Thomas Schraitle *DocBook-XML*

Thomas Schraitle

DocBook-XML

Medienneutrales und plattformunabhängiges Publizieren

2. Auflage

mitlin

DocBook-XML

Medienneutrales und plattformunabhängiges Publizieren

Thomas Schraitle

2. Auflage

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detailierte bibliografische Daten sind im Internet über `http://dnb.ddb.de` abrufbar.

ISBN:	978-3-938626-14-6
Umschlaggestaltung:	Millin Verlag, Lohmar (`http://www.millin.de`)
Gesamtlektorat:	Nicolaus Millin
Fachlektorat:	Jan-Christoph Bornschlegel, Karl Eichwalder, Mike Fabian, Berthold Gunreben, Jana Jaeger, Jan Loeser, Susanne Oberhauser, Martin Polster, Tanja Roth, Frank Sundermeyer, Tobias Wehrum
Layout/Satz:	Thomas Schraitle, gesetzt aus der Charis mit XEP
Druck:	druckhaus köthen GmbH, Köthen (`http://www.koethen.de`)

Printed in Germany on acid free paper

Für meine Eltern und Jürgen

Inhaltsverzeichnis

Vorwort

Ein Buch ohne Vorwort ist wie ein Körper ohne Seele.
—Jiddisches Sprichwort

Warum dieses Buch?

Ungefähr im Jahr 2000/2001 zeigte mir mein Kollege Karl Eichwalder DocBook. Es war neu doch irgendwie vertraut und so suchte ich nach einem deutschsprachigen Buch. Leider war die Ausbeute ziemlich dürftig, um nicht zu sagen: Es gab keines!

Also wühlte ich mich durch die DocBook-Referenz, testete und passte XSLT-Stylesheets an, installierte und konfigurierte Programme, schrieb Dokumente und all die anderen großen und kleinen Dinge, die man zum Arbeiten mit DocBook braucht. Irgendwann formte sich die Idee, aus dem gesammelten Material ein Buch zu erstellen. Zur damaligen Zeit gab es noch SuSE Press und ich schrieb an Nicolaus Millin und stellte meine Idee vor. Er war sofort angetan und seiner Bereitschaft ist es zu verdanken, dass es dieses Buch überhaupt gibt. Nach vielen Stunden wurde die erste Auflage des Buches 2003/2004 veröffentlicht.

Nach mehreren Jahren halten Sie eine aktualisierte Auflage in den Händen. Ich hoffe, dass mit dieser Auflage DocBook bekannter wird und seine Vorteile erkannt werden.

Zielpublikum

Das Buch soll jeden ansprechen der sich für DocBook interessiert: egal ob Sie ambitionierter Anwender sind, ein Entwickler der seine Dokumentation in diesem Format schreiben möchte oder ein technischer Redakteur, der sich aus beruflichen Gründen für die neue Materie interessiert. DocBook ist dann für Sie interessant, wenn mindestens eine der folgenden Punkte auf Sie zutrifft:

- *Unterschiedliche Zielformate und „Single Source Publishing“* Sie schreiben *ein* Dokument, benötigen es jedoch in unterschiedlichen Formaten: als (X)HTML für Ihre Webseite, PDF für den Druck und EPUB für Ihre E-Books. Weitere Formate werden von den DocBook-Stylesheets bereitgestellt.

- *Trennung von Inhalt und Layout* Sie sind daran interessiert, sich nur auf den Inhalt Ihres Dokumentes zu konzentrieren, um in einem späteren Schritt das Aussehen hinzuzufügen. Dadurch lässt sich das Layout zum größten Teil „programmieren“ und die Konsistenz Ihrer Dokumente erhöhen. Durch die DocBook-Stylesheets ist das Layout bereits implementiert.
- *Technische Dokumentation und mehr* Sie suchen ein ausgereiftes Format für (technische) Dokumentationen. DocBook hat aufgrund seiner Geschichte eine starke Affinität zu Programmdokumentationen, dennoch lassen sich auch andere Aufgaben damit umsetzen.
- *Automatisierung* Ihre Dokumente sollen automatisch zu verarbeiten sein. Möglicherweise soll bei jeder Änderung Ihres Dokuments über Nacht Ihr Buch in HTML und PDF erstellt werden, um es am nächsten Tag zu begutachen. Oder Sie möchten automatisch eine Liste aller URLs, Bilder oder Befehle ausgeben um sie maschinell zu überprüfen. All diese Dinge sind mit DocBook in Kombination mit XSLT (*eXtensible Stylesheet Language for Transformation*) möglich.
- *„Variantenmanagement“* Sie benötigen unterschiedliche „Geschmacksrichtungen“ aus demselben Dokument, Varianten die sich nur in Details unterscheiden. Beispielsweise schreiben Sie über eine Anwendung für verschiedene Betriebssysteme, bei denen die Installation und Pfade unterschiedlich sind. Mittels DocBook lassen sich alle Varianten in einem Dokument verwalten.
- *Unabhängig von Herstellern* Sie möchten unabhängig von einem Herstellern sein. DocBook ist ein Gemeinschaftsprojekt unter der Leitung des OASIS-Komitees und ist unter einer freien Lizenz veröffentlicht.
- *Austauschbarkeit* Sie tauschen Ihre Dokumente mit anderen aus. Hierbei ist Ihnen wichtig, dass das Format offen, patentfrei, kostenlos und nach vielen Jahren noch zu lesen und zu verarbeiten ist.

Um Ihnen Enttäuschungen zu ersparen: DocBook ist weniger geeignet für layoutintensive Dokumente, die eine genaue typografische Kontrolle benötigen wie Broschüren, Wurfzettel oder Magazine. Für diese Aufgaben ist ein klassisches DTP- oder Textverarbeitungsprogramm die bessere Wahl. Dennoch soll dieses Buch demonstrieren, dass mit DocBook hochwertige Drucksachen möglich sind — dieses Buch wurde mit DocBook erstellt!

DocBook und XSLT sind mächtige Werkzeuge, die eine gewisse Einarbeitungszeit benötigen. Lassen Sie sich davon nicht entmutigen oder gar abschrecken! Langfristig zahlt sich dies durch eine erhöhte Flexibilität aus.

Welche Vorkenntnisse Sie benötigen

Falls Sie bereits mit LaTeX, HTML oder sogar XML gearbeitet haben, dürfte Ihnen der Umgang mit DocBook relativ leicht fallen. Auch Personen, die noch nicht mit XML in Kontakt gekommen sind, finden in diesem Buch eine Einführung in die Grundlagen.

Dieses Buch verwendet XML-Technologien wie XSLT, XPath und XSL-FO, die für sich allein schon ganze Bücher füllen. Zwar erhalten Sie eine Einführung in diese Sprachen, dennoch ist es empfehlenswert, ergänzende Literatur zu besorgen. Weiteres findes Sie im Literaturverzeichnis (Seite 609).

Wie Sie dieses Buch verwenden

Dieses Buch ist „linear“ aufgebaut: weiter hinten stehende Kapitel stützen sich auf Definitionen oder Vorgehensweisen von früheren. Dies hat seinen Grund: Obwohl DocBook selbst relativ einfach zu erlernen ist, habe ich diesen Aufbau gewählt um den Leser Schritt für Schritt an die Sprache und deren Möglichkeiten heranzuführen. Dadurch sollen Sie ein tieferes Verständnis für die Zusammenhänge erlangen, als durch ein überblicksartiges Tutorial. Für Ungeduldige gibt es Kapitel 1, *DocBook in 10 Minuten* (Seite 1). Wie üblich habe ich viele Beispiele gewählt, um das Ganze verständlicher zu machen.

Aufbau

Entsprechend Ihrem Wissenstand werden unterschiedliche Kapitel empfohlen:

Neulinge
: Beginnen Sie mit dem Kapitel *DocBook in 10 Minuten* (Seite 1) und im Anschluss daran mit dem Grundlagenteil. Wenn Sie nicht beabsichtigen, DocBook anzupassen, überspringen Sie die entsprechenden Kapitel zu DTD und RELAX NG. Für ein tieferes Verständnis sind sie jedoch zu empfehlen. Das Kapitel zu XML-Kataloge sollten Sie jedoch lesen, da spätere Informationen dieses voraussetzen.

Fortgeschrittene
: Überspringen Sie den Grundlagenteil und beginnen Sie mit Teil II, um sich in DocBook einzuarbeiten. Für den Transformationsteil sollten Sie zusätzliche Literatur hinzuziehen.

Experten
: Wählen Sie Ihrem Wissenstand entsprechend die Themen aus, die Sie interessieren.

Kapitel 1 ist ein kurzer Einstieg in DocBook und zeigt überblicksartig die Arbeitsweise.

Teil I: Grundlagen
: Dieser Teil richtet sich an alle Leser, die sich die Grundlagen aneignen möchten.

  Kapitel 2 erklärt Ihnen die Grundlagen von XML.

  Kapitel 3 betrachtet DTD, die „Strukturdefinition“ für XML.

  Kapitel 4 beschreibt RELAX NG, eine andere Art Strukturen in XML zu definieren.

  Kapitel 5 ist eine Einführung, wie Sie URIs durch XML-Kataloge auflösen.

Teil II: DocBook-Markup
Dieser zentrale Teil gibt Ihnen eine Übersicht über DocBook, wie Sie Ihre Texte auszeichnen, gliedern, modularisieren und DocBook an eigene Bedürfnisse anpassen.

Kapitel 6 gibt eine Übersicht über DocBook und erläutert die Unterschiede zwischen Version 4 und 5.

Kapitel 7 erklärt die Strukturelemente wie Buch, Artikel, Kapitel usw.

Kapitel 8 beschreibt die Block-Elemente wie Abbildung, Tabelle, Liste usw.

Kapitel 9 zeigt welche Inline-Elemente es in DocBook gibt und wie sie angewendet werden.

Kapitel 10 hilft Ihnen zu verstehen, wie in DocBook Links und Querverweise ausgezeichnet werden.

Kapitel 11 zeigt, wie Sie modulare Dokumente in DocBook erzeugen.

Kapitel 12 demonstriert, wie DocBook an eigenen Wünsche angepasst wird.

Teil III: DocBook-Transformationen
Wie DocBook in verschiedene andere Formate transformiert wird, behandelt dieser Teil.

Kapitel 13 gibt eine Einführung in XPath, einer Sprache um aus einem XML-Dokument bestimmte Informationen auszuwählen.

Kapitel 14 beschreibt die Grundlagen von XSLT, mit dessen Hilfe Sie DocBook in andere Formate umwandeln.

Kapitel 15 gibt einen Überblick über XSL-FO, einem Zwischenformat beim Erzeugen von PDF.

Kapitel 16 ist ein Querschnitt über die DocBook XSL Stylesheets.

Kapitel 17 demonstriert die Fähigkeiten von *Profiling*, einer Methode um DocBook-Elemente auszublenden.

Kapitel 18 beschreibt die Anpassungen, um DocBook nach HTML umzuwandeln.

Kapitel 19 beschreibt die Anpassungen, um DocBook in PDF umzuwandeln.
Kapitel 20 beschreibt, wie Sie aus Ihrem Dokument ein E-Book erzeugen, das Sie auf jedem E-Book-Lesegerät anzeigen können, sofern es Formate wie EPUB oder Mobipocket versteht.

Teil IV: DocBook vertiefen
Dieser Teil gibt weitere Einblicke in DocBook.

Kapitel 21 gibt verschiedene Hilfestellungen und Tipps zu DocBook.

Kapitel 22 erklärt die Migrationspfade nach DocBook, wenn Sie ein älteres Format haben und es umwandeln möchten.

Kapitel 23 betrachtet einige Komponenten, die für DocBook in naher Zukunft relevant sein könnten.

Anhang A zeigt Ihnen alle Platzhalter im Buch und wie Sie diese für Ihr System auflösen.

Anhang B hilft Ihnen dabei, wie Sie eine XML-Umgebung einrichten, um DocBook zu verarbeiten.

Anhang C listet alle DocBook-Elemente auf und zwar in thematischer und alphabetischer Sortierung.

Typografische Konventionen

`Schreibmaschinenschrift` wird verwendet für:

- Listings
- Alle XML-spezifische Strukturen, wie Element- und Attributnamen, Attributwerte, Entity-Referenzen, Namensräume, Verarbeitungsanweisungen
- Programmspezifisches wie Datei- und Verzeichnisnamen, Umgebungsvariablen, Optionen, Variablen, Funktionen
- Sonstiges wie URLs, URNs, kurze Codeschnipsel usw.

`Schreibmaschinenschrift fett` wird verwendet für:

- Befehle
- Hervorgehobene Stellen in Listings

`Schreibmaschinenschrift kursiv`:

- Zu ersetzende Platzhalter

Kursivschrift wird verwendet für:

- Definition neuer Begriffe
- Für hervorgehobenen Text

Beispiele verwenden

Dieses Buch soll Ihnen bei der Arbeit mit DocBook helfen. Da ich kurze Beispiele liebe, ist das Buch voll davon, häufig durch „Callouts“ dokumentiert.[1]

Teilweise sind die Beispiele in sich abgeschlossen, aus Platzgründen wurde hier und da ein sinnvoller Ausschnitt ausgewählt. Manche dieser Codefragmente sind ungültig, häufig nicht einmal wohlgeformt. Damit das Beispiel funktioniert, müssen Sie dies entsprechend erweitern. Aus dem Zusammenhang sollte dies klar werden.

In diesem Buch beschreibe ich sowohl DocBook 4 als auch DocBook 5. Sofern nicht anders angegeben, kann der Code für beide Versionen verwendet werden. Manchmal ergeben sich jedoch kleine Unterschiede, beispielsweise zwischen den Attributen `id`/`lang` (DocBook 4) und `xml:id`/`xml:lang` (DocBook 5). In diesem Fall steht an der Stelle des Attributnamens der Platzhalter *`ANKER`* oder *`SPRACHE`* und eine Erklärung.

1 Wie Sie diese nützlichen Helferlein für Ihr Dokument nutzen, zeigt Abschnitt 8.11, „Markierungspunkte (Callouts)“ (Seite 176)

Unterscheidet sich der Code zwischen DocBook 4 signifikant von DocBook 5 wird dies wie folgt dargestellt:

```
DB4  Code für DocBook 4
.....................................................................
DB5  Code für DocBook 5
```

Da DocBook plattformunabhängig ist, habe ich möglichst Programme beschrieben, die auf Linux und Windows erhältlich sind. Aus Platzgründen habe ich die Linuxvariante bei Pfaden bevorzugt. Dennoch lässt sich das Beispiel ebenso für Windows anwenden, entsprechende Pfade in den Platzhaltern angepasst. Anhang A (Seite 555) hilft Ihnen dabei, wie Sie diese für Ihr System anwenden.

Sie dürfen die Beispiele in diesem Buch in Ihren Programmen verwenden ohne um Erlaubnis zu fragen. Wir, Verlag und Autor, freuen uns über eine Quellennennung. Unter der URL `http://www.vorkon.de` gibt es ein Archiv mit allen Beispielen.

Was ist neu in der zweiten Auflage?

Vieles hat sich geändert in den Jahren zwischen erster und zweiter Auflage: mittlerweile gibt es DocBook 5, die DocBook-Stylesheets wurden weiter stabilisiert, unterstützen mehr Sprachen und Formate und es gibt weitere wunderbare Tools, die DocBook verwenden. Die grundlegenden Konzepte sind unverändert geblieben.

Die vielfältigen Änderungen, die an der zweiten Auflage vorgenommen wurden, sind zu zahlreich um alle zu erwähnen. Daher nur ein kleiner Auszug: Neu hinzugekommen sind die Kapitel zu RELAX NG, Modulare Dokumente, Anpassungen an DocBook 5, Profiling, Einrichten einer XML-Umgebung, Tipps und Tricks für DocBook und Migration nach DocBook. Des Weiteren wurde der Transformationsteil stark ausgebaut: Korrekturen einer Titelseite, Nutzen einer gemeinsame Bibliografie- oder Glossardatenbank, Anpassen von Überschriften sind einige der interessanten Themen. Das gesamte Buch wurde für DocBook 5 erweitert.

Des Weiteren wurde der Transformationsteil stark ausgebaut: Korrekturen einer Titelseite, Nutzen einer gemeinsame Bibliografie- oder Glossardatenbank, Anpassen von Überschriften sind nur einige der interessanten Themen.

Danksagungen

Einige besondere Menschen möchte ich erwähnen, die einen maßgeblichen Anteil am Gelingen dieses Buches hatten:

Danksagung zur zweiten Auflage

- Nicolaus Millin, meinem Verleger und Lektor beim Millin-Verlag. Vielen Dank Nico für deine grenzenlose Geduld und Ausdauer mit mir und meinem Buch. Deine Ermutigungen und Erfahrungen haben das Buch wesentlich geprägt und bereichert.

