer 5, nachdem er angeklickt wurde.

Das Einbetten des Verhaltens funktioniert, wie Sie den Vorher-/Nachher-Abbildungen (21.10 und 21.11) entnehmen können. Es ist jedoch keine vollständige Lösung, weil der Code sich immer noch in dem HTML/XML-Dokument befindet.

Microsofts und auch Netscapes Lösung für das Importieren extern definierten Verhaltens ist die Verwendung des existierenden CSS-Stylesheet-Mechanismus. Microsoft hat die CSS-Syntax jedoch etwas erweitert und eine neue `behavior`-Eigenschaft eingeführt. Ich habe Listing 21.13 so umgeschrieben, daß dieses Verhalten extern zur Verfügung gestellt werden kann (siehe Listing 21.14).

Listing 21.14: Eine HTML-Datei mit dynamischem Verhalten

```
1:  <html>
2:  <head>
3:  <title>Jumping XML</title>
4:  <XML:namespace prefix="IE5"/>
5:  <style>
6:  IE5\:jump {style="font-color: red"; behavior: url(jump.sct);}
7:  </style>
8:  </head>
9:  <body>
10: <h1>Application of CSS Behavior to XML</h1>
11: <P>This paragraph is standard HTML code.</p>
12: <IE5:jump>This paragraph is XML code, with behavior
13: attached to it by a style.
```

```
14: style. Click on it, and it will jump
15:  to a new position.</IE5:getsclass>
16: <P>This paragraph is also standard HTML code,
17:  it will make room for the XML
18: paragraph when it moves.</p></body>
19: </html>
```

Listing 21.14 verhält sich genau wie Listing 21.13 (ich muß den Verhaltenscode natürlich noch in die Datei schreiben, auf die die URL im `behavior`-Attribut verweist). Beachten Sie, daß ich in Listing 21.14 ein eingebettetes Stylesheet verwendet habe (mit Hilfe des `STYLE`-Tags), das ich aber auch ganz einfach in ein externes Stylesheet hätte verschieben können.

Microsoft scheint große Pläne in Hinblick auf das Verhalten zu haben. Unter anderem geht es dabei um die Möglichkeit, nicht nur Skripten zu verwenden, um Elementen ein Verhalten zuzuweisen, sondern auch ActiveX-Steuerelemente. In der Praxis könnte das etwa denselben Effekt haben wie einer Webseite zu erlauben, ein normales Programm auf Ihrem Computer auszuführen. Damit eröffnen sich aufregende Möglichkeiten für Webseiten (Sie könnten sich beispielsweise interaktiv zeigen lassen, wie bestimmte Funktionen Ihrer Software arbeiten – und das auf Ihrem eigenen Bildschirm), aber auch hier tauchen sofort Fragen in Hinblick auf die Sicherheit auf. Egal was passiert – es wird sicher interessant sein.

Microsoft Chrome

Für ein Unternehmen, das das Internet so spät entdeckt hat, hat Microsoft sehr schnell aufgeholt. Es gibt Bereiche, wo sie die unbestrittenen Vorreiter sind. Chrome ist einer dieser Bereiche. Leider stellt Chrome auch Bedingungen – man braucht dafür einen 350-MHz-Pentium-Computer mit mindestens 64 Mbyte RAM, einer Busgeschwindigkeit von 100 MHz, einer AGP-Grafikkarte und Windows 98.

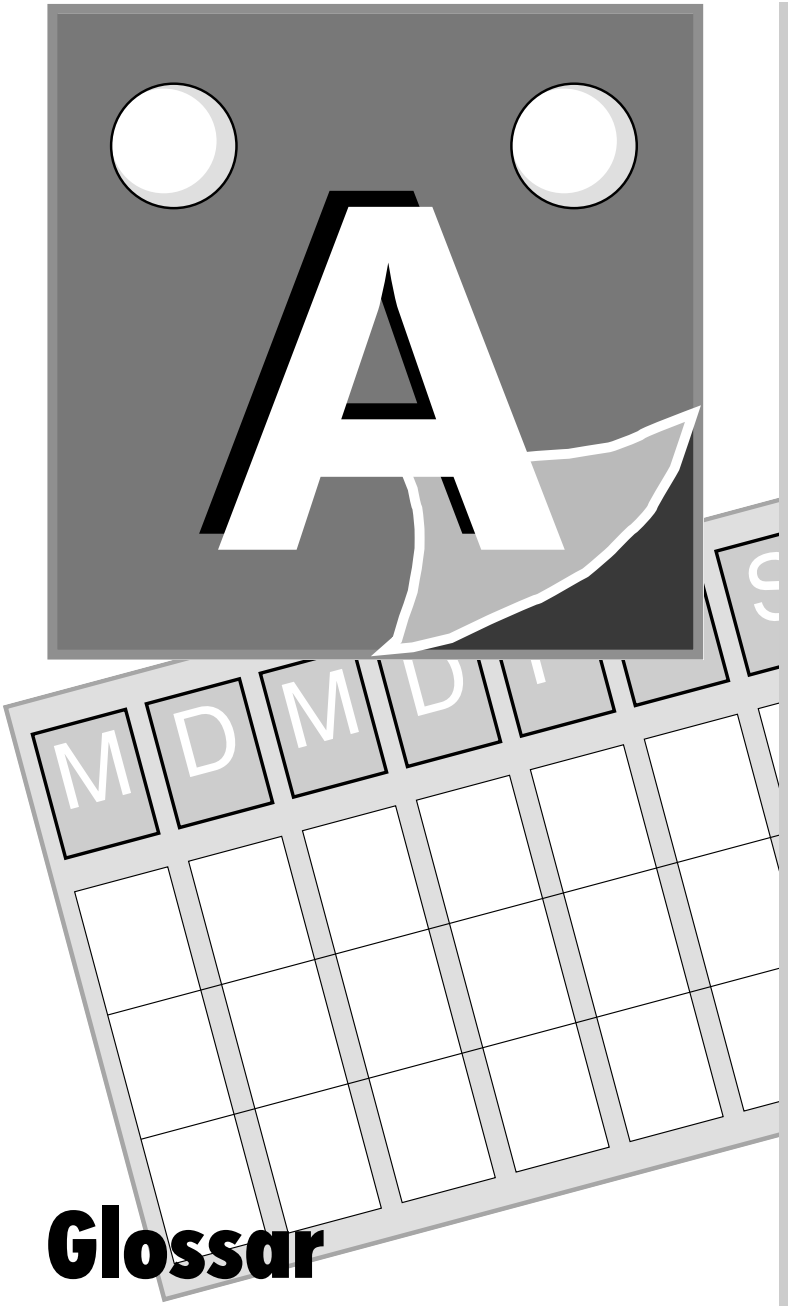
Chrome ist eine XML-Anwendung, mit der Sie interaktive 2D- und 3D-Animationen erzeugen können. Ich weiß, daß das auch schon mit VRML möglich war, und ich habe auch bereits erwähnt, daß es noch einen entsprechenden Ansatz gibt (VXML). Stellen Sie sich einen dreidimensionalen Würfel im Fenster Ihres Web-Browsers vor. Sie können den Würfel mit Hilfe der Maus nach links und rechts, nach oben und unten drehen. Sie können eine der Oberflächen vergrößern, indem Sie das Mausrad drehen. Jede der Würfelseiten zeigt eine Webseite an – als perfekte Miniatur: Text, Grafik usw. (Chrome erlaubt Ihnen, beliebige Oberflächenstrukturen anzuwenden, und eine solche Oberflächenstruktur kann auch eine HTML-Webseite sein).

Drehen Sie sich einfach eine interessante Seite nach vorne, und klicken Sie darauf. Der Würfel wird aufgeklappt und zeigt sechs weitere Seiten an, die als Oberflächenstruktur auf die Innenseiten des Würfels aufgetragen wurden.