- Ich habe das Glück gehabt, dass Antje Faber viele meiner Kapitel korrigiert hat. Vielen Dank Antje, deine umfangreichen Vorschläge waren unbezahlbar und haben das Buch entscheidend vorangebracht.
- Einen besonderen Dank schulde ich Tobias Wehrum. Seine umfangreichen Korrekturen, Vorschläge und Ideen haben das Buch sehr bereichert und verbessert.
- Einen großen Anteil am Gelingen des Buches haben meine vielen Testleser Jan Blunck, Jan-Christoph Bornschlegel, Karl Eichwalder, Berthold Grunreben, Jana Jaeger, Susanne Oberhauser, Martin Polster, Tanja Roth und Frank Sundermeyer. Tut mir leid, dass ich euch immer nerven musste! Eure Korrekturen und Ideen waren sehr wertvoll für mich.
- An Robert Lihm und Martin Schmidkunz für ihre tollen Ideen zum Cover des Buches.
- Mike Fabian und Jan Loeser für Ihre Mühe beim Korrigieren des E-Book-Kapitels. Mike hat mir beim Testen meiner EPUB-Datei auf seinem Lesegerät geholfen und Jan hat das Kapitel durch seine vielen Vorschläge verbessert.
- Den vielen Fragestellern der DocBook-Mailingliste, wo ich immer wieder Neues entdecken und lernen kann.
- Den Entwicklern von Subversion, ohne die das Buch wesentlich schwieriger zu verwalten gewesen wäre.
- An Muhamed Memovic und Helge Schimeck für ihren unermüdlichen und heldenhaften Einsatz beim Reparieren meines unwilligen PCs.
- Und Jürgen Streb, der mich auch diesesmal wieder in vielfältiger Weise unterstützt hat. Danke Jürgen!

Danksagung zur ersten Auflage

- Zuerst möchte ich meinem Lektor Dr. Markus Wirtz (jetzt Open Source Press GmbH `http://www.opensourcepress.de`) nicht nur für die Diskussion über Rachmaninov-Interpretationen, sondern insbesondere für seine Geduld, Ausdauer und Bemühungen danken, die über einen langen Zeitraum viel dazu beigetragen haben, dem Buch die vorliegende Form zu geben. Hinweise auf eventuell verbliebene Fehler, die sich wohl niemals vermeiden lassen, nehmen wir gerne entgegen und versichern, sie in der nächsten Auflage zu berücksichtigen.
- Nicolaus Millin von SUSE PRESS, der dieses Buch erst möglich gemacht hat.
- Einen maßgeblichen Anteil am Gelingen dieses Buches haben meine Testleser Karl Eichwalder, Dennis Geider, Ralf Haferkamp, Christoph Niemann, Susanne Oberhauser, Stefan Probst und Thomas Rölz, die großzügig von ihrer Zeit investiert haben. Ihr Feedback in fachlichen, sprachlichen und technischen Korrekturen war für mich von unschätzbarem Wert.

  Antje Faber, Berthold Gunreben, Stefan Hundhammer und Jana Jaeger schulde ich besonderen Dank, da von ihnen große Teile gelesen wurden. Viele ihrer

konstruktiven Vorschläge sind eingeflossen und haben das Buch entscheidend verbessert. Mit Sicherheit sind sie jetzt um viele Rotstifte ärmer…

- Roman Halstenberg danke ich für die freundliche Bereitstellung des SGML/XML-Editors epcEdit
- Mein besonderer Dank gilt Peter Reinhart, der fast das komplette Buch durchgeschaut und mir das umfangreichste Errata zugeschickt hatte. Hier handelte es sich nicht nur um Schreibfehler, sondern auch um sachliche Unkorrektheiten, Layoutfehler usw. Dank seiner Kompetenz, Geduld und Mühe hat dieses Buch gewiß an Qualität gewonnen.
- Matthias Voigt für unsere vielen interessanten Gespräche.
- Und Jürgen Streb für seine moralische Unterstützung.

Durch die harte Arbeit von zahlreichen Entwicklern konnte dieses Buch überhaupt erst entstehen. Es sind derart viele, dass ich hier nur einige aufzählen kann:

- Norman Walsh für DocBook, den DocBook-Stylesheets und noch viele weitere Tools.
- Bob Stayton für seine konstante Hilfe und Pflege der DocBook-Stylesheets auf den DocBook-Mailinglisten. Vielen Dank Bob, für die Hilfe mit den Kopfzeilen im Index!
- Daniel Veillard für seinen XML-Parser `xmllint` und seinen XSLT-Prozessor `xsltproc`. Seine Tools haben die Arbeit am Buch entscheidend erleichtert.
- Michael Kay für seinen XSLT-Prozessor Saxon.
- Daniel Naber für sein XML-Plugin des KDE-Editors Kate. Die ersten Versionen des DocBook-Quellcodes wurden damit geschrieben.
- James Clark für nXML, einem Emacs-Modus um XML-Dokumente mit RELAX NG zu schreiben. Und natürlich für RELAX NG! Ohne diese einfache Schemasprache kann ich mir DocBook 5 nicht mehr vorstellen.
- Lennard Staflin für PSGML, den SGML/XML-Modus von Emacs.

Herzlichen Dank an alle! Und all meinen Lesern wünsche ich viel Erfolg mit DocBook.

Nürnberg, im August 2009
Thomas Schraitle

E-Mail: <docbook-xml@online.ms>
Pakete: http://download.opensuse.org/repositories/home:/thomas-schraitle/
Blog: http://lizards.opensuse.org/author/thomas-schraitle/

Kapitel 1
DocBook in 10 Minuten

Dieses Kapitel ist ein Einstieg in die verschiedenen Themen rund um DocBook und soll Ihnen einen ersten Eindruck vermitteln, welche vielfältigen Möglichkeiten DocBook bietet. Im Anschluss werden diese Themen dann im Detail behandelt.

> *Don't panic.*
> — Douglas Adams

1.1 Warum DocBook?

Hier eine kleine Zusammenfassung der wichtigsten Argumente, die für DocBook als Grundlage umfangreicher Dokumentationsprojekte sprechen:

Technisch

- *Medienneutrales Format* Ein DocBook-Dokument kann in unterschiedliche Formate konvertiert werden, wie (X)HTML, XSL-FO und einige Andere. Dadurch müssen Sie sich nicht um jedes einzelne Ausgabeformat kümmern, sondern konzentrieren sich lediglich auf Ihr DocBook-Dokument.
- *Automatisierung* Die Verarbeitung von DocBook lässt sich automatisieren. Das bedeutet, dass Sie quasi „auf Knopfdruck" Ihre Formate erzeugen.
- *Anpassungsfähigkeit an eigene Bedürfnisse* Sowohl DocBook selbst als auch die Umwandlung durch sog. XSLT-Stylesheets lassen sich auf vielfältige Weise an Ihre Bedürfnisse anpassen.
- *Offene Spezifikationen* DocBook und andere Hilfsmittel basieren auf offenen und allgemein akzeptierten Spezifikationen wie XML, DTD, RELAX NG und W3C-Schema (die ersten drei werden Sie im Laufe des Buches kennen lernen).

Ökonomisch und Politisch

- *Beständigkeit* Dokumente in diesem Format sind „langlebig“. Das bedeutet, Ihre Dokumente sind noch nach vielen Jahren zu lesen und zu verstehen. Im Vergleich zu proprietären Formaten benötigen Sie kein spezielles Programm zum Lesen, denn DocBook ist Text mit Markup.
- *Weite Verbreitung* DocBook ist mittlerweile zu einem „De-facto-Industriestandard“ geworden, der von vielen Institutionen und Unternehmen genutzt wird: KDE, GNOME, das Linux Documentation Projekt, der Linux-Kernel, verschiedene Linux Distributoren aber auch größere Firmen schreiben ihre Dokumente in diesem Format. Unter `http://wiki.docbook.org/topic/WhoUsesDocBook` finden Sie weitere Namen.
- *Plattformunabhängigkeit* DocBook ist ein *plattformunabhängiges Format*, das heißt, schreiben und verarbeiten Ihres Dokuments lässt sich sowohl unter Linux als auch unter Windows oder irgend einem anderen System durchführen.
- *Freie Lizenz* DocBook ist freie Software und unter der BSD-Lizenz veröffentlicht. Dadurch haben Sie die Freiheit, es zu kopieren, zu verändern, zu verbreiten und sogar Geld damit verdienen. Weitere Informationen finden Sie unter `http://de.wikipedia.org/wiki/BSD-Lizenz`.
- *Lange Entwicklungszeiten* Die Entwicklung von DocBook vollzieht sich in längeren Zyklen. Inkompatible Änderungen werden vom Technischen Komitee von OASIS bereits eine Version zuvor bekannt gegeben (siehe `http://www.oasis-open.org/committees/docbook/`). Folglich können Sie sich frühzeitig auf Änderungen einstellen und einen eventuellen Umstieg besser planen.
- *Dokumentation* DocBook ist gründlich dokumentiert, siehe [13, 11].

Der Vollständigkeit halber seien aber auch einige Nachteile genannt:

- DocBook bringt den größten Nutzen, wenn aus dem Ursprungsdokument mehr als ein Format erzeugt werden soll.
- Für Dokumente mit sehr komplexem Layout ist DocBook weniger geeignet; hier sind DTP-Programme meist die bessere Wahl.
- Wird in DocBook das Inhaltsmodell geändert, müssen ebenso die verwendeten Stylesheets (CSS und XSLT) angepasst werden. Dadurch müssen Sie zusätzlich eine Stylesheet-Sprache lernen.

1.2 Wie sieht ein DocBook-Dokument aus?

DocBook ist primär für technische Dokumentationen aller Art entwickelt worden, um Bücher und Artikel in unterschiedlichen Medien bzw. Zielformaten, wie z. B. HTML oder PDF wiederzugeben. Jedoch ist es flexibel genug, um unterschiedliche Typen von Dokumenten zu beschreiben. Wie ein Artikel in DocBook aussehen kann, zeigt Ihnen folgendes Beispiel.

Beispiel 1.1. *Ein DocBook-Artikel (first.xml)*

```
<?xml version="1.0">
<!DOCTYPE article PUBLIC
  "-//OASIS//DTD DocBook XML V4.5//EN"
  "http://www.docbook.org/xml/4.5/docbookx.dtd">
<article lang="de">
  <title>Mein erster Artikel</title>
  <section>
    <title>Eine Abschnittsüberschrift</title>
    <para>Hier steht ein Absatz.</para>
  </section>
</article>
```

Das vorige Beispiel zeigt einen Artikel mit einem Abschnitt der einen Absatz enthält. Die Details sind vorerst unwichtig, diese werden in den nächsten Kapiteln ausführlich besprochen. Wichtig sind vielmehr die folgenden Eigenschaften, die sich durch das bloße Betrachten ergeben:

1. Die DocBook-Datei ist eine Textdatei auf Basis von XML, erkennbar an der ersten Zeile.
2. Ihr Text enthält zusätzliche Markierungen, so genannten *Tags* wie `<article>` oder `<para>`.
3. Tags sind untereinander verschachtelt. Jedes Tag hat einen „Anfang“ (wie `<article>`) und ein dazugehöriges „Ende“ (`</article>`).
4. Die Namen der DocBook-Tags sind in Englisch und beschreiben den Inhalt.
5. Tags sagen nur aus, *was* Sie vor sich haben, nicht jedoch *wie* es aussieht.

Gerade der letzte Punkt ist entscheidend, denn dadurch lässt sich ein darstellungsunabhängiges Format in verschiedene Zielformate überführen. Dies hat auch Konsequenzen darauf, wie Sie Ihr Dokument schreiben:

1. Sie verwenden ein Tag aufgrund des Inhalts, nicht aufgrund der Darstellung („semantische Auszeichnung“).
2. Die Darstellung wird in einem separaten Prozess hinzugefügt.

Dieses Verfahren zieht sich durch die gesamte DocBook-Verarbeitung. Hierzu benötigen Sie verschiedene Komponenten. Ihr DocBook-Dokument ist eines davon. All diese Teile greifen wie Zahnräder ineinander. Gesteuert wird das Zusammenspiel manuell, durch Skripte oder durch ein übergeordnetes Programm (beispielsweise durch Ihren XML-Editor).

Erste Möglichkeit ist fehleranfällig, daher werden häufig Skripte oder Programme eingesetzt, die alle diese Komponenten in einer vorgegebenen Reihenfolge aufrufen. Das Zusammenspiel all dieser Teile wird in diesem Buch als *XML-Umgebung* bezeichnet und wird in Abschnitt 1.8 (Seite 9) erklärt.

Bevor Sie die XML-Umgebung kennen lernen, erfahren Sie, welche Verarbeitungsschritte für DocBook notwendig sind.

1.3 Welche Verarbeitungsschritte werden benötigt?

Um ein DocBook-Dokument zu verarbeiten, werden Sie für gewöhnlich immer die selben Schritte durchlaufen. Dieser Verarbeitungsprozess von DocBook wird in folgender Abbildung gezeigt.

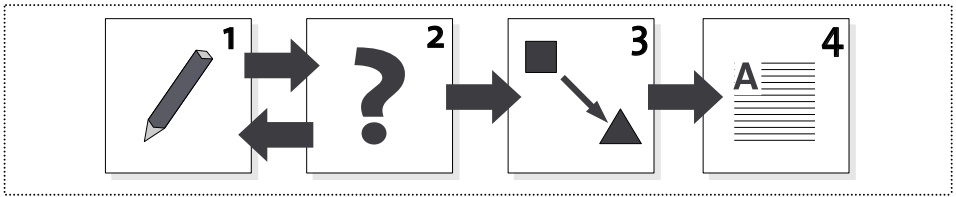

Abbildung 1.1. *Verarbeitungsprozess von DocBook*

Die einzelnen Schritte bedeuten:

1. *Schreiben*
   Sie schreiben Ihren Text im DocBook-Format, hierbei unterstützt Sie ein *XML-Editor*. Falls Sie ältere Dokumentationen in einem anderen Format besitzen, müssen Sie diese zuerst nach DocBook konvertieren.
2. *Überprüfen (Validieren)*
   Sie überprüfen Ihr DocBook-Dokument mit Hilfe eines *XML-Parsers*. Dieser liest Ihr Dokument ein und vergleicht es mit dem DocBook-Schema. Fehler müssen behoben werden, da weitere Schritte sich auf ein einwandfreies („valides“) Dokument verlassen. Obwohl dies von manchen als Gängelung empfunden wird, ermöglicht eine Überprüfung, dass der Prozess immer zuverlässig dasselbe Resultat liefert.
3. *Umwandeln (Transformieren)*
   Sie transformieren das überprüfte DocBook-Dokument in das gewünschte Zielformat, für gewöhnlich mit einen *XSLT-Prozessor*. Dieser liest XSLT-Stylesheets ein, die Regeln enthalten, um daraus Ihr Zielformat zu erstellen. Als XSLT-Stylesheets werden in diesem Buch die DocBook-Stylesheets beschrieben.
4. *Formatieren/Darstellen*
   PDF wird für gewöhnlich mit Hilfe eines *FO-Formatierers* generiert und für Manpages wird der Befehl `man` verwendet. Bei manchen Formaten entfällt dieser Schritt, beispielsweise wenn Sie eine Textausgabe erzeugen oder (X)HTML generieren.

Der vorige Prozess genügt normalerweise für die Mehrzahl aller Dokumente. Für ausgefallenere Wünsche lassen sich Zwischenschritte einfügen, zum Beispiel um vor dem Formatieren bestimmte Strukturen auszublenden oder andere Daten einzufügen.

Die folgende Tabelle zeigt Ihnen den gesamten Prozeß nochmals im Überblick:

Tabelle 1.1. *Verwendete Programme für die einzelnen Schritte der DocBook-Verarbeitung*

Schritt	*Programm(e)*	*Benötigte Dateien*
1. Schreiben	XML-Editor	DocBook-Dokument, DocBook-Schema
2. Überprüfen	XML-Parser	DocBook-Dokument, DocBook-Schema
3. Umwandeln	XSLT-Prozessor	DocBook-Dokument, DocBook-XSLT-Stylesheet(s), (DocBook-Schema)
4. Formatieren/ Darstellen	Browser, FO-Formatierer, …	evlt. Zwischenformat aus Transformationsschritt, Schriften, Grafiken

1.4 Schreiben eines DocBook-Dokuments

DocBook-Dokumente schreiben Sie auf eine der folgenden Arten:

- *XML-Editor* Ein Editor, der Sie bei der Eingabe von XML unterstützt. Ein XML-Editor liest das DocBook-Schema und Ihr Dokument ein und zeigt Ihnen auf Abruf eine Liste an erlaubten Tags, die Sie an der aktuellen Position einfügen dürfen. Manche dieser Editoren verbergen die Tags und präsentieren eine Darstellung ähnlich den bekannten Textverarbeitungen. Ein XML-Editor ist die bevorzugte Methode, um mit DocBook zu arbeiten.

  Im Open Source Bereich gibt es leider nur wenige XML-Editoren, bekannte Programme sind Emacs und jEdit. Im kommerziellen Bereich ist die Auswahl bedeutend größer, wobei auch die Bandbreite der Fähigkeiten sehr unterschiedlich sind. Unter der URL `http://wiki.docbook.org/topic/DocBookAuthoringTools` finden Sie eine Auswahl von gängigen XML-Editoren, sowohl kommerziell als auch Open Source.

- *Herkömmlicher Text-Editor* Alle Nicht-XML-Editoren. Prinzipiell lässt sich jeder Editor verwenden, um ein DocBook-Dokument zu schreiben. Allerdings ist es mühsamer als mit einem XML-Editor, da Sie die Tags manuell eingeben müssen. Des Weiteren zeigt ein herkömmlicher Editor keine Tags an, die an der aktuellen Position erlaubt sind. Je nach Fähigkeiten Ihres Editors lassen sich bestimmte Nachteile teilweise ausgleichen, indem Sie Makros oder Textbausteine verwenden.

Wie ein Testdokument im Editor Emacs aussieht, zeigt Abbildung 1.2, „Testdokument im Editor Emacs“ (Seite 6).

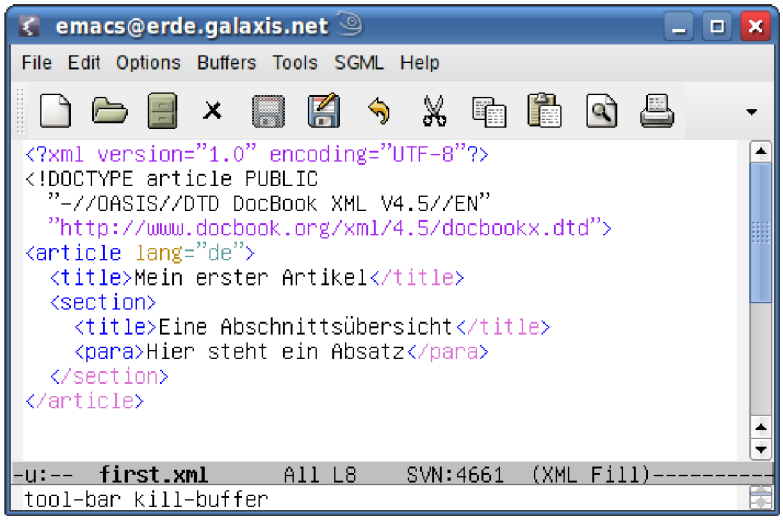

Abbildung 1.2. *Testdokument im Editor Emacs*

1.5 Überprüfen eines DocBook-Dokuments

XML-Dateien lassen sich nicht nur syntaktisch, sondern auch strukturell überprüfen. Der Überprüfungsvorgang heißt im Fach-Jargon *Validierung* und wird mit einem *XML-Parser* vorgenommen. Gängige XML-Parser sind `xmllint` (Paket libxslt), `msv`, `jing` und andere. Ein XML-Parser überprüft Ihr DocBook-Dokument mit den Regeln des zugehörigen DocBook-Schema.

Das DocBook-Schema legt fest, was für Tag-Namen erlaubt sind und in welcher strukturellen Beziehung sie zueinander stehen. Beispielsweise definiert das DocBook-Schema, wie ein Buch aufgebaut ist (Titel, verschiedene Kapitel, Anhang, möglicherweise ein Glossar usw). Weicht Ihr Dokument von dieser Definition ab, ist es ungültig.

Das Dokument `first.xml` von Abbildung 1.3 (Seite 7) wurde mit Hilfe des DocBook-Schema überprüft. Der folgende Aufruf verwendet den XML-Parser `xmllint`:

```
xmllint --noout --valid first.xml
```

Die Option `--valid` aktiviert die Validierung und `--noout` unterdrückt die Ausgabe des validierten Dokuments, ausgenommen mögliche Fehlermeldungen. In der folgenden Abbildung wurde der Parser innerhalb des Editors aufgerufen. Zusätzlich wurde im rechten Bild ein Fehler eingebaut, um Ihnen den Unterschied zu demonstrieren.

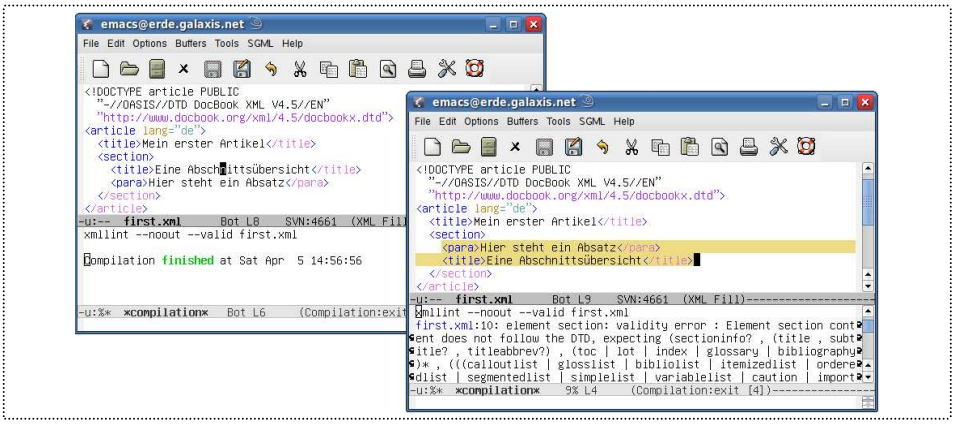

Abbildung 1.3. *Validierung ohne Fehler (links) und mit Fehlern (rechts)*

1.6 Umwandeln eines DocBook-Dokuments

Eine der Vorteile von DocBook ist, dass Sie nur ein Dokument schreiben, es sich jedoch in nahezu beliebige Ausgabeformate umwandeln lässt („Single-Source-Publishing“). Die bevorzugte Methode hierzu ist die Transformation mit den *DocBook-Stylesheets* mit Hilfe eines XSLT-Prozessors.

Die DocBook-Stylesheets bestehen aus unterschiedlichen Dateien, welche jeweils nur ein Format ausgeben. Gegenwärtig werden als Ausgabeformate Manpages, HTML, XHTML, XSL-FO, HTMLHelp, EPUB (ein Format für E-Books), JavaHelp und ein Format für das Eclipse Hilfesystem erzeugt.

Gängige XSLT-Prozessoren sind `xsltproc` aus der libxslt-Bibliothek, Saxon oder Xalan. Um Ihr DocBook-Dokument mit `xsltproc` nach HTML umzuwandeln, geben Sie ein:

```
xsltproc --output first.html DB/html/docbook.xsl first.xml
```

Der Platzhalter *DB* zeigt auf Ihr Installationsverzeichnis der DocBook-Stylesheets. Um Ihr DocBook-Dokument nach PDF umzuwandeln, müssen Sie zuerst das Zwischenformat XSL-FO (Endung `.fo`) erstellen. Geben Sie folgendes ein:

```
xsltproc --output first.fo DB/fo/docbook.xsl first.xml
```

1.7 Formatieren eines DocBook-Dokuments

Um das Endergebnis darzustellen, benötigen Sie je nach Zielformat ein anderes Programm. Für HTML ist ein beliebiger Browser erforderlich. Das Ergebnis ist in der folgenden Abbildung gezeigt:

Abbildung 1.4. *Ergebnis der DocBook-Umwandlung nach HTML*

Im letzten Schritt wurde als zweites Format XSL-FO als ein Zwischenformat für PDF erzeugt. Zur anschließenden Umwandlung in PDF benötigen Sie einen FO-Formatierer. Gängige Produkte sind AntennaHouse Formatter, FOP, XEP und noch einige andere.

Der folgende Aufruf verwendet den FO-Formatierer FOP, um aus XSL-FO das Zielformat PDF zu erstellen:

```
fop first.fo first.pdf
```

Das Ergebnis unseres vorigen Aufrufs sehen Sie in Abbildung 1.5, „Ergebnis der DocBook-Umwandlung nach PDF“ (Seite 8).

Abbildung 1.5. *Ergebnis der DocBook-Umwandlung nach PDF*

1.8 Eine XML-Umgebung

Eine XML-Umgebung fasst die Verarbeitungsschritte von Abschnitt 1.3, „Welche Verarbeitungsschritte werden benötigt?“ (Seite 4) zusammen: schreiben, überprüfen, umwandeln und formatieren.

Da die Installation und Konfiguration lästig und zeitraubend ist, finden Sie unter der Adresse `http://www.vorkon.de` vorkonfigurierte Pakete für Linux und Windows. Nachdem Sie das Archiv heruntergeladen und installiert haben, können Sie sofort loslegen und DocBook erkunden. Kenner finden in Anhang B, *XML-Umgebung einrichten* (Seite 557) die nötigen Schritte, dies auf Ihrem Rechner selbst einzurichten.

1.9 Die Reise beginnt...

Sie haben nun einen kleinen Einblick in die Möglichkeiten bei der Erstellung und Weiterverarbeitung von DocBook-Dokumenten erhalten. In den folgenden Kapiteln werden diese Themen detailliert ausgeführt und stets an Beispielen verdeutlicht. So erfahren Sie alles Wissenswerte, das Sie zum erfolgreichen Einsatz von DocBook benötigen.

I
Grundlagen

Kapitel 2
Einführung in XML

XML, die eXtensible Markup Language, gilt als eine der wichtigsten Techniken zur Strukturierung von Daten. Sie schafft eine Grundlage, um Daten mit Hilfe von einfachen, von Menschen lesbaren Tags auszuzeichnen. Dadurch ist die Sprache flexibel genug, um auf solch unterschiedliche Dinge angepasst zu werden wie Vektorgrafiken, mathematische Formeln, Musiknoten, chemische Reaktionen oder technische Dokumentationen. Dieses Kapitel führt Sie in die Grundlagen dieser faszinierenden Sprache ein und gibt einen Überblick, worauf in XML-Dokumenten zu achten ist und in welchen Bereichen XML zur Anwendung kommt.

> *XML will be the ASCII of the Web—basic, essential, unexciting*
> —Tim Bray

2.1 Geschichtliches zu SGML, HTML und XML

Lange bevor XML die Bühne der IT-Welt betrat, gab es die *Standard Generalized Markup Language*, kurz SGML. Diese ist praktisch der Vorgänger von XML, und wir werden sie kurz betrachten, bevor wir uns mit XML beschäftigen. Die *Standard Generalized Markup Language* wurde in den 70er Jahren von Charles Goldfarb, Ed Mosher und Ray Lorie bei IBM erfunden und im Jahre 1986 von der International Organization for Standardization (ISO) unter der Nummer ISO 8879 als Standard verabschiedet.

Idee und Ziel von SGML ist es, Struktur und Layout strikt voneinander zu trennen. Wo früher der Schwerpunkt bei Programmen auf der typografischen Darstellung lag, verlagert er sich mit SGML hin zur logischen und strukturellen Auszeichnung, so genanntem *Markup*. Bemerkenswert an SGML ist, dass es darauf ausgelegt ist, *beliebige* Dokumente mit den *unterschiedlichsten* Strukturen zu beschreiben. SGML ist eine „Meta-Sprache“, das heißt, sie gibt nur syntaktische Bedingungen vor, wie eine Markup-Sprache auszusehen hat. Sie ist gewissermaßen eine Sprache, um anderen Markupsprachen zu erstellen. Der Entwickler definiert diese in den Grenzen und Möglichkeiten von SGML.

Durch die Verlagerung auf die strukturelle Auszeichnung ergeben sich zwei grundsätzliche Vorteile von SGML:

Vollständige Hard- und Softwareunabhängigkeit
: Die Bearbeitung von Dokumenten ist nicht an spezifische Hard- oder Software gebunden.

Medienneutrale Publikation
: Daten lassen sich in unterschiedliche Formate konvertieren.

Die Nachteile sollen jedoch nicht verschwiegen werden:

Komplexität
: SGML-Systeme können viel komplexer als andere Systeme sein, die auf ganz bestimmte Aufgaben spezialisiert sind (beispielsweise Web, Typografie usw.)

Kosten
: Diese hohe Komplexität SGML-basierter Systeme bedingt höhere Kosten.

Diese Nachteile konnten die Verbreitung von SGML jedoch nicht aufhalten. Es hat sich in den Bereichen durchgesetzt, in denen sehr viele Informationen aktuell gehalten werden müssen, wie etwa bei technischen Anleitungen für Flugzeuge, im Schiffbau oder in der Raumfahrt.

Eine der populärsten Anwendungen von SGML ist HTML (engl. *Hypertext Markup Language*). Die Physiker am Zentrum für Hochenergiephysik (CERN) benötigten ein Verfahren, um möglichst schnell Informationen mit anderen Wissenschaftlern auszutauschen, und so entwickelteTim Berners-Lee 1989 die Markup-Sprache HTML auf der Basis von SGML. Damit wurde es möglich, Texte zu *verlinken*, das heißt, in einem Text Verweise auf andere Dokumente zu integrieren und mit einem speziellen Anzeigeprogramm („Browser“) über diese Verweise direkt auf die entsprechenden Dokumente zuzugreifen. Das System setzte sich insbesondere im universitären Bereich rasch durch.

HTML-Code selbst zu schreiben ist sehr einfach, was sicherlich zu dessen Verbreitung beigetragen hat. Das Internet wäre ohne HTML undenkbar, und es kann nicht oft genug betont werden, wie wichtig ein offenes, plattformübergreifendes, lizenzfreies, herstellerunabhängiges und jedem zugängliches Format wie HTML ist. Oder, um es mit einem Schreckensszenario noch deutlicher zu zeichnen: Stellen Sie sich vor, Sie müssten für das bloße Betrachten einer Webseite Lizenzgebühren zahlen!

Vermutlich ahnte auch Tim Berners-Lee diese Gefahr und gründete deshalb in Zusammenarbeit mit dem CERN bereits im Jahre 1994 das World Wide Web Consortium, kurz W3C. Unterstützt wurden sie dabei von *Defense Advanced Research Projects Agency*, kurz DARPA und der Europäischen Kommission.

„Leading the Web to its Full Potential“ – das Internet zu seinem vollen Potential zu bringen, das ist nach eigener Aussage die Aufgabe des W3C. Obwohl das W3C keine verpflichtenden Normen ähnlich wie ISO oder DIN (Deutsches Institut für Normung)

Kapitel 3
Dokumententyp-Definitionen (DTDs)

Dieses Kapitel führt Sie in die Syntax von Dokumententyp-Definitionen (DTD) ein, die Sie mit Ihrem XML-Dokument verknüpfen können. In DTDs wird genau definiert, welche Elemente, Entities und Attribute zur Verfügung stehen und wie ein Dokument auszusehen hat, damit es gültig ist. Anwender, die keine Anpassungen an DocBook vornehmen möchten, benötigen die Ausführungen in diesem Kapitel zum weiteren Verständnis nicht unbedingt und können gleich in Kapitel 6, *Übersicht über DocBook* (Seite 111) einsteigen.

> *Es [Licht] ist Energie und ebenso Information – Inhalt, Form und Struktur.*
> *Es bildet das Potential für alles.*
> —David Bohm

3.1 Wozu DTDs?

Sie können ein XML-Dokument auch ohne DTD schreiben. Dennoch besitzen DTDs einige Vorteile:

- Die Syntax einer DTD ist kompakt aber dennoch einfach zu erlernen
- Zwischen XML-Dokument und DTD gibt es einen „Vertrag“ über die erlaubte Struktur.
- DTDs gibt es schon zu SGML-Zeiten und sind dementsprechend durch Tools und Editoren sehr gut unterstützt.
- Die DTD-Syntax ist in der XML-Spezifikation eingebaut, jedoch auch die Tatsache, dass ein XML-Parser diese nicht zwingend unterstützen muss. Dennoch sind heutige XML-Parser ohne DTD-Unterstützung kaum mehr anzutreffen, da es mittlerweile Standard geworden ist.

DTDs haben folgende Nachteile:

- DTDs kennen keine Namensräume. Die XML-Namespace Spezifikation [54] teilt den Elementnamen in einen Präfix und einen lokalen Namensteil auf. Der Präfix kann beliebig gewählt werden, er ist einem Namensraum-URI zugeordnet. Eine DTD kennt diese Trennung nicht. Ob mit oder ohne Präfix, eine DTD „sieht“ nur einen ungeteilten Elementnamen. Dadurch sind beispielsweise zwei `title`-Elemente mit dem selben Namensraum, die sich nur durch einen Präfix unterscheiden, für eine DTD unterschiedliche Elemente. Dies führt vor allem beim Mischen von verschiedenen DTDs zu Problemen, da hierbei Namenskonflikte auftreten.
- DTDs besitzen nur sehr schwachte Datentypen. Es ist beispielsweise nicht möglich, in einer DTD zu definieren, dass ein Tag nur ein Datum als Inhalt haben darf.
- DTDs besitzen keine XML-Syntax, sondern haben ein eigenes Format. Das führt dazu, dass sich DTDs nicht mit gängigen XML-Werkzeugen behandeln lassen.
- DTDs lassen sich nur über XML-Kommentare dokumentieren. Andere Schemasprachen haben hierfür ein eigenes Element.
- DTDs verbieten so genannte *nicht-deterministische Inhaltsmodelle*. Dies liegt dann vor, wenn in einem Inhaltsmodell Mehrdeutigkeiten auftreten. Siehe Abschnitt 3.2.10, „Nicht-deterministisches Inhaltsmodell“ (Seite 47) für weitere Erklärungen.

3.2 Elementdeklarationen

In einer DTD werden Element-Deklarationen verwendet, die einen Element-Namen mit einem so genannten *Inhaltsmodell* assoziieren:

Beispiel 3.1. *Syntax einer allgemeinen Elementdeklaration*

```
<!ELEMENT Elementname Inhaltsmodell>
```

Für den Elementnamen gelten dabei die in Abschnitt 2.3.2, „Gültige XML-Namen“ (Seite 17) genannten Bedingungen. Der Platzhalter *`Inhaltsmodell`* gibt bestimmte Schlüsselwörter an, wie die Kindelemente heißen, in welcher Reihenfolge diese auftreten dürfen und in welcher Anzahl. Die folgenden Unterabschnitte geben an, welche Regeln für das Inhaltsmodell innerhalb einer Elementdeklaration möglich sind.