Es gibt zwei wichtige Herausforderungen in Chrome: die technische (wie man es zum Funktionieren bringt) und die praktische (wie um alles in der Welt wir es nutzen sollten). Es gibt Millionen Fragen, und es wird noch einige Zeit vergehen, bis wird diese neue Form der Interaktion mit dem Benutzer verstehen werden.

Zusammenfassung

In diesem Kapitel habe ich Sie von ganz normalen Aufgabenstellungen wie beispielsweise der Anzeige mathematischer Gleichungen hin zu den Science-Fiction-Vorstellungen interaktiver 3D-XML-Darstellungen geführt. Dabei haben Sie Informationen über Grafikanwendungen erhalten und wie XML Lösungen für sehr alte Probleme schafft – und wie XML und die Implementierung von Verhalten einige der neueren Probleme der Wiederverwendung und Weitergabe von Code lösen können.



Glossar

Anwendung	Software, für die ein XML-Prozessor XML-Dokumente verarbeitet. Allgemeiner ausgedrückt, eine Nutzung des generischen XML-Rahmens für eine bestimmte Aufgabe, mit eigener DTD, Linking-Konventionen und Stylesheets.
Attribut	Ein Name/Wert-Paar, das einem Element zugeordnet wird und weitere Informationen über den Inhalt dieses Elements bereitstellt. Attributwerte können im Start-Tag des Elements angegeben werden. Standardwerte können von der DTD geerbt werden.
Attributdeklaration	Eine Deklaration in einer DTD, die den Namen, den Typ und gegebenenfalls den Typ eines bestimmten Elements angibt.
Bedingter Abschnitt	In der DTD ein Markup-Abschnitt, der in die logische Struktur der DTD aufgenommen oder daraus ausgeschlossen werden kann, abhängig von dem Schlüsselwort am Anfang.
CDATA-Abschnitt	Ein Teil eines XML-Dokuments, in dem Markup (außer dem Ende-Kennzeichner des CDATA-Abschnitts) nicht als Markup interpretiert, sondern unverändert an die Anwendung weitergegeben wird.
Codierungsdeklaration	Eine Deklaration des Zeichencodierungsschemas für eine bestimmte Texteinheit.
CSS (Cascading Style Sheets)	Eine Stylesheet-Sprache für die Darstellung von HTML-Dokumenten.
Dokument	Siehe XML-Dokument.
Dokumentelement	Ein Element, das alle anderen Elemente und Zeichendaten enthält, aus denen sich ein XML-Dokument zusammensetzt; auch als Wurzelelement bezeichnet.
Dokumenttypdefinition	Siehe *DTD*.
Dokumenttypdeklaration	Eine Deklaration am Anfang eines XML-Dokuments, das angibt, wo die externe DTD-Untermenge zu finden ist, und die die interne DTD-Untermenge enthält.

DSSSL	Document Style Semantics and Specification Language. Ein internationaler Standard (ISO/IEC 10179:1996), der eine Transformationssprache und eine Stilsprache für die Verarbeitung gültiger SGML-Dokumente definiert.
DSSSL-o	Eine Untermenge von DSSSL, das sogenannte Online-Profil, das als reduzierte Version von DSSSL für die Darstellung von XML-Dokumenten vorgeschlagen wurde.
DTD	Regeln für die Elementtypen, die innerhalb eines XML-Dokuments erlaubt sind. Sie bestimmen den Inhalt und die Attribute für diese Elementtypen. Die DTD deklariert außerdem alle externen Entities, auf die innerhalb des Dokuments verwiesen wird, ebenso wie die möglichen Notationen. Siehe auch *Externe DTD-Untermenge*.
Einfacher Link	Ein Inline-Link, wie beispielsweise das vertraute `<A HREF="">`-Tag in HTML, der einen bestimmten Punkt in einem XML-Dokument mit einem Ziel verbindet.
Element	Eine logische Informationseinheit in einem XML-Dokument.
Elementinhalt	In einer DTD bestimmt ein Inhaltsmodell, daß in einem bestimmten Elementtyp nur bestimmte andere Elemente enthalten sein dürfen. (Siehe auch *Gemischter Inhalt*.)
Elementkonstruktionsregel	Eine Anweisung in einem XSL-Stylesheet, die angibt, wie Flußobjekte konstruiert werden sollen, wenn ein bestimmter Elementtyp erkannt wird.
Elementtyp	Ein bestimmter Elementtyp, beispielsweise ein Absatz. Ein Elementtyp wird durch den im Start- und Ende-Tag angegebenen Namen gekennzeichnet.
Ende-Tag	Ein Tag, das das Ende eines Elements kennzeichnet, beispielsweise `</section>`.
Entity	Beliebige Daten, die als Objekt behandelt werden können, wie beispielsweise eine externe Datei, die ein Bild enthält.

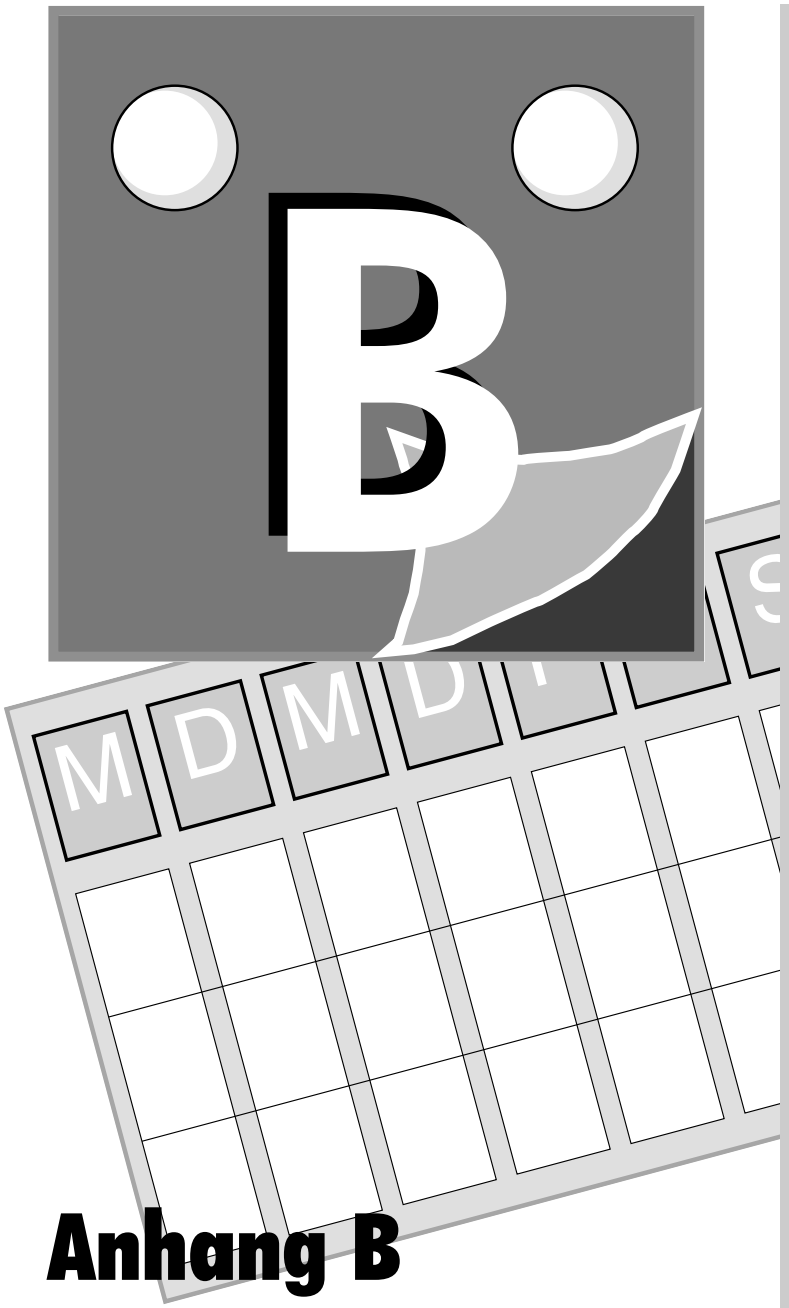
Entity-Deklaration	Teil der DTD. Eine Entity-Deklaration deklariert einen Namen für ein Entity und weist ihm einen Ersatzstring oder extern gespeicherte, durch eine URL identifizierte Daten zu.
Entity-Verweis	Ein Verweis innerhalb des Texts eines XML-Dokuments auf ein zuvor deklariertes Entity, wodurch festgelegt wird, daß der Inhalt des Entity an dieser Stelle verarbeitet werden soll (Verarbeitung bedeutet in diesem Fall auch einfaches Einbinden).
Erweiterter Link	Ein Link, der mehrere Ressourcen beinhalten kann. Ein erweiterter Link muß sich nicht unbedingt innerhalb einer der am Link beteiligten Ressourcen befinden.
Externe DTD-Untermenge	Der Teil der DTD, der in einer durch eine URL adressierten, separaten Ressource abgelegt ist. Die externe DTD-Untermenge wird häufig auch als die DTD einer Dokumentenklasse bezeichnet. Siehe auch *DTD*.
Externes Entity	Ein Entity, dessen Inhalt in einer externen XML-Ressource enthalten ist, beispielsweise in einer Grafikdatei, auf die in einem XML-Dokument verwiesen wird.
Flußobjekt	Eine Formatfunktion, wie beispielsweise ein Absatz oder eine Tabellenzelle, in die der Inhalt eines XML-Dokuments unter der Kontrolle eines XSL-Stylesheet eingetragen wird.
Flußobjektbaum	Die Menge aller Flußobjekte, in die ein XML-Dokument durch ein XSL-Stylesheet umgewandelt wird.
Gemischter Inhalt	In der DTD ein Inhaltsmodell, das erlaubt, Zeichendaten mit untergeordneten Elementen in beliebiger Reihenfolge zu kombinieren (siehe auch *Elementinhalt*).
Generischer Bezeichner	Der einem Elementtyp zugeordnete Bezeichner.
Gültiges XML-Dokument	Ein wohlgeformtes XML-Dokument, dessen Inhalt der in seiner DTD spezifizierten Struktur entspricht.