3.2.1 Das Schlüsselwort EMPTY

Der wohl einfachste Fall einer Elementdeklaration ist *kein* Inhalt; dies wird mit dem Schlüsselwort `EMPTY` erreicht:

```
<!ELEMENT leer EMPTY>
```

Durch dieses Schlüsselwort kann das Element `leer` nur auf eine der beiden folgenden Arten genutzt werden:

```
<leer/>
<leer></leer>
```

Zwischen <leer> und </leer> darf sich *nichts* befinden, nicht einmal Leerzeichen! Bei der folgenden Zeile würde der Parser eine Fehlermeldung anzeigen:

```
<leer>Dies ist nicht zulässig!</leer>
```

Leere Elemente haben üblicherweise eine funktionale, keine inhaltliche Bedeutung, das heißt, sie dienen nicht der Gliederung oder Formatierung von Text oder anderen Daten. So enthält DocBook zum Beispiel ein <xref/>-Tag:

```
<xref linkend="chap.xmlintro"/>
```

Hier ist das Attribut linkend ein Verweis auf ein Kapitel, das den id-Wert chap.xmlintro enthält. <xref/> wurde von DocBook als leeres Element deklariert, da keine direkte Textausgabe, sondern nur der Verweis von Bedeutung ist.

3.2.2 Das Schlüsselwort #PCDATA

Geht man einen Schritt weiter, kann man innerhalb der Start- und End-Tags Zeichendaten erlauben, jedoch keine Kindelemente. Dies wird durch das Schlüsselwort #PCDATA (*parsed character data*, das heißt vom Parser analysierte Zeichendaten) ausgedrückt. Somit bedeuten die Elementdeklarationen

```
<!ELEMENT stadt (#PCDATA)>
<!ELEMENT land  (#PCDATA)>
```

dass das Element stadt bzw. land nur Text enthalten darf, jedoch keine anderen Elemente. Allerdings kann der Inhalt auch leer sein, das heisst, dieser Code wäre erlaubt:

```
<stadt></stadt>
```

Die DTD kann nicht überprüfen, ob der Inhalt leer ist. Falls die für Sie von Bedeutung ist, müssen Sie entweder auf RELAX NG ausweichen, oder über XSLT diese Bedingung überprüfen.

3.2.3 Das Schlüsselwort ANY

Sollen alle Inhalte erlaubt sein, die in der DTD deklariert wurden, bietet sich das Schlüsselwort ANY an.

```
<!ELEMENT irgendwas ANY>
```

Innerhalb des Elements irgendwas sind andere Kindelemente, Zeichendaten und sogar andere irgendwas-Elemente erlaubt. Es ist immer noch erforderlich, dass mögliche Elemente an anderer Stelle deklariert wurden.

Dieses Inhaltsmodell wird nicht sehr häufig verwendet, da es allzu „lockere“ DTDs erzeugt. Für die Entwicklung einer DTD bietet sich *ANY* bisweilen an, um im Entwurfsstadium mit verschiedenen Deklarationen zu experimentieren. Fertige DTDs sollten dieses Schlüsselwort besser nicht enthalten, weil damit die Vorteile von DTDs, nämlich Restriktionen der Elemente zu bilden, um die Gültigkeit zu prüfen, quasi „ausgehebelt“ werden.

`%local.cd-cd.content;` ersetzt. Die eigentliche Deklaration vom Element `cd:cd` sieht wie folgt aus:

```
<!ELEMENT cd:cd (cd:title, cd:artist, cd:composer, cd:tracklist
             , cd:web?)>
```

Die letzte Zeile in fett ist der Inhalt Ihres definierten `%local.cd-cd.content;`.

Als letztes Beispiel für eine Anpassung werden die erlaubten Werte im Attribut genre erweitert. Vermutlich sind die Werte für Sie zu einschränkend. Wie Sie diese anpassen, zeigt folgender Code:

```
<!DOCTYPE cd-collection SYSTEM "cd.dtd"
 [
   <!ENTITY % local.cd.format.enum
     "| gothic | ...">
 ]>
<cd:cd-collection xmlns:cd="http://www.example.org/ns/cd">
 ...
</cd:cd-collection>
```

3.9 Zusammenfassung

In diesem Kapitel haben Sie einiges über DTDs erfahren:

- Die DTD-Syntax stammt aus der SGML-Zeit. Neuere Schemasprachen sind RELAX NG und W3C XML Schema.
- Die DTD-Syntax besitzt historisch gewisse Einschränkungen. Mehr Möglichkeiten bieten die im ersten Punkt erwähnten Schemasprachen.
- Eine DTD besteht hauptsächlich aus Element-, Attribut-, Notations- und Entitydeklaration sowie Beziehungen zwischen den Elementen.
- Die Beziehungen zwischen den Elementen werden durch ein *Inhaltsmodell* beschrieben.
- Elemente können leer sein oder nur Text oder andere Elemente (Kindelemente) enthalten.
- Das Inhaltsmodell eines Elements kann Elementfolgen, Wiederholungen, Gruppierungen oder einen gemischten Inhalt (Text und andere Elemente) enthalten.
- Ein Element kann Attribute enthalten, die über eine Attributdeklaration erstellt werden können.
- Attribute bestehen aus einem Attributnamen und einem Attributwert. Der Wert besteht aus einem Datentyp.
- Die DTD-Syntax erlaubt Entities, die Platzhalter für Daten sind.

- Allgemeine Entities sind sowohl in der DTD als auch im XML-Dokument erlaubt. Die Daten können sich innerhalb der DTD oder außerhalb (so genannt externe Entities) befinden.
- Parameter-Entities sind nur in der DTD erlaubt. Die Daten können sich innerhalb der DTD oder außerhalb (so genannt externe Parameter-Entities) befinden.
- Ungeparste Entities referenzieren Daten, die keinen Text enthalten.
- Innerhalb der DTD sind bedingte Abschnitte erlaubt. Sie ermöglichen über Schalter, Teile der DTD ein- oder auszublenden.

Kapitel 4
RELAX NG

Dieses Kapitel führt Sie in die Syntax von RELAX NG ein, einer modernen, leicht zu erlernenden Schemasprache in XML. DocBook 5 wurde in RELAX NG geschrieben. Das Verständnis dieser Schemasprache erleichtert Ihnen Ihre Anpassungen an die neueste Version.

Einfachheit ist das Resultat der Reife.
—Friedrich Schiller

Take a deep breath and RELAX!
—Unbekannt

4.1 Wozu RELAX NG?

Die DTD haben Sie im letzten Kapitel kennen gelernt. Es gibt noch weitere Schemasprachen, die bekanntesten sind RELAX NG, W3C-Schema und Schematron. RELAX NG steht für *Regular Language Description for XML (New Generation)* und ist eine Reaktion auf die Komplexität von W3C-Schema. RELAX NG entstand aus einem Zusammenschluss von RELAX von James Clark und TREX von Makoto Murata und ist seit 2003 als ISO-Standard 19757-2 veröffentlicht.

Die erwähnten Schemasprachen haben eine unterschiedliche Herangehensweise, um ein Inhaltsmodell festzulegen:

- *Als Beschreibung* DTDs und W3C-Schema drücken Beschränkungen aus, indem Sie das Inhaltsmodell von Elementen beschreiben und die zugehörigen Attribute angeben: „das Element `book` hat ein Attribut `id` und das Inhaltsmodell von `book` enthält…“
- *Durch Regeln* Schematron ist ein Beispiel, das Beschränkungen durch Regeln ausdrückt: „das Element `book` muss ein Attribut `id` enthalten und der Inhalt von

book passt auf die folgende Regel…“. Weitere Informationen finden Sie in Abschnitt 12.5.6, „Schematron“ (Seite 256).

- *Durch Muster (engl. patterns)* RELAX NG beschreibt die Beschränkungen durch Muster. Diese Muster müssen auf erlaubte Elemente, Attribute oder Textknoten passen, ähnlich regulären Ausdrücken in Programmiersprachen: „Es gibt ein Elementmuster das auf das Element book passt und dieses passt auf ein id-Attributmuster. Das Elementmuster passt auf weitere Muster, die wie folgt aussehen…“

Die folgende Liste zeigt Ihnen einige Vorteile von RELAX NG:

RELAX NG ist leicht zu erlernen
: Mit knapp 30 Elementen ist RELAX NG ein Leichtgewicht. Bereits wenige Elemente führen zu einem Schema, das leicht zu lesen und zu verstehen ist. Kennern von DTDs werden viele Konstrukte bekannt vorkommen.

RELAX NG hat zwei gleichwertige Formate (eine XML- und kompakte Syntax)
: Mit der ausführlicheren XML-Syntax lassen sich alle XML-Tools anwenden, wie Validieren oder Transformieren eines RELAX NG-Schema. Für Entwickler, die Kürze bevorzugen, gibt es die kompakte Schreibweise. Beide Formen lassen sich durch James Clarks `trang` (siehe Abschnitt 4.14) verlustfrei ineinander umwandeln.

RELAX NG wird für viele XML-Anwendungen eingesetzt
: RELAX NG wird mittlerweile gerne für die Entwicklung von populären XML-Anwendungen verwendet. Es wird verwendet für MathML 2.0, XHTML, TEI, P3P 1.0, SVG 1.1 und DocBook 5.

RELAX NG unterstützt Namensräume
: DTDs unterstützen keine Namensräume im Gegensatz zu RELAX NG. Durch Namensräume ermöglichen Sie, dass unterschiedliche RELAX NG-Schemata unterscheidbar bleiben.

RELAX NG unterstützt verschiedene Datentypen-Bibliotheken
: Standardmäßig besitzt RELAX NG zwei eingebaute Datentypen, und zwar String und Token. Jedoch kann RELAX NG mit jeder Datentyp-Bibliothek zusammenarbeiten, die im RELAX NG-Schema definiert ist und im RELAX NG-Validator implementiert wurde. In der Praxis bedeutet dies, dass die Datentypen des W3C-Schema (*XML Schema Datatypes*) von den meisten RELAX NG-Validatoren verstanden werden. Dadurch besitzen Sie eine Auswahl von über 40 Datentypen.

RELAX NG kann dokumentiert werden
: Standardmäßig liefert RELAX NG ein `annotation`-Element mit. Das zu beschreibende Element oder Attribut kann mittels XHTML oder einer beliebigen anderen Sprache dokumentiert werden.

RELAX NG lässt sich mit Schematron kombinieren
: Mittels Schematron, einer weiteren Schemasprache, lassen sich kontextabhängige Regeln definieren, die allein durch RELAX NG nicht auszudrücken sind. Beispielsweise das Berechnen von Prüfsummen, überprüfen einer Vielzahl von Knoten auf

ein gemeinsames Merkmal und noch viele andere. In diesem Buch wird Schematron lediglich sehr oberflächlich behandelt.

RELAX NG erlaubt Mehrdeutigkeiten und nicht-deterministische Inhaltsmodelle
: RELAX NG unterstützt nicht-deterministische Inhaltsmodelle und Mehrdeutigkeiten. In Abschnitt 3.2.10 ist diese Thematik genauer beschrieben.

4.2 Aufbau eines RELAX NG-Schema

Ein RELAX NG-Schema kann auf zwei Arten geschrieben werden: in einer kompakten Syntax oder in XML. Beide Formate sind gleichwertig und lassen sich verlustfrei ineinander umwandeln. Für die kompakte Syntax wird konventionell die Endung `.rnc` verwendet, wohingegen die XML-Form die Endung `.rng` erhält.

Der Aufbau Ihres RELAX NG-Schema ist zu einem gewissen Teil vom jeweiligen Format abhängig. Nicht alle der folgenden Bestandteile müssen enthalten sein. Dies hängt vom entsprechenden RELAX NG-Schema ab. Die Bestandteile sind:

1. Ein optionaler Kommentar am Anfang, der das RELAX NG-Schema beschreibt. In der kompakten Schreibweise verwenden Sie das #-Zeichen. Die XML-Form sollte mit einer XML-Deklaration beginnen, Kommentare werden durch die Zeichenfolge `<!-- ... -->` markiert.
2. Falls Ihr RELAX NG-Schema in einem Namensraum liegt, benötigen Sie eine oder mehrere Namensraumdeklarationen. Die kompakte Syntax verwendet hierfür das Schlüsselwort `namespace`. Die XML-Form verwendet dagegen das Attribut `ns` im Wurzelelement.
3. Falls Sie mit Datentypen arbeiten, benötigen Sie eine Deklaration Ihrer Datentyp-Bibliothek. Die kompakte Syntax verwendet hierfür das Schlüsselwort `datatypes`. Die XML-Form benötigt dagegen das Attribut `datatypeLibrary` im Wurzelelement.
4. Die eigentliche Struktur erfolgt über Muster, die in den folgenden Abschnitten beschrieben werden.

4.3 Die 3 Grundmuster: Text-, Element- und Attribut

Durch die Kombination von Element-, Attribut- und Textmuster zusammen mit Wiederholung und Gruppierung lassen sich äußerst flexible Inhaltsmodelle entwerfen. In RELAX NG gibt es 28 Muster, doch lediglich drei Grundmuster. Diese stimmen mit den drei Knotentypen überein, die in XML vorkommen dürfen:

- Textknoten
- Elemente
- Attribute

Kapitel 5
XML-Kataloge

Ein Katalog ist eine XML-Datei, die URIs mit lokale Pfade verknüpft. Wird ein URI gesucht, wird sie durch den lokalen Pfad im Katalog ersetzt. Dadurch müssen Sie keine zeitraubende Internetverbindung zum Server aufbauen. Das vorliegende Kapitel zeigt Ihnen die Vorteile von Katalogen auf und erklärt Ihnen, wie Sie Kataloge erstellen, verwalten und abfragen.

Jedes Fragen ist ein Suchen.
—Martin Heidegger

5.1 Wozu Kataloge?

Ressourcen über einen Pfad anzusprechen bereitet eine Reihe von Schwierigkeiten:

- Verzeichnisstrukturen sind abhängig vom Betriebssystem sehr unterschiedlich: Linux verwendet den Schrägstrich, DOS/Windows den Backslash und MacOS vor Version 10 den Doppelpunkt.
- Manche Betriebssysteme haben Vorgaben: sie beschränken die Länge der Dateinamen oder unterscheiden nicht zwischen Groß- und Kleinschreibung.
- Werden Dokumente mit anderen ausgetauscht, liegen die Daten beim Empfänger an einer anderen Stelle im Dateisystem als beim Absender.

Diese Probleme werden durch Kataloge gelöst. Ihre XML-Umgebung wird anpassungsfähig und zwar hauptsächlich aus den folgenden Gründen:

Verarbeitungsgeschwindigkeit
: Vereinfacht gesagt, ersetzen XML-Kataloge Internetadressen durch betriebssystemabhängige Pfade. Dadurch wird die entsprechende Ressource von Ihrer lokalen Festplatte geladen, wodurch sich die Verarbeitungsgeschwindigkeit erhöht. Vorausgesetzt, Sie besitzen eine lokale Kopie.

Austauschbarkeit
: Sollen XML-Dokumente auf verschiedenen Betriebssystemen verarbeitet werden, entstehen Schwierigkeiten durch unterschiedliche Pfade oder eine unbekannte Syntax. Allgemeine URIs verlagern den betriebssystemabhängigen Teil in Kataloge und vermindern diese Probleme beim Austausch.

Zentralisierung
: Durch Kataloge haben Sie *eine* zentrale Stelle, um generische URIs zu konfigurieren, um sie für alle Ihre XML-Dateien und Dokumente zu verwenden.

Kataloge werden durch diese Eigenschaften für einige interessante Szenarien eingesetzt:

- Lassen Sie einen generischen URI unterschiedlich auflösen, indem Sie zur Laufzeit dem XML-Parser verschiedene Kataloge zuweisen.
- Verwenden Sie in Makefiles, DTDs, Stylesheets und anderen Dateien einen konstanten URI, der unempfindlich gegen Verschieben in einen anderen Pfad oder auf einen anderen Rechner ist.
- Testen Sie neue Versionen von DTDs, Schemata oder Stylesheets, indem Sie im Katalog den URI auf den neuen Pfad zeigen lassen.

Der Aufbau eines XML-Katalogs sieht wie folgt aus:

Beispiel 5.1. *Aufbau eines XML-Katalogs*

```
<?xml version="1.0"?>
<!DOCTYPE catalog PUBLIC
 "-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN"
 "http://www.oasis-open.org/committees/entity/release/1.0/catalog.dtd"> ❶
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"> ❷
  <!-- Einträge --> ❸
</catalog>
```

❶ Die optionale DOCTYPE-Deklaration. Manche XML-Parser haben die Katalog-DTD eingebaut und brauchen keine Internetverbindung anfordern. Bei Fehlern entfernen Sie die DOCTYPE-Deklaration.
❷ Das Wurzelelement `catalog` ist an den Namensraum `urn:oasis:names:tc:entity:xmlns:xml:catalog` gebunden.
❸ Tragen Sie an dieser Stelle Ihre Einträge ein, die URIs in die entsprechenden Pfade umschreiben.

5.2 Ein erstes Beispiel

In Beispiel 5.1, „Aufbau eines XML-Katalogs“ (Seite 100) haben Sie bereits die Struktur eines XML-Kataloges gesehen. Zwischen `<catalog>` und `</catalog>` sind verschiedene XML-Elemente erlaubt, die in folgender Tabelle aufgelistet werden. Fettgedruckte Attribute sind anzugeben, normalgeschriebene sind optional. Jedes der gezeigten Elemente erlaubt zusätzlich die optionalen Attribute `id` und `xml:base`. Ersteres dient

zum Identifizieren des Elements, letzterer verändert den Basis-URI (ähnlich dem BASE-Element von HTML).

Tabelle 5.1. *Erlaubte Einträge in XML-Katalogen (Version 1.0)*

Element	*Beschreibung/Syntax*
delegatePublic	identifiziert einen Teil eines öffentlichen Bezeichners und delegiert den Ersetzungsprozess an einen anderen XML-Katalog <delegatePublic **publicIdStartString**="*PUBLIC_IDENTIFIER_PRÄFIX*" **catalog**="*URI*"/>
delegateSystem	identifiziert einen Teil eines Systembezeichners und delegiert den Ersetzungsprozess an einen anderen XML-Katalog <delegateSystem **systemIdStartString**="*STRING*" **catalog**="*URI*"/>
delegateURI	identifiziert einen Teil eines URIs und delegiert den Ersetzungsprozess an einen anderen XML-Katalog <delegateURI **uriIdStartString**="*STRING*" **catalog**="*URI*"/>
group	gruppiert Einträge und spezifiziert einen Basis-URI <group prefer="public|system" > ... </group>
nextCatalog	lädt einen weiteren Katalog <nextCatalog **catalog**="*URI*"/>
public	identifiziert einen kompletten öffentlichen Bezeichner <public **publicId**="*PUBLIC_IDENTIFIER*" **uri**="*URI*"/>
rewriteSystem	identifiziert einen Teil eines Systembezeichners und ersetzt ihn durch eine URI-Referenz

II
DocBook-Markup

Kapitel 6
Übersicht über DocBook

In diesem Kapitel erfahren Sie was DocBook ist, wofür es eingesetzt werden kann und wo nicht und welche Vorteile es hat. Des Weiteren lernen Sie, welche Schemata verfügbar sind und was es für Unterschiede zwischen Version 4 und 5 gibt.

> *The Source for Documentation*
> —DocBook.org

6.1 Was ist DocBook?

Die DocBook-SGML-DTD wurde im Jahre 1991 als ein Gemeinschaftsprojekt von HaL Computer und O'Reilly ins Leben gerufen und hatte zum Ziel, den Austausch von UNIX™-Dokumentationen zu fördern. DocBook beschreibt sich selbst wie folgt:

> DocBook ist ein Schema, entwickelt vom Technischen DocBook Ausschuss von OASIS und erhältlich als RELAX NG, SGML/XML-DTD und W3C XML Schema. DocBook ist besonders gut geeignet für Bücher und Dokumente über Computer-Hard- und Software, ist jedoch keineswegs nur auf diese Anwendungen beschränkt.
> —`http://www.docbook.org/whatis`

Bis zur Version 3.x war DocBook ausschließlich als SGML-DTD verfügbar. In Version 4.0 wurde erstmals eine Implementierung als XML-DTD geschaffen. Seit 2001 gibt es DocBook 4.x als inoffizielles W3C Schema, RELAX NG und Schematron.

Mit der Version 5.0 wurde DocBook als RELAX NG-Schema neu implementiert und davon DTD und W3C-Schema abgeleitet. In diesem Buch werden die XML-DTD und RELAX NG-Version beschrieben. Weitere geschichtliche Informationen erhalten Sie in dem Artikel unter `http://www.xml.com/lpt/a/176`.

6.2 Sinnvolle Dokumentation mit DocBook

Ob DocBook für Sie sinnvoll ist hängt davon ab, welche Art von Dokumentation Sie erstellen. DocBook ist am besten geeignet für folgende Typen von Dokumentationen:

- Technische Dokumentation aller Art, wie Handbücher für Hard- und Software, Spezifikationen und Whitepapers
- Manpages
- Howtos
- Trainingshandbücher

Selbst die folgenden Typen sind mit DocBook möglich (`http://wiki.docbook.org/topic/WhoUsesDocBook`)

- Präsentationen
- Webseiten
- Novellen
- Biographien
- Tagebücher

Nicht die beste Wahl ist DocBook für folgende Dokumente:

- Visitenkarten
- Korrespondenzen
- Poster
- Aufwändige, layoutintensive Broschüren

Die vorige Liste muss nicht als absolut angesehen werden. Abhängig von Art und Umfang Ihres Dokuments und mit genügend Aufwand sind obige Dokumententypen ebenso möglich.

6.3 Schemadschungel

Offiziell unterstützt das DocBook-Komitee DTD und RELAX NG. W3C-Schema ist lediglich aus Komfortgründen hinzugefügt worden. Alle Schemasprachen sind identisch hinsichtlich Struktur und Namen der Elemente und Attribute, unterscheiden sich jedoch in ihrer Ausdrucksfähigkeit (dazu gleich mehr).

Bevor Sie beginnen, entscheiden Sie sich für ein Schema, das Sie als Grundlage Ihrer Dokumente verwenden möchten. DocBook gibt es in den folgenden Varianten:

DTD (*Document Type Definition*)
: Als älteste und verbreiteste Schemasprache ist die DTD bei XML-Editoren oder Parser bestens unterstützt. Bis zur Version 4.5 wurde DocBook als DTD formuliert, ab 5.0 als RELAX NG-Schema.

Eine DTD ist eine grammatikbasierte[1] Schemasprache. Sie unterstützt keine Namensräume und besitzt ein nur auf Attribute anwendbares, schwach ausgebildetes Datentypsystem.

Verwenden Sie eine DTD, wenn Sie keine Datentypen benötigen und Programme besitzen, die nur DTDs kennen.

RELAX NG
: RELAX NG ist eine leicht zu erlernende und zu schreibende grammatikbasierte Schemasprache. Sie unterstützt Datentypen sowohl in Elementinhalt als auch in Attributen. Von manchen XML-Experten wird sie als einfachere und zuverlässigere Sprache angesehen als W3C-Schema. Zusätzlich hat sie eine äquivalente, kompakte Schreibweise.

  DocBook 5 wurde wegen dieser Fähigkeiten offiziell in RELAX NG formuliert. Verwenden Sie diese Sprache, wenn Ihre Programme RELAX NG unterstützen, Sie eine strengere Kontrolle Ihrer Dokumente benötigen und wenn Sie andere RELAX NG-Schemata in DocBook 5 integrieren möchten.

Schematron
: Schematron nimmt eine Sonderstellung ein. Es ist eine regelbasierte[2] Schemasprache, mit Bezug zu XPath und XSLT. Es überprüft ein XML-Dokument durch so genannte *kontextabhängige Zusicherungen* (engl. *assertions*). Schematron wird seltener als eigenständige Schemasprache verwendet, sondern mehr als Ergänzung zu RELAX NG oder W3C Schema gesehen. Aufgrund der unterschiedlichen Syntax lassen sich Schematron und DTD nicht in derselben Datei zusammen nutzen.

  Für DocBook 5 bedeutet dies, dass ein Dokument nicht separat mit Schematron validiert wird, sondern stets zusammen mit RELAX NG oder W3C-Schema. Beide DocBook 5-Schemata enthalten Schematron-Anweisungen innerhalb der DocBook-Elemente. Durch Schematron-Anweisungen werden Bedingungen überprüft, die durch RELAX NG oder W3C-Schema allein schwierig oder unmöglich auszudrücken sind. Beispielsweise erfordert DocBook 5 ein Attribut `version` im Wurzelelement. Mittels W3C- oder RELAX NG-Schema ist diese Bedingung zwar zu überprüfen, jedoch ist es eine sehr mühsame Angelegenheit alle möglichen Kombinationen abzudecken. Für diesen Zweck wird Schematron eingesetzt.

W3C-Schema
: Die W3C XML Schemasprache ist eine sehr stark typisierte Schemasprache. Sie beschreibt Dokumente aus einem objektorientierten Blickwinkel. W3C-Schema enthält Grundtypen aus denen sich komplexere Datentypen zusammenstellen lassen. Des Weiteren erlaubt diese Schemasprache namensraumsensitive Element- und Attributdeklarationen.

---

1 Grammatikbasierte Schemasprachen konzentrieren sich auf die Struktur der Instanzdokumente und formulieren Regeln um diese Struktur zu beschränken („Element `A` besteht aus …“).

2 Regelbasierte Schemasprachen spezifizieren Bedingungen zwischen Elementen und Attributen, die sich durch eine Regel oder Algorithmus formulieren lassen („ist das Attribut `version` im Wurzelelement vorhanden?“)

Verwenden Sie W3C-Schema dann, wenn Sie Programme besitzen, die RELAX NG-Schema nicht kennen, jedoch die selben Möglichkeiten wie bei einem RELAX NG-Schema benötigen.

Unterschiede in der Validierung
Es gibt Validierungsunterschiede, die weniger in DocBook selbst, als in der jeweiligen Schemasprache liegen (beispielsweise bei Datentypen). Dies kann dazu führen, dass Dokumente bei der Validierung mit einer DTD gültig sind, jedoch nicht mit einem RELAX NG oder W3C-Schema – und umgekehrt. Aus diesem Grund validieren Sie ein DocBook 5 stets mit dem RELAX NG- oder W3C-Schema.

Welche Schemasprache und welche Version sollten Sie wählen? Da Dokumentation, Systemumgebung, Zielsetzung und Anforderungen zu unterschiedlich sind, gibt es keinen allgemeingültigen, „richtigen" Weg. Die zwei Aspekte Version und Schemasprache sind in der folgenden Liste beschrieben:

Version: 4.x oder 5.x?
: DocBook 5 bietet gegenüber DocBook 4 einige Vorteile die in Abschnitt 6.5, „Unterschiede zwischen DocBook 4 und 5" (Seite 116) gezeigt werden. Für neue Projekte sollten Sie Version 5 bevorzugen, da Version 4 nicht mehr weiterentwickelt wird. Das bedeutet, neue Elemente werden nur in der Version 5 und später eingefügt. Bei älteren Projekten stellt sich die Frage, ob Sie die Vorteile der neueren Version nutzen möchten und Ihre alten Dokumente migrieren. Abschnitt 22.1, „Von DocBook 4 nach DocBook 5" (Seite 541) zeigt Ihnen, wie dies funktioniert.

Schemasprache: DTD, RELAX NG oder W3C-Schema?
: Da Ihre Dokumente nicht losgelöst von der Verarbeitung sind, bestimmen Ihre eingesetzen Programme zu einem gewissen Teil die Schemasprache. Wenn Ihr XML-Editor kein RELAX NG unterstützt, ist es sinnlos als Basis für Ihre Dokumente RELAX NG zu wählen. Sind Sie jedoch frei von diesen Einschränkungen sind die modernen Schemasprachen RELAX NG oder W3C-Schema meist die bessere Wahl. Sie erlauben eine exaktere Kontrolle Ihres Inhalts anhand von Datentypen oder „Kontrollanweisungen" in Form von Schematron. Zudem ist es mit RELAX NG sehr einfach, DocBook an Ihre Bedürfnisse anzupassen.

6.4 Verweis auf ein DocBook 5-Schema

Um ein Dokument mit dem entsprechenden DocBook-Schema zu verknüpfen, existieren je nach Schemasprache unterschiedliche Methoden:

DTD
: Verwenden Sie eine DocBook-DTD benötigen Sie eine DOCTYPE-Deklaration mit den entsprechenden Bezeichnern. Der Aufbau ist stets gleich und unterscheidet sich nur durch die DocBook-Version.

Kapitel 7
Strukturelemente

Durch die Vielzahl verfügbarer Elemente in DocBook kann leicht die Übersicht verloren gehen. Dieses Kapitel gruppiert die Elemente nach Themen und beschreibt Sie in einem größeren Zusammenhang, um dabei die Herangehensweise Ihnen nahezubringen. Dabei werden grundlegenden Fragen beantwortet, wie Bücher, Artikel oder Tabellen in DocBook eingegeben werden. Das Kapitel zeigt Ihnen auch die Unterschiede zwischen DocBook 5 und früheren Versionen.

> *Jeder Geist baut sich selbst ein Haus*
> *und jenseits dieses Hauses eine Welt*
> *und jenseits dieser Welt einen Himmel.*
> —Ralph Waldo Emerson

7.1 Über die Darstellung der DocBook-Elemente

DocBook-Elemente sind Bedeutung ohne Aussehen. Als Autor fügen Sie Ihrem Text Elemente aufgrund ihrer Bedeutung hinzu, nicht aufgrund ihrer Darstellung. Die Darstellung des Dokuments erfolgt separat über ein so genanntes *Publishing-System*.

Dieses Buch versteht unter diesem Begriff den Vorgang und die beteiligten Anwendungen, um aus DocBook ein anderes Format zu generieren: Das können *Stylesheets* sein, die über einen Befehl in einer Shell manuell aufgerufen werden, oder eine grafische Oberfläche, die den Transformationsprozess über ein Menü oder Dialogfenster steuert.

Damit ein Autor zumindest eine grobe Vorstellung über die Darstellung im Endformat hat, gruppiert DocBook seine Elemente in drei Kategorien:

Inline-Elemente
: Inline-Elemente erscheinen in einem gemischten Inhaltsmodell, das heisst, wo Text und weitere Elemente erlaubt sind. Ein Beispiel ist `emphasis` (hebt Text hervor) innerhalb von `para`. Inline-Elemente haben normalerweise keinen Zeilenumbruch,

jedoch ändert sich stellenweise Schriftfamilie, Schriftstil oder beides. Viele Inline-Elemente werden identisch dargestellt.

Block-Elemente
: Block-Elemente fassen größere Einheiten zusammen, wie beispielsweise ein Kapitel (`chapter`). Sie zeichnen sich für gewöhnlich dadurch aus, dass sie zusätzlich mit einen Zeilenumbruch, größeren Abstand oder sogar Seitenumbruch formatiert werden. Größere Einheiten wie Kapitel, Anhang usw. erscheinen normalerweise auf ungeraden Seiten. Manche Block-Elemente enthalten einen Titel der in einem Verzeichnis aufgenommen wird.

Unterdrückte Elemente
: Unterdrückte Elemente sind weder Inline- noch Block-Elemente, sie werden überhaupt nicht angezeigt. Für gewöhnlich sind es Metainformationen (wie `refmeta` für einen Referenzeintrag), Steuerungselemente (wie `areaspec` zum Sammeln und Referenzieren von Bereichen für so genannte „Callouts“) oder Elemente die nicht an der angegebenen Stelle erscheinen, sondern woanders (wie `indexterm`). Je nach Stylesheet können jedoch auch unterdrückte Elemente verwendet werden, um zusätzliche Informationen darzustellen. Dann ist ihre Darstellung allerdings standardmäßig undefiniert und anwendungsspezifisch.

Der Abschnitt *Processing expectations* der DocBook-Referenz klassifiziert ein Element in einer der obigen drei Kategorien. Dieses „Standardverhalten“ wird in den DocBook XSL Stylesheets implementiert.

Haben Sie eigene Layoutvorstellungen, müssen Sie die DocBook-Stylesheets anpassen. Sie sollten jedoch nicht ohne triftigen Grund von der Empfehlung abweichen, um Anwender nicht zu verwirren.

In den folgenden Abschnitten wurde in den meisten Beispielen daher auf eine Darstellung des Endergebnisses verzichtet. Wie DocBook in andere Formate umgewandelt werden kann, ist Thema in Teil III, „DocBook-Transformationen“ (Seite 287).

7.2 Allgemeine Attribute

DocBook stellt Attribute bereit, die in jedem Element verfügbar sind und in der folgenden Liste aufgeführt werden. Für gewöhnlich sind die Attribute optional. In manchen Elementen werden einige davon jedoch erwartet.

Die Attribute enthalten weitere Informationen zu einem Element wie Sprache, Zielgruppe, Sicherheit, Schwierigkeitsgrad und andere. Dadurch kann ein Element zusätzlich durch diese Attribute von einem anderen, gleichnamigen Element unterschieden werden. Dies ist manchmal praktisch, wenn Sie bestimmte Abschnitte nach Schwierigkeitsgrad filtern, oder Sie eine Übersicht über alle englischsprachigen Zitate in Ihrem Dokument erstellen möchten.

Allgemeine Attribute für alle DocBook-Elemente der Version 4 und 5

`arch` (Architektur)
: spezifiziert die Chip- oder Computerarchitektur eines Elements, wie `i386`, `x86_64` usw.

`audience` (Zielgruppe)
: spezifiziert die vorgesehene Zielgruppe wie Programmierer, Systemadministrator oder Anfänger.

`condition` (Bedingung)
: Eingeführt zur Version 4.0, wird es als „Allzweck-Attribut" verwendet und enthält anwendungsspezifische Bedingungen.

`conformance` (Konformität)
: bezeichnet einen Standard (ISO, IETF, DIN usw.) dem das Zielelement des Attributs entspricht; wird nicht sehr häufig verwendet.

`dir` (Schreibrichtung)
: markiert die Schreibrichtung, die sich gegebenenfalls überschreiben lässt. Dieses Attribut ist erst ab einschließlich DocBook V4.3 verfügbar. Für gewöhnlich wird dieses Attribut selten benötigt und nur verwendet, wenn Ihr Publishing-System die Schreibrichtung nicht selbst aus dem Unicode-Zeichen ermitteln kann.

`id` (Anker, Markierung, Kennung)
: markiert ein Element, um darauf verweisen zu können. Dieses Attribut ist nur für DocBook 4 und früher verfügbar. In DocBook 5 wird `id` durch `xml:id` ersetzt. Die meisten Elemente benötigen dieses Attribut nicht, jedoch gibt es einige wenige, für die es erforderlich ist.