HTML	Hypertext Markup Language. Ein Codierungsschema für die Anzeige und das Hyperlinking von Informationsseiten im World Wide Web. Einige HTML-Versionen sind SGML-Anwendungen.
HyTime	ISO/IEC 10744 Hypermedia/Time-based Structuring Language. Eine SGML-Anwendung, die die SGML-Fähigkeiten durch Dinge wie Multimedia-Unterstützung und komplexe Adressierungsmechanismen erweitert.
Inhaltsmodell	In der DTD eine Beschreibung dessen, was innerhalb von Instanzen eines bestimmten Elementtyps passieren könnte.
Inline Link	Siehe einfacher Link.
Interne DTD-Untermenge	Der Teil der DTD, der innerhalb des eigentlichen XML-Dokuments vor dem ersten Start-Tag deklariert ist.
Internes Entity	Ein Entity, dessen Wert in seiner Entity-Deklaration in der DTD angegeben ist.
Kommentar	Markup in einem XML-Dokument, das Text enthält, der nicht als Teil des Dokuments behandelt werden soll.
Leeres Element	Ein Element, das keine anderen Elemente und keine Zeichendaten enthält.
Logische Struktur	Die Deklarationen, Elemente, Zeichenverweise, Verarbeitungsanweisungen usw., aus denen sich ein XML-Dokument zusammensetzt. Sie werden alle durch explizites Markup gekennzeichnet.
Lokator	Ein Zeichenstring, der das Ende eines Links angibt.
Markup	Information, die mit dem Text eines XML-Dokuments vermischt ist, um dessen logische und physische Struktur zu kennzeichnen.
Name	In einer XML-DTD ein Buchstabe oder ein Unterstrich, gefolgt von einem oder mehreren weiteren Namenszeichen.
Namensraum	Eine Methode, die in XML-Dokumenten verwendeten Namen zu konkretisieren, indem ihnen der durch URIs spezifizierte Kontext zugeordnet wird.

Namens-Token	Eine beliebige Kombination aus Namenszeichen.
Namenszeichen	Ein Buchstabe, eine Ziffer, ein Gedankenstrich, ein Unterstrich, ein Punkt oder eines von bestimmten im XML-Standard spezifizierten Sonderzeichen.
Nichtanalysierender XML-Prozessor	Ein XML-Prozessor, der XML-Dokumente auf ihre Wohlgeformtheit, nicht aber auf ihre Gültigkeit überprüft.
Notation	Das Format eines externen Entity, wie beispielsweise ein GIF-Bild oder ein MPEG-Video.
Out-of-line-Link	Ein Link, der nicht als eine seiner eigenen Ressourcen dient.
Parameter-Entity	Ein Text-Entity, das innerhalb einer DTD verwendet oder mit dem die Verarbeitung bedingter Abschnitte gesteuert wird.
Physische Struktur	Die Anordnung physischer Speichereinheiten (Entities), in denen ein XML-Dokument abgelegt ist.
Prolog	Der Teil eines XML-Dokuments, inklusive XML-Deklaration und DTD, der dem eigentlichen Dokumentelement vorausgeht.
Rendering	Die Verarbeitung des Dokuments, so daß es angezeigt werden kann. Dabei geht es in der Regel um die Darstellung auf dem Bildschirm; es könnte sich aber auch auf andere Verarbeitungen beziehen, beispielsweise auf eine Text/Sprach-Konvertierung für Sehbehinderte.
Ressource	Eine adressierbare Informationseinheit, die Teil eines Links sein kann. Eine Ressource kann vollständige XML-Dokumente, Elemente (oder Elementbereiche) darin umfassen, ebenso wie einfache Textabschnitte.
SGML	Standard Generalized Markup Language. Ein internationaler Standard (ISO8879:1986), der ein verallgemeinertes Markup-Schema für die Darstellung der logischen Struktur von Dokumenten auf system- und plattformunabhängige Weise erlaubt.

Start-Tag	Ein Tag, das den Anfang eines Elements markiert, beispielsweise <para>.
Stylesheet	Mehrere Anweisungen, die angeben, wie die Elemente in einem Dokument formatiert werden sollen.
Verarbeitungsanweisung (Processing Instruction, PI)	Ein Markup, das Informationen oder Anweisungen an Software weitergibt, die ein XML-Dokument verarbeiten soll. Eine Verarbeitungsanweisung ist nicht Teil der Zeichendaten eines Dokuments.
W3C	World Wide Web Consortium. Eine Gruppe aus Anbieter-Unternehmen, die als standardschaffendes Gremium für das Web dient.
Web	Ein Internet-Dienst, der das HTTP-Protokoll und das HTML-Format verwendet, um Dokumente weiterzugeben.
Wohlgeformtes XML-Dokument	Ein XML-Dokument, das aus einem einzigen Wurzelelement besteht, das korrekt verschachtelte untergeordnete Elemente enthält. Alle Entity-Verweise innerhalb eines wohlgeformten XML-Dokuments beziehen sich auf Entities, die in der DTD deklariert wurden oder eine der Standard-Entities darstellen, und alle Attributwerte sind in Anführungszeichen eingeschlossen. Ein wohlgeformtes XML-Dokument gehorcht außerdem einigen weiteren Regeln.
Wurzelelement	Siehe Dokumentelement.
XML	Extensible Markup Language. Ein Profile, eine vereinfachte Untermenge von SGML.
XML-Deklaration	Eine Verarbeitungsanweisung am Anfang eines XML-Dokuments, die dieses als XML-Code deklariert.
XML-Dokument	Ein Dokument, das aus einer optionalen XML-Deklaration besteht, gefolgt von einer optionalen Dokumenttypdeklaration, gefolgt von einem Dokumentelement.