Erlaubte Zeichen im Attributwert
Obwohl XML kaum die erlaubten Zeichen beschränkt, sollten Sie sich dennoch auf Buchstaben (a-z, A-Z), Ziffern, den Unterstrich und den Punkt beschränken. Die DocBook-XSLT-Stylesheets können aus dem `id`-Attributwert Teildateien für (X)HTML erzeugen. Manche XSLT-Tools haben Probleme beim Erzeugen dieser Teildateien, wenn Sie Umlaute enthalten.

`lang` (Sprache)
: kennzeichnet den Sprachcode, wie `de` für Deutsch, `en` für Englisch usw. Zusammengesetzte Sprachcodes aus Sprache und Land die ein Bindestrich haben, werden automatisch in einen Unterstrich umgewandelt, das heisst, aus `de-DE` wird `de_DE`. Weitere finden Sie unter der Adresse `http://www.evertype.com/standards/iso639/iso639-en.html`. Dieses Attribut ist nur für DocBook 4 und früher verfügbar. In DocBook 5 wird `lang` durch `xml:lang` ersetzt.

  Wird keine Sprache angegeben, ist normalerweise Englisch die Standardsprache. Dies kann jedoch von den verwendeten Programmen abhängig sein. Gegenwärtig werden von den DocBook-XSL-Stylesheets alle handelsüblichen Sprachen erkannt.

Kapitel 8
Block-Elemente

Block-Elemente verbinden größere Einheiten zusammengehöriger Informationen. Dieses Kapitel zeigt Ihnen, wie Sie Tabellen, Grafiken, Handlungsanweisungen und noch viele weitere Elemente nutzen.

> *Vier Zeilen in einem guten Lexikon sind mehr wert als der schönste Grabstein.*
> — Sir Alec Guinness

8.1 Versionsgeschichten

Mit Hilfe des Elements `revhistory` erstellen Sie eine so genannte „Versionsgeschichte". Für gewöhnlich wird `revhistory` als Metainformation verwendet (erlaubt in `info`, `bookinfo`, `partinfo`, `chapterinfo` usw.) Setzen Sie dieses Element ein, wenn Sie Ihre Leser über die Änderungen im entsprechenden Element informieren möchten. Ein Beispiel, wie Sie `revhistory` einsetzen:

Beispiel 8.1. *Eine typische Versionsgeschichte mit revhistory*

```
<revhistory>
  <revision>❶
    <revnumber>1.0</revnumber>❷
    <date>Okt. 2002</date>❸
    <authorinitials>tux</authorinitials>❹
    <revdescription>❺
      <para>...</para>
    </revdescription>
  </revision>
  <!-- evtl. weitere <revision> Elemente  -->
</revhistory>
```

❶ Das Elternelement `revhistory` enthält mindestens ein Element `revision`.
❷ Die Revisionsnummer. Dies kann ein beliebiger Wert sein.
❸ Datum zur Revisionsnummer
❹ Initialen des Autors
❺ längere Beschreibung der Revision; für kürzere Texte ist auch `revremark` möglich.

Aufgrund eines Versehens bis zur Version 4.2 kann dieses Element auch an einigen anderen „exotischen" Stellen auftreten, wie in `seealso`, `lineannotation`, `title`, `link` und `quote`. Verwenden Sie es dort *nicht*, da es in DocBook 5 entfernt wurde.

Automatische Versionsgeschichte mit Subversion
Verwenden Sie Subversion als Versionsverwaltung, lässt sich eine Versionsgeschichte automatisch erstellen. Gerade wenn sich Ihre Dokumente häufig ändern, ist dies eine große Arbeitserleichterung. Unter `https://docbook.svn.sourceforge.net/svnroot/docbook/trunk/releasetools/svnlog2docbook.xsl` gibt es für interessierte Anwender ein Stylesheet, das die Ausgabe von `svn log` in ein `revhistory` überführt. Mit Hilfe von XIncludes oder externe Entities lässt sich die so erzeugte Datei an der gewünschten Stelle einbinden.

Eine weitere, dazu passende Ergänzung ist das Attribut `revision`. Dieses Attribut ist in jedem Element verfügbar (Version 4 bzw. 5). Wenn Sie Ihre Texte mit Hilfe CVS oder Subversion verwalten, lässt sich die aktuelle Version in einem `revision`-Attribut unterbringen. Fügen Sie eine Markierungen wie `$Id:$`, `$Date:$` oder `$Revision:$` hinzu, ergänzt das Versionsverwaltungssystem dies durch den den oder die aktuellen Werte:

```
<chapter revision="$Id: db-elements-block.xml 973 2009-07-12 19:20:52Z tom $">
```

8.2 Tabellen

DocBook besitzt zwei Elemente für Tabellen: `table` und `informaltable`. Nur das erste Element hat einen Titel. Das Inhaltsmodell kann jedoch, je nach DocBook-Version, unterschiedlich sein. In DocBook gibt es folgende Tabellenmodelle:

- Das CALS-Tabellenmodell (alle DocBook-Versionen, Voreinstellung in DocBook 4)
- Das (X)HTML-Tabellenmodell (DocBook >=4.3, 5.0)
- Das OASIS XML Exchange Tabellenmodell (nicht DocBook 5)

Der nächste Abschnitt befasst sich mit den Unterschieden der verschiedenen Modelle. Welches Modell Sie verwenden, ist teilweise eine Geschmacksfrage. Da das CALS-Tabellenmodell der Standard ist und in beiden DocBook-Versionen verwendet wird, wird es in den folgenden Abschnitten genauer betrachtet. Anwender die das (X)HTML-Tabellenmodell bevorzugen müssen die dortigen Beschreibungen an ihre Bedürfnisse übertragen.

Kapitel 9
Inline-Elemente

Inline-Elemente erscheinen im Textfluss und zeichnen Dateinamen, Produkte, Personen, Tastenkombination und vieles mehr aus. Damit die Vielzahl der Elemente einigermaßen überschaubar bleibt, wurden sie in verschiedene Kategorien gruppiert.

Aus kleinem Anfang entspringen alle Dinge.
—Marcus Tullius Cicero

9.1 Programmspezifische Elemente

Zu dieser Kategorie[1] zählen folgenden Elemente:

`classname`, `exceptionname`, `interfacename`, `methodname`
: Diese Elemente werden eingesetzt, um objektorientierte Programme, Sprachen oder Architekturen zu dokumentieren:

```
<para>Die Klassenbibliothek Qt besitzt die
  Klasse <classname>QSvgWidget</classname> mit der
  Methode <methodname>load</methodname>. XML-Exceptions werden
  durch <exceptionname>QXmlParseException</exceptionname>
  behandelt.
</para>
```

`function`, `parameter`, `returnvalue`
: Zum Beschreiben von Funktionen, Parametern und Rückgabewerten:

```
<para>Python erwartet in der Funktion
  <function>lexists</function> den Parameter
  <parameter>path</parameter> und gibt als Rückgabewert
  <returnvalue>True</returnvalue> oder
```

1 Die Kategorien sind nicht willkürlich, sondern stammen aus dem DocBook-RELAX NG-Schema.

```
    <returnvalue>False</returnvalue> zurück.
  </para>
```

varname
: Der Name einer Variablen:

```
<para>In Python enthält <varname>sys.argv</varname>
  die Kommandozeilenparameter beim Aufruf des Skripts.
</para>
```

9.2 Betriebssystemspezifische Elemente

Zu dieser Kategorie zählen die folgenden Elemente (verfügbar in beiden DocBook-Versionen):

command
: Enthält der Name eines auszuführenden Befehls:

```
<para>Der Befehl <command>cp</command> (Linux)
   oder <command>copy</command> (Windows) kopiert
   Dateien und Verzeichnisse.
</para>
```

computeroutput
: Enthält Daten die durch ein Computer angezeigt werden:

```
<para>Die Prüfsumme der Datei <filename>foo.xml</filename> ist:
<computeroutput>800bb5d71d532942eeaab23cf964170b foo.xml</computeroutput>
</para>
```

envar
: Dieses Element enthält den Namen einer Umgebungsvariable:

```
<envar>PATH</envar>
```

filename
: Für gewöhnlich enthält es Datei- oder Verzeichnisnamen:

```
<para>Unter <filename class="directory">/home</filename> finden Sie
das Homeverzeichnis der lokalen Anwender.</para>
```

  Über das Attribut `class` können Sie den Inhalt weiter spezifizieren:

`devicefile`	Gerätedatei: `/dev/mouse`
`directory`	Verzeichnis: `/etc`, `C:\Programme`
`extension`	Dateierweiterung: `.xml`
`headerfile`	Headerdatei: `stdio.h`
`libraryfile`	Bibliotheksdatei: `libglib-2.0.so`, `win.dll`

Kapitel 10
Querverweise und Links

Querverweise und Links ermöglichen dem Leser, sich weiter über ein Thema zu informieren. DocBook ermöglicht Ihnen als Autor, Elemente in Ihr Dokument zu integrieren, mit denen sich diese Verweise beschreiben lassen.

Der Weg ist das Ziel.
—Konfuzius

10.1 Warum Verweise?

Verweise verbinden einen Ursprung mit einem Ziel. Hierbei gibt es zwei Methoden: Sie fügen Ihren Verweis textuell ein oder sie verwenden ein entsprechendes DocBook-Element, das beim Verarbeiten aufgelöst wird. Warum Sie letzteres nutzen sollten, beschreibt Ihnen die folgende Liste:

- *Einfachere Pflege* Wird eine Information an mehreren Stellen benötigt, vereinfacht ein Verweis die Pflege Ihres Dokuments. Wird die Information geändert, passen sich die Verweise automatisch an, ohne dass Sie manuell eingreifen müssen.
- *Aktuellere Informationen* Ein textuell eingefügter Verweis auf einen Inhalt ist problematisch, wenn der Inhalt geändert oder gelöscht wird. Ein Verweis über ein DocBook-Element bleibt immer aktuell, selbst wenn Sie die Information ändern.
- *Bessere Fehlererkennung* Ein Verweis lässt sich beim Validieren oder Transformieren überprüfen. Dadurch erkennen Sie eventuelle Fehler bereits beim Verarbeiten.
- *Flexiblere Darstellung* Verweise passen sich an individuelle Wünsche an. Zum einen lassen sie sich für das komplette Dokument oder individuell angleichen. Zum anderen passen sich Verweise auf das Zielformat an: beispielsweise ist eine Seitenzahl für ein Onlineformat sinnlos, während dies für gedruckte Dokumente erforderlich ist.

- *Reduziert Inhaltsverdopplungen* Eine Information braucht nur einmal beschrieben werden, lässt sich jedoch beliebig oft referenzieren. Dadurch reduzieren Sie unnötige Verdopplungen.
- *Verbesserte Benutzerfreundlichkeit* Durch Verweise verbessern Sie Ihr Dokument für Ihre Endanwender. Onlineformate erlauben, Verweise zu verfolgen, während dies bei einem reinen Text nicht möglich ist.

10.2 Interne Querverweise

Ein interner Querverweis verbleibt innerhalb des selben Dokuments. Beim Validieren überprüft der XML-Parser, ob der Verweis vom Typ IDREF mit dem Ziel vom Typ ID übereinstimmt. Ist dies nicht der Fall, ist Ihr Dokument ungültig.

Verweis und Verweisziel dürfen sich in verschiedenen Dateien befinden, wenn Sie Ihr Dokument modularisieren. Dennoch bleibt auch in diesem Fall dieser Verweis ein interner Querverweis. Wichtig ist, dass beim Zusammenführen der Verweis auf ein Verweisziel innerhalb des Dokuments zeigt.

DocBook besitzt die folgenden Querverweiselemente:

`biblioref`	ein Querverweis auf einen bibliografischen Eintrag
`coref`	ein Querverweis auf ein co-Element, siehe Abschnitt 8.5.1 (Seite 163)
`footnoteref`	ein Querverweis auf eine Fußnote (`footnote`)
`xref`	ein allgemeiner Querverweis. Verwenden Sie `xref`, wenn die obigen Elemente für Sie nicht erforderlich oder unerwünscht sind

Alle diese internen Querverweise sind leere Elemente. Sie besitzen drei wichtige Attribute, mit denen Sie den Querverweistext beeinflussen:

`linkend` (erforderlich)	Verweist auf ein Element mit einem Anker, der im selben Dokument sein muss. Der Anker und der Attributwert von `linkend` müssen identisch sein, andernfalls ergibt dies einen Validierungsfehler.
`endterm` (optional)	Zeigt auf den Anker eines Elements, bei dem der Inhalt als Querverweistext verwendet wird. Wird eher selten verwendet.
`xrefstyle` (optional)	Beeinflusst die Darstellung des Querverweis-Elements. Der Inhalt ist abhängig von Ihrem Publishing-System. Mehr zu diesem Thema finden Sie in Abschnitt 16.6.2, „Anpassen mittels xrefstyle-Attribut“ (Seite 391)

10.2.1 Einfache Querverweise

In den meisten Fällen reicht es aus, wenn Sie:

1. Einen Anker an die Stelle setzen, auf die Sie verweisen möchten.

Kapitel 11
Modulare Dokumente

Ein DocBook-Dokument lässt sich als Ganzes schreiben, oder in kleinere Dateien zerlegen und später zusammensetzen. Das vorliegende Kapitel beschreibt die Methode über externe Entities und XInclude.

> *Es gibt ein unfehlbares Rezept, eine Sache gerecht unter zwei Menschen aufzuteilen: Einer von ihnen darf die Portionen bestimmen, und der andere hat die Wahl.*
> —Gustav Stresemann

11.1 Wozu aufteilen?

Ein DocBook-Dokument in kleinere Dateien aufzuteilen, um sie bei der Veröffentlichung wieder zusammenzuführen, besitzt einige Vorteile:

- Aufteilung in kleinere Dateien erlaubt, dass mehreren Autoren gleichzeitig an einem Dokument schreiben, ohne sich gegenseitig zu behindern.
- Das Dokument wird bei cleverer Strukturierung und Benennung der Dateien übersichtlicher.
- Kleinere Dateien verringern die Ladezeit in Ihrem Editor.
- Genauere Versionskontrolle, die sich an den Dateien orientiert und nicht am kompletten Dokument.
- Dateien können vor dem Zeitpunkt der Zusammenführung dynamisch erzeugt oder ergänzt werden.
- Wiederverwendbare Dateien werden nur einmal angelegt, können jedoch beliebig oft referenziert werden.

Es gibt zu den oberen Vorteilen nur einige kleinere Einschränkungen: Eine Datei mit dateiübergreifenden Querverweisen ist nicht einzeln zu validieren, sondern nur im Verbund. Der Querverweis kann in der einzelnen Datei nicht aufgelöst werden und

sorgt für einen Validierungsfehler. Dieser „Fehler“ verschwindet, wenn Sie Ihr komplettes Dokument validieren.

Des Weiteren sind dateiübergreifende Querverweise manchmal ein Problem für XML-Editoren. Je nach Funktionsweise betrachtet dieser häufig nur die Datei, nicht jedoch das komplette Dokument. Dadurch werden Verweise auf Markierungen nicht aufgelöst und erzeugen im Editor einen Validierungsfehler.

11.2 Aufteilungsmethoden von Dokumenten

In der Praxis hat es sich bewährt, Dokumente in kleinere Einheiten aufzuteilen, die jeweils ein Kapitel, Glossar, Anhang oder anderes enthalten. Die einzelnen, noch losen Einheiten, werden in einer *Verbunddatei* referenziert. Die Verbunddatei ist nur eine weitere DocBook-Datei, jedoch enthält sie zusätzliche Referenzen auf Ihre Einheiten.

Bei der Verarbeitung werden alle Dateien zusammengeführt, validiert und müssen ein gültiges DocBook-Dokument ergeben. Für die Referenzen in Ihrer Verbunddatei gibt es zwei Methoden:

- *Externe Entities* Diese Methode ist die älteste und funktioniert mit jedem Parser, der die XML-Spezifikation komplett implementiert hat.
- *XIncludes* In Ihrem Dokument werden Elemente der XInclude-Spezifikation eingefügt, die beim Verarbeiten aufgelöst werden.

11.3 Einbinden über externe Entities

Um einzelne Dateien eines DocBook-Dokument über externen Entities einzufügen, benötigen Sie die interne Teilmenge der DTD:

Beispiel 11.1. *Verbunddatei und Adressierung der Dateien*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC
    "-//OASIS//DTD DocBook XML V4.5//EN"
    "http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"
[
  <!ENTITY Info-Buch SYSTEM "bookinfo.xml"> ❶
  <!ENTITY Info-XML  SYSTEM "xmlinfo.xml">  ❶
  <!ENTITY Datenbank SYSTEM "dbinfo.xml">   ❶
]>

<book>
  <title>Das erste DocBook-Buch</title>
  &Info-Buch; ❷
  &Info-XML;  ❷
  &Datenbank; ❷
</book>
```

Kapitel 12
Anpassen von DocBook

DocBook ist für viele Anforderungen geeignet; dennoch mag es Situationen geben, in denen es nicht Ihren Vorstellungen entspricht. Dieses Kapitel zeigt Ihnen, wie Sie das DocBook-Schema (DTD oder RELAX NG) an Ihre Wünschen anpassen.

Man muß etwas Neues machen, um etwas Neues zu sehen.
—Georg Christoph Lichtenberg

12.1 Warum DocBook anpassen?

DocBook besitzt sehr allgemeingültige, teilweise umfangreiche Inhaltsmodelle. Dies wird benötigt, um verschiedene Dokumentarten zu erfassen und sprach- und landestypische Eigenschaften zu berücksichtigen.

Wenn Ihnen die Vielfalt zu umfangreich erscheint oder Sie andere Elemente und Attribute vermissen, lässt sich DocBook an Ihre Wünsche anpassen. Hierzu schreiben Sie eine *Anpassungsdatei* und fügen Ihre nachträglichen Korrekturen ein. Sie benötigen Sie, um folgende Ziele zu erreichen:

- hinzufügen neuer Elemente oder Attribute,
- löschen vorhandener Elemente oder Attribute
- erweitern der vorhandenen Werte eines Attributs,
- einschränken der vorhandenen Werte eines Attributs
- ändern des Inhaltsmodells bestehender Elemente,
- hinzufügen von anderen Schemata zu DocBook.

Alle Anpassungen werden in den nachfolgenden Abschnitten für DocBook 4 und DocBook 5 an kleinen Beispielen beschrieben.

12.2 Überlegungen vor der Anpassung

Bevor Sie DocBook an Ihre Wünsche anpassen, sollten Sie sich zuvor über einige Dinge eine Meinung bilden:

- *Welche DocBook-Version?* Eine pauschale Aussage kann nicht getroffen werden und ist vom jeweiligen Autor und dessen Vorbedingungen und Zielsetzungen abhängig. Einige Vorteile der neueren Version sind in Abschnitt 6.5, „Unterschiede zwischen DocBook 4 und 5“ (Seite 116) angegeben. Haben Sie Programme, die nur DocBook 4 unterstützen, ist es besser, bei dieser Version zu bleiben. Möchten Sie DocBook an eigene Bedürfnisse anpassen oder die Vorteile der neuen Version nutzen, ist DocBook 5 möglicherweise die bessere Wahl.
- *Aufwandsabschätzung* Die folgende Liste beschreibt den Aufwand einer jeden Anpassung. Schreiben Sie diese so, dass sie sich von anderen ebenso einfach wie DocBook anpassen lässt. Die folgende Liste ist sortiert nach Schwierigkeitsgrad, beginnend mit dem einfachsten:

  Keine Anpassung
  : Dies ist die einfachste und bequemste Art. Falls Sie dennoch „neue Elemente“ benötigen, ohne DocBook anzupassen, ist die einzige Möglichkeit, eines der allgemeinen DocBook-Attribute zu nutzen, wie `role` oder `condition`. Hierdurch erhält das Element eine zusätzliche Eigenschaft, die sich über XSLT entsprechend von gleichen Elementen ohne Attribut unterscheiden lassen.

  Entfernen von Elementen oder Attributen
  : Die „schonendste“ Anpassung, da Sie nur eine Untermenge von DocBook erstellen und somit kompatibel bleiben. Stylesheet-Änderungen sind nicht erforderlich. Der Vorteil beim Entfernen besteht darin, dass Sie bei Bedarf immer auf das Original DocBook-Schema wechseln können, falls Sie die Anpassung nicht zur Hand haben.

  Einfügen neuer Element oder Attribute
  : Neue Elemente in DocBook führen zu einer Erweiterung und sind damit inkompatibel zum Ursprungsschema. Des Weiteren führen neue Elemente zwangsläufig zu einer Stylesheet-Anpassung, egal ob sie transformiert oder unterdrückt werden.

  Einfügen eines Schema in DocBook
  : Ein anderes Schema in DocBook zu integrieren, ist prinzipiell nichts anderes als neue Elemente und Attribute einzufügen. Die Herausforderung besteht darin, die Stellen in DocBook zu finden, mit denen das neue Schema verbunden wird.

- *Weniger Elemente?* Überprüfen Sie, ob Simplified DocBook für Sie eine Alternative ist. Es enthält nur einen Bruchteil des vollständigen Schema.
- *Einschränken von Attributen* Für manche Attribute ist es praktisch, nur bestimmte Attributwerte zuzulassen. Wenn Sie beispielsweise nur für zwei Sprachen schreiben, könnten Sie nur diese erlauben.

III
DocBook-Transformationen

Kapitel 13
XPath

Informationen zu erfassen ist die eine Seite, Informationen wiederzufinden die andere. Die XML Path Language, kurz XPath, erlaubt Ihnen, in XML-Daten genau die Dinge zu finden, aus denen XML aufgebaut ist: Elemente, Attribute, Texte, Kommentare, Verarbeitungsanweisungen und XML-Namensräume. Dieses Kapitel zeigt Ihnen, wie Sie mit XPath umgehen.

Alles auf Erden lässt sich finden,
wenn man nur zu suchen sich nicht verdrießen lässt.
—Philemon

13.1 Wozu XPath?

Das W3C betrachtet XPath als eine Basis für weitere XML-Technologien. Sie verwenden somit XPath für gewöhnlich nicht isoliert, sondern in Kombinationen mit anderen Spezifikationen. Die folgende Liste gibt Ihnen einen Überblick, wo XPath überall eingesetzt wird:

- XSLT benötigt XPath, um in einem XML-Dokument genau die Informationen zu selektieren, die transformiert werden sollen.
- XPointer erweitert XPath um die Fähigkeit, Positionen und Bereiche von spezifischen Daten zu erfassen.
- Die XInclude-Spezifikation verwendet XPointer, um ausgewählte Daten aus einer XML-Datei zu extrahieren und im Ursprungsdokument einzufügen.
- XLink (eine Spezifikation zur Erstellung von Links in XML-Dateien) wiederum leiht sich Teile von XPointer; damit kann XLink Verbindungen zwischen verschiedenen Ressourcen schaffen.
- Die an C, XSLT und SQL angelehnte Abfragesprache XQuery erstellt Abfragen mit Hilfe von XPath.

XPath gibt somit Antworten auf Ihre Suchanfragen. Durch das Sammeln von Erfahrungen zu XPath können die dort erworbenen Kenntnisse für die obigen Technologien weiter genutzt werden. Dadurch müssen Sie sich nur noch mit den spezifischen Teilen befassen.

13.2 Was ist ein „XPath“?

XPath arbeitet auf der *logischen* Struktur eines XML-Dokuments und sieht ein Dokument intern als Baum von *Knoten* an. Um diese Bestandteile zu lokalisieren, müssen Sie als Anwender den „Pfad“ zu Ihrer Information erstellen, einen so genannten *XPath-Ausdruck*. Einige Beispiele von XPath-Ausdrücke sehen Sie in folgenden Zeilen:

```
/chapter
screen[3]
section[@id='intro']/title
db:chapter/db:section/db:section/db:screen
```

Was diese XPath-Ausdrücke identifizieren, wird momentan noch zurückgestellt; sie sollten nur einen Eindruck bekommen, wie sich XPath „anfühlt“. In den nachfolgenden Abschnitten werden Sie Gelegenheit haben, weitere Beispiele kennen zu lernen.

13.3 Knotenarten

XPath kennt sieben verschiedene Knotenarten:

- Wurzelknoten,
- Elementknoten,
- Attributknoten,
- Textknoten,
- Kommentarknoten,
- Namensraumknoten,
- Verarbeitungsanweisungsknoten (*Processing Instruction*).

Jeder dieser Knoten hat seine Entsprechung im XML-Dokument, nicht jedoch die XML- und DOCTYPE-Deklaration. Zur Veranschaulichung wird die folgende Datei verwendet, um die verschiedenen Knoten zu identifizieren.

```
<?xml version="1.0"?>
<?oasis-xml-catalog  catalog="file:///cat/catalog.xml"?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
    "http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
<!-- Vor dem Kapitel -->
<chapter lang="de" status="beta">
  <title>Ein Test</title>
  <para>...</para>
  <screen><![CDATA[...]]></screen>
  <xi:include
    xmlns:xi="http://www.w3.org/2001/XInclude"
```

```
    href="foo.xml"/>
  <!-- Nach xi:include -->
</chapter>
```

Eine Darstellung finden Sie in Abbildung 13.1. In den folgenden Abschnitten werden die verschiedenen Knoten im Detail erklärt.

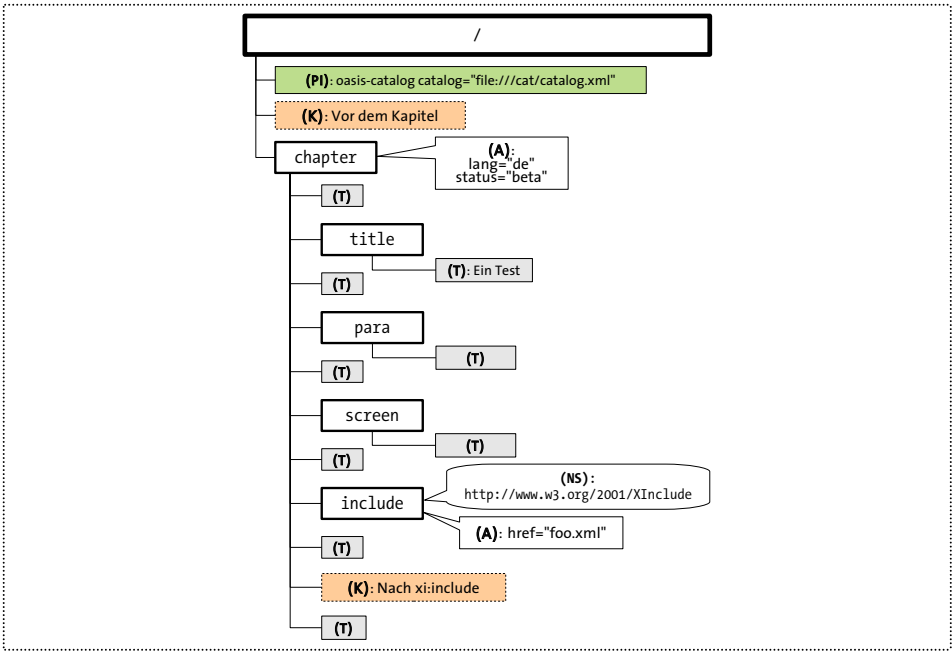

Abbildung 13.1. *Verfügbare Knoten einer Baumstruktur aus XPath-Sicht*

13.3.1 Der Wurzelknoten

Der Wurzelknoten enthält das gesamte Dokument und besitzt genau *einen* Elementknoten, der auch als *Dokument-Element* oder *Dokumentknoten* bezeichnet wird. In vorigem Fall ist das Dokument-Element `chapter`, es ist jedoch *nicht* der Wurzelknoten selbst. XPath kennzeichnet den Wurzelknoten durch einen Schrägstrich (`/`).

Innerhalb des Wurzelknotens sind außer dem Dokumentknoten noch Kommentar- oder Verarbeitungsanweisungsknoten erlaubt, jedoch keine Textknoten. Im vorigem Beispiel gehören die beiden Kommentare vor und nach dem Element `chapter` (Zeile 5 und 13) und die Verarbeitungsanweisung (Zeile 2) zum Wurzelknoten.

Zu beachten ist, dass ein Wurzelknoten keine XML- oder DOCTYPE-Deklaration enthält.

Kapitel 14
XSLT

Die erweiterbare Stylesheetsprache für Transformationen (eXtensible Stylesheet Language for Transformations) – abgekürzt XSLT – ist eine Untermenge von XSL, zusammen mit XSL-FO und XPath. XSLT ermöglicht, XML-Dokumente mit Hilfe von XPath in HTML, XML oder Text umzuwandeln. Dieses Kapitel führt Sie in die Grundlagen von XSLT ein und erläutert, wie Sie Ihre DocBook-Dokumente verarbeiten können.

> *Meine Definition einer etablierten Sprache ist, dass Computer Weekley Ausschreibungen enthält, die Leute mit drei Jahren Erfahrung suchen. XSLT hat diesen Test mühelos bestanden.*
> —Michael Kay (auf der xml-dev Mailingliste, 2 Dec 2004)

14.1 Wozu XSLT?

XSLT besitzt einige Stärken gegenüber herkömmlichen Programmiersprachen. XSLT wurde entworfen, um XML-Daten auf einfachere Art und Weise zu transformieren:

- *Plattformunabhängigkeit* Ein XSLT-Stylesheet lässt sich auf jeder Plattform ausführen, die ein XSLT-Prozessor besitzt. Dadurch kann ein Stylesheet unter Linux entwickelt werden und unter MacOS oder Windows zum Einsatz kommen (und umgekehrt).
- *Herstellerunabhängigkeit* XSLT wird vom World Wide Web Konsortium entwickelt und betreut. Es wird nicht von einem Hersteller kontrolliert. Durch die Statuten des W3Cs ist XSLT eine offene Spezifikation, bei der keine Lizenzgebühren fällig werden.
- *Toolunabhängigkeit* Wählen Sie aus einer Vielzahl aus freien und kommerziellen XSLT-Prozessoren. Manche sind sogar auf mehreren Plattformen lauffähig, sodass Sie sich nicht umgewöhnen brauchen.
- *Textbasierte Ein- und Ausgabe* XSLT ist textbasiert und kann sowohl die Eingabe (das Stylesheet) als auch die Ausgabe (Ihr Zielformat) mit jedem Editor bearbeiten. Beim Stylesheet lässt sich der Ablauf einsehen, daraus lernen und es an eigene

Wünsche anpassen. Das Zielformat kann durch weitere Filtermechanismen oder Programme weiterverarbeitet werden.

- *Erweiterungsfähigkeit* XSLT lässt sich in der Sprache Ihres XSLT-Prozessors um neue Funktionen oder Elemente erweitern. Beispielsweise können Sie eine Funktion schreiben, welche die Breite und Höhe einer Grafik ausliest, um Berechnungen im Stylesheet anzufertigen.

14.2 Übersicht über XSL, XSLT und XSL-FO

Die erweiterbare Stylesheetsprache XSL besteht aus zwei Spezifikationen:

XSL for Transformations (abgekürzt XSLT)
: XSLT ist die Transformationssprache, mit der Sie Ihre XML-Dokumente in HTML, XML oder Text umwandeln. XSLT nutzt XPath (vorgestellt in Kapitel 13 (Seite 289)), um aus Ihrem XML-Dokument die zu transformierenden Teile auszuwählen.

Formatting Objects (FO oder XSL-FO)
: XSL-FO beschreibt die Formatierungssprache. In diesem Buch wird diese Sprache in Kapitel 15, *XSL-FO* (Seite 351) behandelt. XSL-FO benötigen Sie nur, wenn Sie Ihr XML-Dokument in ein Druckformat umwandeln möchten. In diesem Fall enthält Ihr XSLT-Stylesheet zusätzlich Elemente aus der XSL-FO-Spezifikation.

Der allgemeine Aufbau eines XSLT-Stylesheets ist in Beispiel 14.1 dargestellt.

Beispiel 14.1. *Allgemeiner Aufbau eines XSLT-Stylesheets*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet❶ version="1.0"❷
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="..."/> ❸
  <xsl:template match="/"> ❹
    <!-- Inhalt --> ❺
  </xsl:template>
  <xsl:template match="chapter"> ❹
    <!-- Inhalt --> ❺
  </xsl:template>
  <!-- Weitere Templates -->
</xsl:stylesheet>
```

❶ Beginnt mit dem Wurzelelement `xsl:stylesheet` oder `xsl:transform`, die an den Namensraum `http://www.w3.org/1999/XSL/Transform` gebunden sind. Für gewöhnlich wird das Präfix xsl verwendet.

❷ Kennzeichnet die verwendete XSLT-Version (nur 1.0 oder 2.0 sinnvoll).

❸ Bestimmt das Ausgabeformat. Ein XSLT-Stylesheet kann nur HTML, XML oder Text als Zielformat ausgeben, Binärformate sind nicht möglich.

❹ Beginnt eine so genannte *Template-Regel*. Eine Template-Regel hat ein `match`-Attribut, das zu einen bestimmten Knoten Ihres XML-Dokuments passt. Dieser Knoten wird über einen XPath-Ausdruck selektiert. Im obigen Beispiel greift

das erste Template auf den Wurzelknoten zu, das zweiten Template erkennt ein `chapter`-Element.

❺ Enthält den auszugebenden Inhalt. Ein Template kann die Bearbeitung unterdrücken (dann ist der Inhalt leer), Text oder andere Elemente enthalten. XSLT-Instruktionen steuern die Ausgabe.

14.3 Der Transformationsvorgang

Folgende Schritte durchläuft ein XML-Dokument bei einer XSLT-Transformation:

1. Der XSLT-Prozessor liest das XML-Dokument ein. Es wird in eine Baumdarstellung – einen so genannten *Quellbaum* – konvertiert.
2. Der XSLT-Prozessor liest das XSLT-Stylesheet ein und erzeugt ebenfalls eine Baumstruktur, die allerdings von der ersten getrennt vorliegt.
3. Die im Stylesheet vorliegenden Template-Regeln werden für jedes Element im XML-Dokument angewendet und erzeugen einen so genannten *Ergebnisbaum*.
4. Der Ergebnisbaum wird ausgegeben.

Abbildung 14.1, „Ablauf einer XSLT-Transformation“ (Seite 315) zeigt den Vorgang grafisch.