XML-Prozessor	Ein Programm, das XML-Dokumente liest, sie auf Gültigkeit und Wohlgeformtheit überprüft und ihren Inhalt XML-Anwendungen bereitstellt.
XPointer	Eine Syntax zur Kennzeichnung eines Elements, eines Elementbereichs oder eines Texts innerhalb eines XML-Dokuments, die Zielressourcen eines Links sind.
XSL	Die XML-Stilsprache.
Zeichen	Eine atomare Texteinheit, dargestellt durch einen Bit-String.
Zeichendaten	Der eigentliche Text in einem XML-Dokument, im Gegensatz zum Markup für das Dokument.
Zeichenverweis	Ein Escape-Code für ein einzelnes Unicode-Zeichen, der den numerischen Wert des entsprechenden Bit-Strings angibt.



Anhang B

XML-Ressourcen

XML hat hauptsächlich mit der Informationsweitergabe über das Internet zu tun; es sollte also nicht überraschen, daß man die meisten Informationen über XML ebenfalls im World Wide Web findet. Ich habe viele Stunden im Web verbracht, um die folgenden Auflistungen zusammenzustellen, aber es kann natürlich sein, daß ich etwas vergessen habe. Ich habe versucht, Adressen von Web-Sites aufzunehmen, die Verweise auf andere Sites beinhalten, um so vollständig wie möglich zu sein und zu vermeiden, Sie auf Sites zu verweisen, die zum Zeitpunkt der Drucklegung dieses Buchs bereits veraltet sind.

Online-Ressourcen

Wenn Sie sich nach dem Lesen dieses Anhangs ein paar Adressen gemerkt haben, dann hoffentlich die folgenden. Es handelt sich dabei um die wahrscheinlich vollständigsten Referenz-Sites für fast alles, was mit XML zu tun hat:

- *The SGML Bibliography* – Robin Cover hat seit 1986 bibliographische Daten über SGML und verwandte Themen gesammelt. Robin behauptet, daß die Online-Listings nur ein Teil der Datenbank sind, aber dennoch scheinen sie allen Druckmedien überlegen zu sein. Sie erreichen diese Site unter `http://www.oasis-open.org/sgml/biblio.html`.
- *The XML site* – Eine der definitiven Sites für XML-Information, die Tim Brays ausgezeichnete kommentierte Version der XML-Syntax-Empfehlungen beinhaltet: `http://www.xml.com`.
- *The Whirlwind Guide to SGML Tools and Vendors* – Steve Pepper sammelt seit 1992 Informationen, als das noch nicht so zeitaufwendig war. Der schnell expandierende Markt hat ihn gezwungen, etwas selektiver mit seinen Einträgen zu sein, aber es handelt sich immer noch um eine der ultimativen Quellen, die Sie hier finden: `http://www.infotek.no/sgmltool/guide.htm`.

Anwendungen

Web-Sites mit Beschreibungen der XML-Anwendungen, die es zum Zeitpunkt der Drucklegung dieses Buchs gibt. Es gibt ständig neue Anwendungen, was kaum überrascht, wenn Sie diese kurze Liste einsehen und erkennen, welch unglaublich umfassenden Anwendungsbereich sie abdeckt.

- Bioinformatic Sequence Markup Language (BSML) – Beschäftigt sich mit der Codierung und dem Austausch genetischer Informationen unter Verwendung von XML: `http://www.topogen.com/sbir/rfc.html`.
- CDIF Home Page – Das Computer Aided Software & Systems Engineering (CASE) Data Interchange Format soll den Austausch von Informationen über die Softwareentwicklung zwischen CASE-Software-Werkzeugen vereinfachen: `http://www.cdif.org/`.
- CHIP Info Production's Hypertext Management System (HTM-S) – Eine kommerzielle Initiative zur Verwendung von XML für die Verwaltung von Web-Sites: `http://www.chipinfo.com/products/htms.htm`.
- Chrome – Zwei- und dreidimensionale XML-Modellierungstechnologie von Microsoft: `http://www.microsoft.com/workshop/imedia/chromeffects/down_default.asp`.
- Conceptual Knowledge Markup Language DTD – XML für Anwendungen der künstlichen Intelligenz: `http://wave.eecs.wsu.edu/ WAVE/Ontologies/CKML/CKML-DTD.html`.
- Development Markup Language (DML) – Eine XML-Anwendung für den Informationsaustausch zwischen Organisationen zur Entwicklungshilfe: `http://resources. bellanet.org/xml/index.cfm`.
- Document Interchange Initiative – Eine XML-Anwendung für den elektronischen Dokumentenaustausch: `http://interaction.in-progress.com/interchange/index`.
- Extensible Log Format Initiative – XML zur Codierung von Web-Server-Protokollen zur einfacheren Verarbeitung und Abfrage: `http://www.docuverse.com/xlf/`.
- FlixML Home Page – Nicht ganz ernsthafter Versuch von XML-Autor John E. Simpson, seine Datenbank der B-Movie-Reviews in XML darzustellen: `http://www.flixml.org/`.
- Genealogical Data in XML (GedML) – Versuch, Stammbäume mit XML einfacher zu verfolgen und aufzuzeichnen: `http://home. iclweb.com/icl2/mhkay/gedml.html`.

- Handheld Device Markup Language (HDML) – Miniversion von HTML in XML, mit der PDAs und Handys für E-Mail- und Web-Anwendungen genutzt werden können: http://www.uplanet.com/pub/hdml_w3c/.
- HL7 SGML SIG – Großinitiative zur Darstellung von Gesundheitsinformationen (Patienten, Medikamente, Behandlung, Versicherung, Finanzierung) in XML: http://www.mcis.duke.edu/standards/HL7/committees/sgml/.
- Information & Content Exchange (ICE) – Ein Versuch, eine sinnvollere Beschreibung von Webseiteninhalten bereitzustellen: http://www.vignette.com/.
- Internationalization and Localization Tools (ILE) – XML für die Lokalisierung von Software und Software-Dokumentationen: http://www.ile.com/tools/.
- Instructional Management System (IMS) Meta-data – Initiative zur Verwendung von XML zur Unterstützung Internet-basierten Lernens: http://www.imsproject.org/md_overview.html.
- Java Speech Markup Language – Suns Initiative, XML zur Codierung von Voice-Daten zu verwenden: http://java.sun.com/products/java-media/speech/forDevelopers/JSML/.
- MathML – Eine DTD-Anwendung (endlich!), mit deren Hilfe Mathematik in XML ausgezeichnet wird: http://www.w3c.org und http://www.nag.co.uk/projects/OpenMath/mml-files.
- Motorola VoxML – Motorolas XML-Anwendung zur Entwicklung von Voice-Anwendungen: http://VoxML.mot.com/.
- The News Industry Text Format Initiative (NITF) – Anwendung von XML zur Übertragung von Nachrichten: http://www.iptc.org/iptc/ orddocs.htm#nitf.
- NuDoc Technology Brief – Bitstreams XML-Anwendung zur Codierung von Dokumentschriftinformation: http://www.bitstream.com/nudoc/nudoctb.html.
- OpenFilter – Methode zum Austausch von Filterinformationen: http://www.ile.com/tools/openfilter.htm.
- Open Financial Exchange Specification – Microsofts XML-Anwendung zum Austausch von Informationen über Finanztransaktionen: http://www.microsoft.com/finserv/ofxdnld.htm.
- Open Software Description (OSD) – XML-Anwendung, die es ermöglichen soll, Software über das Internet zu veröffentlichen: http://www. microsoft.com/standards/osd/.
- OpenTag – Eine XML-Anwendung zum Austausch von Informationen aus Übersetzungsdatenbanken: http://www.opentag.org/.

- Open Trading Protocol – XML für den Austausch von Handelsinformationen: http://www.otp.org:8080/.
- OMG TC Work in Progress – Die Object Management Group (OMG) ist eine der führenden Organisationen im Bereich komponentenbasierter Software. Dort verwendet man XML, um verteilte Software-Komponenten zu implementieren: http://www.omg.org/library/schedule/Stream-based_Model_Interchange.htm.
- Process Interchange Format – XML – Ein Versuch, XML zur Codierung von Handelsprozeßinformationen zu nutzen: http://www.xmls.com/pif/.
- Python/XML How-to – Die Anwendung der Skripting-Sprache Pyhton zur Verarbeitung von XML-Daten: http://www.python.org/doc/ howto/xml/.
- Q&A Markup Language (QAML) – Ein erster Ansatz, mit XML FAQ-Dokumente im Internet einfacher anzulegen und zu pflegen: http://www.faq.org/qaml/.
- RDF Made (Fairly) Easy – Ein ausgezeichneter Versuch, das RDF (Resource Description Format) sinnvoll zu beschreiben: http://www.ccil.org/~cowan/XML/RDF-made-easy.html.
- Schema for Object-Oriented XML – Eines der vielen möglichen XML-Schemata, das gerne Nachfolger der DTD wäre: http://www.w3.org/TR/NOTE-SOX/.
- Software Component Interface Description in SGML – Ein akademisches Projekt für die Anwendung von XML auf die Beschreibung der Schnittstellen von Softwarekomponenten: http://www.cgl.uwaterloo.ca/Projects/ meta/sgml97/mmccool/index.html.
- TCIF Information Product Interchange Committee – Kommerzielle Initiative zur Verwendung von XML für den Austausch von Produktinformationen: http://www.atis.org/atis/tcif/ipi/5tc60hom.htm.
- VXML (jetzt 3DXML) – Ein Versuch, mit XML VRML-Informationen zu veröffentlichen: http://www.ultrablue.com/dan/vxml/printable.html.
- X-ACT – ActiveX Content Technologies' Ansatz, XML für Geschäftsaktivitäten einzusetzen: http://www.x-act.org/.
- XML/EDI Group's Home Page – SGML wurde bereits längere Zeit für EDI (Electronic Data Interchange) benutzt, und diese Initiative hat dasselbe mit XML vor: http://www.geocities.com/WallStreet/Floor/5815/.
- XML File System – Vorschlag, mit XML die Organsiation von Computerdateisystemen zu beschreiben: http://www.gefen.co.il/xml/whole.html.
- Extended Forms Description Language (XFDL) – Eine XML-Anwendung, die HTMLs beschränkte Fähigkeiten zur Formularverarbeitung erweitern soll: http://www.uwi.com/xfdl/.

- XML-Based Components – Coins – Eine Initiative, XML zur Verwaltung von Java-Komponenten zu nutzen: http://www.jxml.com/ coins/presentations/Coins980616/index.htm.
- XML-QL: A Query Language for XML – Eine Initiative, die Funktionalität von SQL in XML einzubauen: http://www.w3.org/TR/NOTE-xml-ql/.

Standards

ISO-Standards sind durch Copyright geschützte Dokumente und stellen eine wesentliche Einkommensquelle für die ISO dar; deshalb wird es selten Kopien davon im Public Domain-Bereich geben. Entwürfe einiger der ISO- und Nicht-ISO-Standards stehen im Internet zur Verfügung. Unter anderem finden Sie dort Informationen über die folgenden wichtigen Standards und andere halboffizielle Dokumente:

- Submissions to W3C – Alle neuen Vorschläge an das W3C: http://www.w3.org/pub/WWW/Submission/.
- Extensible Markup Language Proposal, Version 1.0 – Die definitive W3C-Quelle: http://www.w3.org/pub/WWW/TR/PR-xml.
- Action Sheets – http://www.w3.org/TR/NOTE-AS.
- Cascading Style Sheets, Level 1 – http://www.w3.org/pub/WWW/TR/PR-CSS1.
- Cascading Style Sheets, Level 2 – http://www.w3.org/TR/REC-CSS2.
- Vergleich von SGML und XML – http://www.w3.org/TR/NOTE-sgml-xml.html.
- Entwurfsskizze zum CDF – http://www.microsoft. com/standards/cdf.htm.
- Document Content Description for XML – http://www.w3.org/TR/NOTE-dcd.
- Document Object Model Specification – http://www.w3.org/TR/WD-DOM/.
- The Annotated XML Specification – http://xml. com/axml/axml.html.
- HDML Language Specification – http://www.uplanet.com/pub/ hdml_w3c/hdml20-1.html.
- User's Guide to ISO/IEC 15445:1998 HTML (in Arbeit) – http://www.ornl.gov/sgml/wg8/document/n1966/UG.html.
- HTML 3.2 Reference Specification – http://www.w3.org/pub/WWW/TR/WD-html.
- HTML 4.0 Specification – http://www.w3.org/TR/WD-html40/.
- Hyper Graphics Markup Language (HGML) – http://www.w3.org/TR/NOTE-HGML.

- Mathematical Markup Language (MathML) – http://www.w3.org/TR/REC-MathML.
- Meta Content Framework Using XML – http://www.textuality.com/sgml-erb/w3c-mcf.html.
- Namespaces in XML – http://www.w3.org/TR/WD-xml-names.
- Open Software Description Specification – http://www.w3.org/TR/NOTE-OSD.html.
- Open Financial Exchange Specification – http: //www.microsoft.com/finserv/ofxdnld.htm.
- P3P (Platform for Privacy Preferences) – http://www.w3c.org/TR/WD-P3P10-syntax.
- PICS-NG Metadata Model and Label Syntax (Platform for Internet Content Selection) – http://207.201.154.232/murray/specs/WD-pics-ng-metadata-970514.html.
- Precision Graphics Markup Language (PGML) – http://www.w3.org/TR/ 1998/NOTE-PGML.
- Resource Description Framework (RDF) Model and Syntax – http://www.w3.org/TR/WD-rdf-syntax/.
- SDML – Signed Document Markup Language – Version 2.0 – http://www.w3.org/ TR/NOTE-SDML/.
- Synchronized Multimedia Integration Language – http://www.w3.org/TR/WD-smil.
- Translation Memory Exchange (TMX) Format Specifications – http://www. lisa.org/tmx/tmx.htm.
- Unicode Home Page – http://www.unicode.org/.
- WebBroker – Distributed Object Communication on the Web -http://www.w3.org/TR/1998/NOTE-webbroker-19980511/.
- Schematic Graphics – http://www.w3.org/TR/1998/NOTE-WebSchematics/.
- WebCGM Profile – http://www.w3.org/TR/NOTE-WebCGM/.
- Web Interface Definition Language (WIDL) – http://www.oasis-open.org/sgml/xml.html#widl.
- Web Collections using XML Submission – http://www.w3.org/pub/WWW/TR/NOTE-XMLsubmit.html.

- XAPI-J – Standardized XML API in Java – http://www.datachannel.com/channelworld/xml/dev/.
- XML-Arch – http://home. sprynet.com/sprynet/dmeggins/xml-arch.html.
- XML in HTML Meeting Report – http://www.w3c.org/TR/NOTE-xh.
- XML Object Model – http://www.microsoft.com/msdn/sdk/inetsdk/ help/inet5017.htm.
- XML-Data Spec – http://www.microsoft.com/standards/xml/xmldata.htm.
- XML Linking Language (XLink) – http://www.w3.org/TR/NOTE-xlink.
- XML Linking Language (XLink) Design Principles – http://www.w3.org/ TR/NOTE-xlink-principles.
- XML Pointer Language (XPointer) – http://www.w3.org/TR/WD-xptr.
- A Proposal for XSL – http://www.w3.org/TR/NOTE-XSL.html.
- HyTime Working Group FTP Archive – offizielle Site der Workgroup für den Austausch von Dateien: ftp://infosrv1.ctd.ornl.gov/ pub/sgml/WG8/HyTime/TC/.
- HyTime: ISO 10744 Hypermedia/Time-based Structuring Language – Öffentliche Kopie des offiziellen Standards: http://dmsl.cs.uml.edu/ standards/hytime.html.
- ISO/IEC 10744 HyTime (Second Edition) – Inoffiziell, aber so gut wie definitive Kopie der letzten Überarbeitung des Standards von Eliot Kimber (Dr. Macro): http://www.drmacro.com/hythtml/is10744r.html.
- Unicode Home Page – Offizielle Unicode-Site: http://www.unicode.org/.

Informationsquellen

Die folgenden Sites beinhalten allgemeine Informationen über SGML und XML:

- SGML and XML Resources – Links und Verweise, die Ihnen sehr nützlich sein können: http://www.arbortext.com/linksgml.html.
- What Is XML? – Die Graphical Communications Association (GCA) ist eine der Hauptorganisationen von XML- und SGML-Konferenzen in der Welt. Hier finden Sie einige praktische Verweise: http://www.gca.org/conf/xml/ xml_what.htm.
- Commonly Asked Questions About the Extensible Markup Language – Wird offiziell für die W3C unterhalten. Hier finden Sie Antworten auf viele Fragen, die in den offiziellen Dokumenten nicht beantwortet werden: http://www.ucc.ie/xml/.

- XML (W3C Site) – Mehr oder weniger das Zentrum aller XML-Aktivität: `http://www.w3.org/pub/WWW/XML/`.
- XML (Robin Cover's Site) – Aladdins Schatzkiste praktischer XML-Informationen: `http://www.oasis-open.org/sgml/ xml.html`.
- XML: The Extensible Markup Language (James K. Tauber's Site) – Wichtige Informationsquelle: `http://www.jtauber.com/xml/`.

Softwarepakete

Die folgenden Softwarepakete unterstützen XML:

- AElfred XML Parser: `http://www.microstar.com/XML/index.htm`
- Agave – SQml-Publisher: `http://www.agave.com/html/products/sqml_pub.htm`
- AgentSoft – XML Technology Demonstration: `http://www.agentsoft.com/xml/`
- Balise Home Page: `http://www.balise.berger-levrault.fr/current/index.htm`
- BCC XML Tools: `http://www.mygale.org/07/jcalles/XML/`
- Bjondi (Unicode Utilities): `http://www.bjondi.com/`
- Channel Maker – Download: `http://www.anyware.co.uk/anyware/cm/download.html`
- Chris Stevenson's Java Applets: `http://www.users.on.net/zhcchz/java.html`
- Chrystal Software Astoria & XML Bulletin: `http://www.chrystal.com/xml.htm`
- Clip XML Editor: `http://xml.t2000.co.kr/product/clip.html`
- Copernican Solutions Incorporated – XML Developer's Toolkit (XDK): `http://www.copsol.com/products/xdk/index.html`
- Crane Softwrights Ltd. – Shareware: `http://www.cranesoftwrights.com/shareware/`
- CSSize v0.9b1, Add CSS Attributes to HTML and XML Code: `http://lara0.exp.edf.fr/glazman/CSSize/cssize.en.htm`
- DataChannel – Datachannel RIO & WebBroker: `http://www.datachannel.com/`
- DTD2HTML, Perl Tools for Making DTDs Navigable: `http://www.ogc.uci.edu/indiv/ehood/PerlSGML.html`
- XML Import Filter for Lotus Domino: `http://www.digitome.com/download.htm`
- Docproc: `http://javalab.uoregon.edu/ser/software/docproc/`

- DSSSLTK, DSSSL toolkit: http://www.copsol.com/products/index.html
- Excosoft – Documentor XML/SGML editor: http://www.excosoft.se/welcome/home_welcome.html
- Expat (XML parser used in Mozilla): http://www.jclark.com/xml/expat.html
- EzDTD DTD editor: http://www.download.com (search for »ezdtd«)
- Free XML tools: http://birk105.studby.uio.no/www_work/xmltools/index.html
- General Picture's CiaoXML Demo: http://www.generalpicture.com/
- GRIF S.A. – SGML and HTML Solutions: http://www.grif.fr/
- GroveWare Home: http://www.groveware.com/
- HEX – The HTML Enabled XML Parser: http://www-uk.hpl.hp.com/people/ak/java/hex.html
- HXA – Hubick's XML Analyzer: http://www.hubick.com/software/HXA/
- HyBrick (DSSSL viewer): http://collie.fujitsu.com/hybrick
- Hypermedic – XML Tools: http://www.hypermedic.com/style/tools/tools.htm
- IRIS XML Editor & DTD Generator: http://www.cabinfo.com/download.htm
- Jade – DSSSL Engine: http://www.jclark.com/jade
- Scientific Information Components Using Java/XML: http://ala.vsms.nottingham.ac.uk/vsms/java/jumbo/ index.html
- Junglee Technology – Virtual Databases: http://wpreal.junglee.com/tech/index.html
- Java XML (jxml) Home Page: http://www.jxml.com/
- Lark – Non-validating parser: http://www.textuality.com/Lark/
- LTG Software – LT XML Tools and Developers' API: http://www.ltg.ed.ac.uk/software/xml/
- Majix (TetraSix) – MS-Word to XML Converter: http://www.tetrasys.fr/ProduitsFrame.htm
- The Microsoft XSL Processor: http://www.microsoft.com/xml/xsl/msxsl.htm
- Microsoft XML Parse in Java: http://www.microsoft.com/standards/xml/xmlparse.htm

- Microstar – Near&Far Graphic DTD Development Tool: http://www.microstar.com/Products-And-Technologies/index.html
- MONDO – Modeling Technologies: http://www.chimu.com/projects/mondo/2
- NXP XML Parser: http://www.edu.uni-klu.ac.at/~nmikula/NXP
- Object Design, Inc. – XML Downloads: http://www.odi.com/content/products/pse/XMLDownload.htm
- OmniMark Konstructor: http://www.techmall.com/techdocs/NP980330-1.html
- OVIDIUS MetaMorphosis SGML/XML Processor: http://www.ovidius.com/efs-mm.html
- QORX – Express and SQL Query as an XML Document: http://www.griffin-brown.co.uk/tools.shtml
- Roustabout, QuarkXPress-to-XML/SGML Translator: http://www.attd.com/xml/products.html#roustabout
- SAX – The Simple API for XML: http://www.megginson.com/SAX/
- SAX – DOM Bridge: http://www.docuverse.com/personal/saxdom.html
- SAXON – A Java class library for XML Applications: http://home.iclweb.com/icl2/mhkay/saxon.html
- SCHEMA.NET – For DTDs and Other XML Schemata: http://www.schema.net/
- Sean Russell's XML Package: http://jersey.uoregon.edu/ser/software/
- Silfide – EN – SXP – Frames – Xsilphide client/server environment for distributing language resources: http://www.loria.fr/projets/XSilfide/EN/sxp/
- Stilo WebWriter XML Editor: http://www.pimc.com/WebWriter/download.html
- Symposia Web Browser/Editor: http://symposia.inria.fr/symposia/download-symposia.html
- Tcl Support for XML: http://tcltk.anu.edu.au/XML/
- UWI.Com – The Internet Forms Company: http://www.uwi.com/
- Visual XML: http://www.pierlou.com/visxml/
- WebMethods – Automate the Web: http://www.webmethods.com/home.html
- Web Publishers – Tools and Utilities Home Page: http://www.webpub.com/tools/home.html
- Woodstock XML Editor: http://www.vtopia.com/
- XML in Mozilla: http://www.mozilla.org/rdf/doc/xml.html

- XML Viewer Applet: http://capita.wustl.edu/xmlres/examples/MSJavaParser/XMLViewer.htm
- XML Application – Prototype: http://www.cam.org/~pierlou/prototype/
- XED – An XML Document Instance Editor: http://www.ltg.ed.ac.uk/~ht/xed.html
- XML::Parser: http://www.netheaven.com/~coopercc/xmlparser/intro.html
- XML Parser Component for Delphi: http://www.icom-dv.de/xml.htm
- XML Pro (Vervet Logic) XML editor: http://www.vervet.com/
- Xpublish (Mac): http://interaction.in-progress.com/xpublish/index
- XMLSOFTWARE.COM – The XML Software Site: http://www.xmlsoftware.com/
- XML Styler – Early XSL style sheet editor (veraltet): ftp://ftp.arbortext.com/pub/downloads/xmlstyler2c.exe
- XML for Java – Kostenlose IBM-Software zur Verarbeitung von XML-Daten: http://www.alphaworks.ibm.com/formula/xml
- XML Viewer Demo: http://208.204.84.117/XMLTreeViewer/demo.html7
- Xparse – XML parser in JavaScript: http://www.jeremie.com/Dev/XML/index.phtml
- Xpose XML Editor: http://www.intravenous.com/products/
- Xslj – An XSL to DSSSL Translator: http://www.ltg.ed.ac.uk/~ht/xslj.html
- XSLProcessor – The first package to support the revised XSL (XSL2) proposal: http://www.mygale.org/07/jcalles/XML/xslProcessor.html
- XT – Free XML parser that implements the tree construction part of the XSL draft: http://www.jclark.com/xml/xt.html
- Zydeco – An XML Development Environment: http://www.dn.net/zydeco/

Software-Unternehmen

Die hier aufgelisteten Firmen bieten bereits Softwarepakete mit XML-Unterstützung an oder planen deren Veröffentlichung in nächster Zukunft

- AIS Berger-Levrault – Produziert SGML- und XML-Verarbeitungs- und Datenbank-Software: http://www.balise.berger-levrault.fr/.
- ArborText Inc. – Produzieren SGML- und XML-Editoren, -Formatwerkzeuge und Konvertierungssoftware: http://www.arbortext.com/.

- Chrystal Software Inc. – Dokument- und Komponentenverwaltung: `http://www.chrystal.com/`.
- Copernican Solutions Incorporated – XML-, DSSSL- und SGML-Software: `http://www.copsol.com/`.
- Grif S.A. – SGML- und XML-Editoren, Browser und Format-Werkzeuge: `http://www.grif.fr/`.
- High Text – Topic Navigation Map-Experten und SGML- und HyTime-Berater: `http://www.hightext.com/`.
- Inso Corporation – Produzenten von SGML- und XML-Konvertierern und Browser-Software: `http://www.inso.com/`.
- Language Technology Group – Entwickler von DSSSL- und XML-Verarbeitungssoftware: `http://www.ltg.ed.ac.uk/`.
- Microsoft Corporation – Hersteller von XML- (und anderer) Software: `http://www.microsoft.com/`.
- SoftQuad – Produzenten von SGML-Editoren und Browsern: `http://www.sq.com/` (eine XML-Version Ihres HTML-Tools HoTMetaL HTML, XmetaL, wurde bereits auf mehreren Konferenzen vorgestellt und war für Anfang 1999 angekündigt).
- STILO Technology Ltd. – SGML-, HTML- und XML-Editoren: `http://www.stilo.com`.
- webMethods – WWW-Server-Software für XML: `http://www.webmethods.com/home.html`.

DSSSL

Auf den großen XML-Sites finden Sie Verweise auf die DSSSL-Komponenten, die direkt relevant für XML sind (wie etwa die XS-Spezifikation). Die folgenden Adressen sind für weitergehend Interessierte gedacht; die erste ist historischer Art (der ursprüngliche DSSSL-o-Entwurf); die anderen sind für diejenigen unter Ihnen gedacht, die die Entwicklung der DSSSL-Spezifikation nachvollziehen wollen (was Sie brauchen, wenn Sie die XS-Spezifikation verstehen wollen):

- Norm Walsh ist der Entwickler der DocBook DSSSL Stylesheets: `http://www.berkshire.net/~norm/dsssl/`.
- Das DSSSL Documentation Project ist eine Initiative Freiwilliger, eine umfassende DSSSL-Dokumentation anzulegen. Die neuesten Dokumente finden Sie unter: `http://www.mulberrytech.com/dsssl/dsssldoc/`.

- James Clark, der Entwickler von Jade, stellt DSSSL-Informationen auf seiner Web-Site bereit: http://www.jclark.com/dsssl/.
- Den ersten Vorschlag für XSL (DSSSL-o), der nicht nur aus historischen Gründen interessant ist, finden Sie unter: http://sunsite.unc.edu/pub/sun-info/standards/dsssl/xs/xs970522.ps.zip.
- Das DSSSL Cookbook ist eine Sammlung praktischer und lehrreicher DSSSL-Codeabschnitte: http://www.mulberrytech.com/dsssl/dsssldoc/cookbook/cookbook.html.
- DSSSL Online Application Profile – Der Entwurf vom 16.8.1996 des Vorläuferstandards von XS: http://sunsite.unc.edu/pub/sun-info/standards/dsssl/dssslo/do960816.htm.
- DSSSL-Tutorial – Eines der besten Tutorials über DSSSL: http://csg.uwaterloo.ca/~dmg/ dsssl/tutorial/tutorial.html.
- Introduction to DSSSL – Paul Prescods kompetente Einführung in die Stilsprache von DSSSL: http://itrc.uwaterloo.ca:80/~papresco/ dsssl/tutorial.html.

SGML

Es gibt Tausende von Web-Sites mit Informationen zu SGML. Hier einige der praktischsten:

- OASIS – Organisation für SGML-Anbieter und andere Unternehmen: http://www.oasis-open.org/.
- SGML FTP Archive – Erik Naggum in Norwegen sammelt seit Jahren Informationen aus der Usenet-Gruppe comp.text.sgml. Das vollständige Archiv mit mehr als 11000 Einträgen finden Sie unter: ftp://ftp.ifi.uio.no/ pub/SGML.
- The SGML University – Praktische Sammlung mit SGML-Informationen: http://www.sgml.com/.
- The SGML Web Page – Die vielleicht beste Online-Quelle für SGML-Informationen: http://www.oasis-open.org/sgml/.
- ISO/IEC JTC1/SC18/WG8 Home Page – Die halboffizielle Site für die ISO-Arbeitsgruppe zu SGML: http://www.ornl.gov/sgml/ WG8/wg8home.htm.
- Charles F. Goldfarb's Web Site – Charles ist einer der Erfinder von SGML – deshalb ist diese Site eine der definitiven Quellen für Informationen zum Thema: http://www.sgmlsource.com/.
- SGML on the Web – Verweise auf Web-Sites, die mit SGML erzeugt wurden: http://www.ncsa.uiuc.edu/ WebSGML/WebSGML.html.

- W3C SGML Working Group Mailing List – Öffentliche Mailing-Liste für die SGML-Arbeitgruppe des W3C: `http://lists.w3.org/ Archives/Public/w3c-sgml-wg/`.

Beachten Sie, daß sich die hier angegebenen Web-Adressen ändern können. Ich habe sie alle überprüft, und viele gibt es schon seit mehreren Jahren; ich kann jedoch nicht für ihre Korrektheit und ihre Dauerhaftigkeit garantieren.

Usenet-Diskussionsgruppen

Es gibt mehrere Usenet-Diskussionsgruppen über XML, die direkt oder indirekt damit zu tun haben.

- `comp.text.xml` – Die definitive Gruppe für alle XML-bezogenen Diskussionen.
- `comp.text.sgml` – Diese Gruppe führt zu viele und einige fehlplazierte Diskussionen über XML, ist aber die Heimat für SGML und verwandte Standards (z.B. DSSSL und HyTime). Die meisten führenden Experten der Welt schreiben bei dieser NewsGroup; es gibt also nur sehr wenige Postings dort, die aber von höchster Qualität sind.
- `public.microsoft.xml` – Eine der vielen öffentlichen Usenet Groups von Microsoft.

Mailing-Listen

Es gibt relativ viele Mailing-Listen zu XML und verwandten Themen. Einige davon sind öffentlich, andere sind für einen festen Kreis, und wieder andere sind moderiert, um sicherzustellen, daß jeder beim Thema bleibt. Im allgemeinen senden Sie eine leere E-Mail-Nachricht an die angegebene Adresse (in der Regel `majordomo`, wobei es sich um ein Softwarepaket handelt, das automatisch auf bestimmte Wörter reagiert), mit dem Wort HELP in der Betreffzeile, um einer Mailing-Liste beizutreten.

- XML Developers' Mailing List – Eine öffentliche Mailing-Liste für höchst technische Diskussionen über die Entwicklung von XML und XML-Software: `http://www.lists.ic.ac.uk/ hypermail/xml-dev/`.
- XML EDI Mailing List – Öffentliche Mailing-Liste für technische Diskussionen über die Verwendung von XML für den elektronischen Datenaustausch (EDI). E-Mail an: `majordomo@bizserve.com`.
- Java-xml Interest Group – Öffentliche Mailing-Liste für extrem technische Diskussionen über die Verwendung von Java zur Verarbeitung von XML-Code. E-Mail an: `majordomo@cybercom.net`.

- Perl-xml Interest Group – Halböffentliche Mailing-Liste für extrem technische Diskussionen über die Verwendung von Perl zur Verarbeitung von XML-Code. E-Mail an: `listmanager@ActiveState.com`.
- XSL List – Öffentliche Mailing-Liste für technische Diskussionen über XSL. E-Mail an: `majordomo@mulberrytech.com`.
- Python-xml SIG – Halböffentliche Mailing-Liste für extrem technische Diskussionen über die Verwendung von Python zur Verarbeitugn von XML-Code. E-Mail an: `XML-SIG-REQUEST@python.org`.
- DSSSL List – Öffentliche Mailing-Liste für technische Diskussionen über die Verwendung von DSSSL. E-Mail an: `majordomo@mulberrytech.com`.
- SGML-HL7 – Halböffentliche Mailing-Liste für extrem technische Diskussionen über SGML oder XML für medizinische Datensätze. E-Mail an: `sgml-hl7@list.mc.duke.edu`.
- Distributed Objects Mailing List – Öffentliche Mailing-Liste für extrem technische Diskussionen über die Entwicklung verteilter Objektsoftware: `http://www.infospheres.caltech.edu/mailing_lists/dist-obj/`.
- html-future Mailing List – W3C-Mailing-Liste nur für Mitglieder. Sie können nicht an der Diskussion teilnehmen, finden das Archiv aber unter: `http://lists.w3.org/Archives/Public/html-future/`.
- DSSSList – The DSSSL Users' Mailing List – Öffentliche Mailing-Liste für die höchst technische Diskussion von DSSSL-Themen: `http://www.mulberrytech.com/dsssl/dssslist/`.

Testdaten

Wenn Sie ernsthaft mit XML experimentieren wollen, brauchen Sie umfangreiche Testdaten. Sie können diese Daten selbst anlegen oder dank der harten Arbeit anderer auf beliebig viel XML-Code im Web zurückgreifen:

- Joseph Conrad's Heart of Darkness – `http://home.sprynet.com/sprynet/dmeggins/texts/darkness/ index.html`
- The Bible, The Koran and The Book of Mormon – `http://sunsite.unc.edu/pub/sun-info/xml/eg/`
- The Works of William Shakespeare (alle Stücke, nicht die Gedichte) – `http://www.hypermedic.com/style/shakespeare/index.htm`
- CSS2 Test Suite, praktisch zur Prüfung, welche CSS2-Funktionen Ihr Web-Browser wie gut unterstützt – `http://speckle.ncsl.nist.gov/~boland/css2/`

CD-ROM zum Buch

Die beiliegende Buch-CD verfügt über eine selbststartende HTML-Oberfläche (Browser vorausgesetzt). Wenn die Oberfläche vollständig geladen wurde können Sie durch Auswählen der einzelnen Menüpunkte diese CD nutzen.

Unter dem Menüpunkt PROGRAMME ZUM BUCH haben die Autoren zusätzliche Materialien abgelegt:

Programm	ScriptWorx
Archiv	SCRPTWRX.ZIP
Beschreibung	Code-basierter High-End-Editor für XML und XSL
Status	Test-Version
Sprache	Englisch

Programm	XML Spy
Archiv	XMLSPY
Beschreibung	Editiert XML., DHTML, DTD und 3DML
Status	Test-Version
Sprache	Englisch

Programm	Xtract
Archiv	XLAT
Beschreibung	XML-Skriptprozessor
Status	Freeware
Sprache	Englisch

Programm	Magic WebValet
Archiv	VALET.ZIP
Beschreibung	Universelles Entwicklungstool zur Erstellung von Internet-Dokumenten. Läßt sich auf XML-Fähigkeit erweitern.

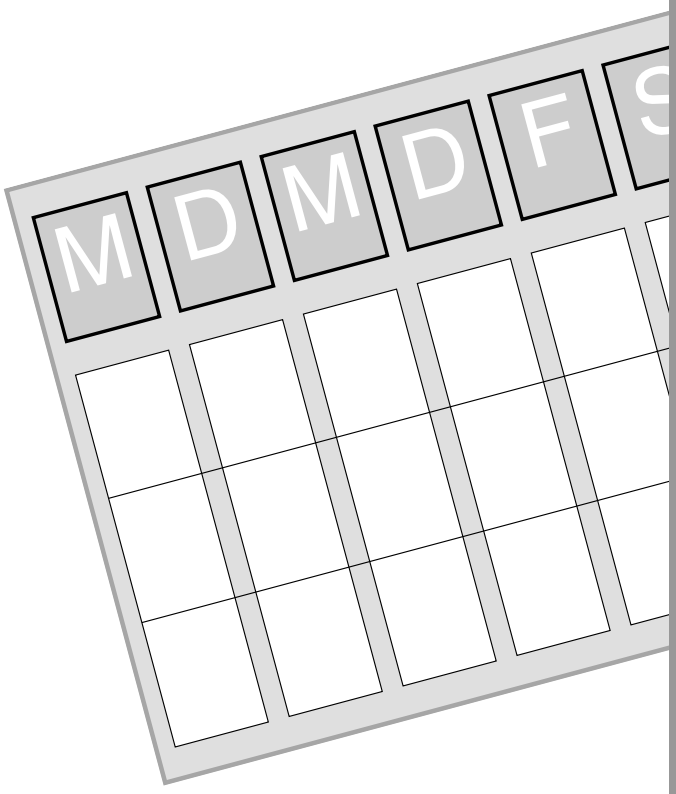
Status	Test-Version
Sprache	Englisch, teilweise deutsche Dokumentation

Programm	XED
Archiv	XED04BTA.EXE
Beschreibung	XML-Editor
Status	Freeware
Sprache	Englisch

Programm	ArborText Styler
Archiv	XINSTALL.EXE
Beschreibung	Komfortables Programm zum Erstellen von XML Stylesheets (XSL). Sehr großer Funktionsumfang. Unterstützt zahlreiche XSL Funktionen.
Status	Freeware
Sprache	Englisch

Programm	Microsoft XML Notepad
Archiv	XMLPAD.EXE
Beschreibung	Einfaches Programm zur übersichtlichen Bearbeitung von XML-Dokumenten. Elemente werden als Baumstruktur angezeigt.
Status	Freeware
Sprache	Englisch

Programm	XML Pro
Archiv	XMLPRO-DEMO.EXE
Beschreibung	Einer der ersten XML-Editoren.
Status	Demo-Version
Sprache	Englisch



Stichwortverzeichnis

Z