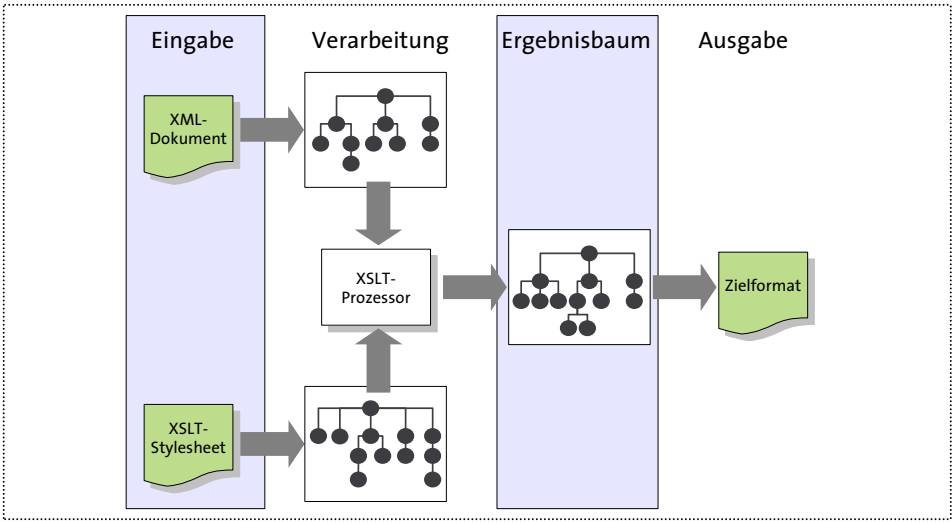

Abbildung 14.1. *Ablauf einer XSLT-Transformation*

Häufig wird das Zielformat mit anderen Programmen weiterverarbeitet: beispielsweise wird XSL-FO an einen FO-Prozessor und LaTeX-Dateien an das Programm `latex` weitergereicht. Je nach Format entsteht so eine Kaskade von Zwischenformaten. Bei (X)HTML wird das Format direkt verwendet (Browser).

Kapitel 15
XSL-FO

Die Extensible Stylesheet Language – Formatting Objects, erzeugt aus den Daten Ihres XML-Dokuments Druckerzeugnisse. Dieses Kapitel zeigt Ihnen eine Übersicht über XSL-FO, bespricht das Seitenmodell, zeigt Ihnen, wie z. B. die Schreibrichtung geändert wird, wie Umbrüche erzwungen oder verhindert werden, wie Listen und Tabellen aufgebaut sind und noch einiges mehr. Da XSL-FO sehr vielfältig ist, finden Sie weitere Informationen in den Büchern diverser Autoren, siehe [29, 27, 28, 30].

> *Mehr als das Blei in der Flinte hat das Blei im Setzkasten die Welt verändert.*
> — Georg Christoph Lichtenberg

15.1 Wozu XSL-FO?

Um XML-Dokumente in ein druckfähiges Format überzuführen, hat sich XSL-FO als eine Möglichkeit herausgebildet, mit folgenden Vorteilen:

- *Unabhängig vom Ausgabeformat* XSL-FO ist ein Zwischenformat, das eigentliche Ausgabeformat ist nur durch den FO-Formatierer begrenzt. Für gewöhnlich werden PostScript und PDF angeboten, manche Formatierer unterstützen auch SVG, RTF und noch andere Formate.
- *Stabile Spezifikation* XSL-FO ist eine ausgereifte Spezifikation. Die Ideen gehen bis in die Zeit von SGML zurück, die im ISO-Standard *Document Style Semantic and Specification Language*, kurz DSSSL, veröffentlicht wurden. XSL-FO ist das Ergebnis dieser langjährigen Forschung und Entwicklung.
- *Automatisierung* XSL-FO in Kombination mit XSLT erlaubt, das Erstellen Ihrer Druckformate zu automatisieren. Dadurch ergeben sich vielfältige Möglichkeiten, die mit einem herkömmlichen Textverarbeitungs- oder Publishingprogramm kaum möglich sind.
- *Anpassungsfähigkeit* XSL-FO und XSLT ermöglichen, dasselbe Dokument auf verschiedene Art darzustellen. Beispielsweise benötigt ein PDF für den Druck

Beschnittmarken, evtl. CMYK-Farben, während ein PDF für den Bildschirm ein anderes Format, Lesezeichen und eine größere Schrift erfordert.

- *Hohe Wiedererkennung* Viele Attributnamen sind aus CSS wiederverwendet und teilweise erweitert worden. Anwender, die bereits Erfahrung mit CSS haben, erkennen vieles wieder und können dies von CSS nach XSL-FO übernehmen.
- *Internationalisierung* XSL-FO unterstützt Unicode und verschiedene Schreibrichtungen. Mittels Unicode dürfen verschiedene Sprachen in *einer* Datei vorhanden sein. Durch unterschiedliche Schreibrichtungen sind Sie nicht auf die westliche festgelegt, sondern es lassen sich Texte setzen, die für arabische, asiatische oder hebräische Leser gedacht sind.

Trotz dieser Vorzüge gibt es einige Einschränkungen die Sie kennen sollten, wenn Sie XSL-FO einsetzen:

- XSL-FO ist deshalb komplex, weil das Setzen von Dokumenten ein komplexer Vorgang ist. Aus diesem Grund besitzt XSL-FO viele Attribute, um eben dieses zu beeinflussen. Für erste Gehversuche reichen die gängigsten aus, für höhere Ansprüche müssen Sie evtl. etwas herumprobieren.
- Für gewöhnlich wird man XSL-FO zusammen mit XSLT einsetzen. Das heisst, ein Anwender, der sich mit der Materie auseinandersetzen möchte, muss zwei verschiedene Techniken erlernen.
- Das Seitenmodell ist auf 5 Bereiche beschränkt: einen Hauptbereich mit bis zu 4 Randbereichen. Sollten Sie ein anderes Seitenmodell bevorzugen, werden Sie mit XSL-FO kein Glück haben. Für die meisten Dokumente ist diese Einschränkung allerdings nicht hinderlich.
- Die Spaltenanzahl lässt sich innerhalb einer Seite nicht nachträglich ändern. Blöcke die in solch eine Seitenvorlage fließen, können nur eine oder alle Spalten ausfüllen, nicht jedoch nur in einen Teil. Dies kann durch formatiererabhängige Erweiterungen umgangen werden, jedoch auf Kosten der Unabhängigkeit von Formatierern.
- Text kann nicht an beliebig geformten Kurven herumfließen. Als Lösung kann teilweise SVG eingesetzt werden, sofern Ihr Formatierer dieses Format unterstützt.
- Es gibt keinen allgemeinen Mechanismus für Inhalt, der seitenabhängig eingefügt oder entfernt werden kann. Beispielsweise lässt sich nicht bestimmen, ob eine Referenz auf derselben Seite ist wie eine dazugehörige Markierung (um ein *auf dieser Seite* oder ähnliche Texte einzufügen).
- Für registergenaue Anordnung von Text ist XSL-FO (noch?) nicht geeignet. Wenn Text „auf Register gesetzt“ wird, bedeutet dies, dass Zeilen auf der Vorderseite und Zeilen auf der Rückseite an derselben Position stehen. Je nach Papierdicke scheinen die Zeilen der Rückseite mehr oder weniger durch und stören den Grauwert der Seite.

Kapitel 16
Die DocBook-XSL-Stylesheets

Um DocBook in andere Formate umzuwandeln, sind die Stylesheets von Norman Walsh und anderen das Mittel der Wahl. In diesem Kapitel werden die formatunabhängigen Bestandteile beschrieben. Sie erfahren, welche Stylesheets es gibt, wie Sie Parameter verändern, eine Anpassungsdatei schreiben, wie Sie Elemente filtern, Erweiterungen nutzen und noch vieles mehr. Da die DocBook-Stylesheets sehr vielseitig sind, kann nicht auf jedes Detail eingegangen werden. Weitere Informationen finden Sie im Buch [14].

> *Keine Leidenschaft der Welt ist größer als die Leidenschaft,*
> *jemand anderes Vorgabe ändern zu wollen.*
> —Orson Welles

16.1 Überblick über die Stylesheets

Ein DocBook-Dokument wird für gewöhnlich nicht „roh“ konsumiert, sondern muss in ein Zielformat wie (X)HTML, XSL-FO usw. transformiert werden. Diese Transformation erfüllen die DocBook-Stylesheets. Mittlerweile gibt es eine Reihe von Zielformaten, welche durch die Stylesheets erzeugt werden können.

Durch DocBook 5 haben sich Varianten herausgebildet, die im Großen und Ganzen auf derselben Codebasis aufbauen, jedoch sich in Details unterscheiden. Zum Zeitpunkt der Bucherstellung gibt es folgende DocBook-Stylesheets:

DocBook XSL (stabil)
: Dies ist die Originalversion, implementiert in XSLT 1.0 und für DocBook 4 (bevorzugt) und 5 geeignet.

DocBook XSL Namensraum (NS) relevant (stabil)
: Diese Variante basiert auf den ersten Stylesheets, ebenfalls in XSLT 1.0 implementiert und für DocBook 4 und 5 (bevorzugt) geeignet. Weitere Informationen finden Sie in Abschnitt 16.1.1, „Welche Stylesheet-Variante nehmen?“ (Seite 374).

DocBook XSL2 (in Entwicklung)[1]
: Diese Stylesheets sind eine Neuimplementierung der Originalversion in XSLT 2. Sie sind zum Zeitpunkt der Bucherstellung noch experimentell und werden in diesem Buch daher nicht weiter beschrieben. Interessierte Leser benötigen einen XSLT 2-Prozessor. Gegenwärtig ist dies nur mit Saxon 8 und höher möglich.

16.1.1 Welche Stylesheet-Variante nehmen?

Die kurze Antwort auf die Frage im Titel lautet: Verwenden Sie die Original-Stylesheets für DocBook 4 und die DocBook XSL NS Stylesheets für DocBook 5. Die etwas längere, ausführlichere Antwort lautet, dass im Prinzip beide Varianten auf beide DocBook-Versionen anwendbar sind, wenn auch mit kleinen Einschränkungen. Dieser Abschnitt soll die Hintergründe kurz erklären.

Der Hauptunterschied zwischen DocBook 5 und früheren Versionen besteht im Namensraum `http://docbook.org/ns/docbook`. Die Originalstylesheets erkennen jedoch nur solche DocBook-Elemente, die an keinen Namensraum gebunden sind. Folglich erkennen Sie keine DocBook 5-Elemente.

Damit DocBook 5-Elemente dennoch erkannt werden, um sie in Ihr Zielformat zu transformieren, bedienen sich die Originalstylesheets eines Tricks: Entdecken sie im Wurzelelement den DocBook 5-Namensraum, wird das Dokument im Modus `stripNS` verarbeitet. Wie der Name suggeriert, entfernt dieser Modus nur den DocBook-Namensraum eines jeden DocBook 5-Elements, lässt die Struktur des Dokuments jedoch unangetastet. Das Ergebnis wird in einer Variablen zwischengespeichert und diese in eine Knotenmenge konvertiert. Dadurch wird den Originalstylesheets vorgegaukelt, sie hätten DocBook 4-Elemente, wodurch sich die vorgegebenen Template-Regeln anwenden lassen.

Die andere Alternative für DocBook 5 wäre, Stylesheets zu schreiben, die den DocBook-Namensraum erkennen. Technisch völlig unproblematisch, jedoch müssten dann zwei Stylesheet-Varianten gewartet werden, die sich nur in kleinen Ergänzungen unterscheiden. Dies hätte einen erhöhten Wartungsaufwand zur Folge. Um dies zu vermeiden, hat Bob Stayton ein Perl-Skript geschrieben, das die vorhandenen Original-Stylesheets als Grundlage verwendet. Das Perl-Skript untersucht die XSLT-Attribute `count`, `from`, `match`, `use`, `select` und `test` und fügt an jedes DocBook-Element ein Präfix an. Zusätzlich wird der DocBook-Namensraum im Wurzelelement eingefügt. Das heißt, aus `<xsl:template match="chapter">` wird `<xsl:template match="d:chapter">`.

Wenn Sie die DocBook XSL NS Stylesheets für Ihre DocBook 5-Dokumente verwenden, haben Sie folgende Vorteile:

1. Die Stylesheets sind speziell für DocBook 5 ausgelegt, sie benötigen keine „Hacks“. Dies führt zu einem geringeren Speicherverbrauch, da das komplette Dokument nicht mehr in einer Variablen zwischengespeichert werden muss.

1 Siehe `https://docbook.svn.sourceforge.net/svnroot/docbook/trunk/xsl2`, im DocBook-Subversion Repository.

Kapitel 17
Konditionale Elemente (Profiling)

Profiling beschreibt eine Methode, um Elemente mit Bedingungen zu versehen. Für gewöhnlich entfernen Sie durch Profiling Elemente, um verschiedene Varianten des Originaldokuments zu erzeugen. Dieses Kapitel zeigt Ihnen, wie Sie Ihre Dokumente für Profiling zugänglich machen und wie Sie verschiedene Varianten aus Ihrem Dokument gewinnen.

> *Das Feinste fällt durchs Sieb.*
> —Wilhelm Busch

17.1 Wozu Profiling?

Profiling ist ein zusätzlicher Mechanismus für Ihre Verarbeitungskette und ist nur sinnvoll, wenn folgende Bedingungen erfüllt sind:

- Sie müssen mehr als eine *Variante* eines Dokuments erstellen. Eine Variante ist eine Untermenge Ihres Originaldokuments. Beispiele folgen sogleich.
- Es gibt *wenig Unterschiede* zwischen den Varianten. *Wenig* bedeutet, es gibt einen Kern Ihres Dokuments der in allen Varianten nahezu identisch ist.

Profiling sollte bereits beim Schreiben berücksichtigt werden. Nachträgliches Einfügen in fertigen Dokumenten ist zwar möglich, jedoch mit etwas Aufwand verbunden. Folgende Beispiele zeigen, wo es sinnvoll sein kann, Profiling anzuwenden:

- Sie dokumentieren eine Software und die Installation unter verschiedenen Betriebssystemen (dieses Beispiel werden Sie in den folgenden Abschnitten kennen lernen). Der Kern Ihrer Anleitung ist bei allen gleich, jedoch gibt es betriebssystemspezifische Abschnitte.

- Sie verfassen eine Anleitung, die an verschiedene Hersteller gerichtet ist. Der Kern Ihrer Anleitung ist bei allen identisch, jedoch gibt es Abschnitte, die nur die jeweiligen Hersteller lesen dürfen.
- Sie schreiben Texte für verschiedene Zielgruppen wie Prüfling/Prüfer, Anfänger/Experte usw.
- Ein Dokument soll verschiedene Sicherheitshinweise enthalten, beispielsweise für Administratoren oder normale Anwender.
- Sie arbeiten an einem Trainingshandbuch und haben Abschnitte mit verschiedenen Schwierigkeitsgraden.

17.2 Verarbeitungsschritte

Jiří Kosek hat für das Profilieren eines Dokuments eine Lösung in Form von DocBook-Stylesheets ausgearbeitet. Anhand folgender Abbildung ist dargestellt, wie Profiling funktioniert:

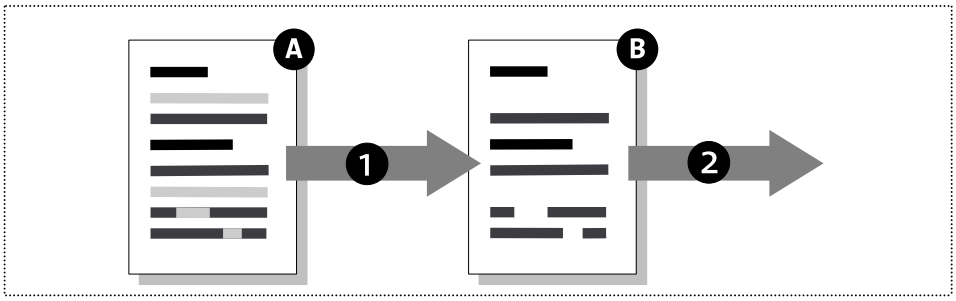

Abbildung 17.1. *Verarbeiten eines DocBook-Dokuments mit Profiling*

Der Profiling-Mechanismus in Abbildung 17.1 enthält zwei Schritte:

1. *Profilieren* Sie wandeln Ihr Dokument mit einem Profiling-Stylesheet um. Hierbei werden Parameter an das Stylesheet weitergereicht, welche die Bedingungen für die einzelnen Varianten festlegen.
2. *Transformieren* Das Ergebnis der Profilierung wird mit einem Transformationsschritt in das Zielformat umgewandelt.

Hierbei sind zwei Dokumente an diesem Prozess beteiligt:

A. *Originaldokument* Sie schreiben Ihren Text in DocBook und verwenden an der geeigneten Stelle Profiling-Attribute. Hierbei setzen Sie ein Kriterium für ein Kapitel, einen Absatz oder gar einen Befehl. Solche Elemente werden nur übernommen, wenn das Kriterium erfüllt ist.
B. *Ergebnis des Profiling* Nach dem Profilingschritt entsteht ein Dokument, das nur die Stellen enthält, für die kein Kriterium angegeben wurde, oder bei denen das

Kapitel 18
(X)HTML erzeugen

Dieses Kapitel zeigt Ihnen, wie Sie die DocBook-Stylesheets nutzen, um eine oder mehrere (X)HTML-Dateien zu erzeugen. Es erklärt Ihnen, wie Sie die Stylesheets nach Ihren Wünschen anpassen, um den Aufteilungsvorgang individuell zu steuern, Inhaltsverzeichnisse zu beeinflussen oder Kopf- und Fußzeilen zu ändern. Durch CSS lassen sich die erzeugten (X)HTML-Dateien individuell formatieren.

> *Mich hat es überrascht, dass die Leute klaglos bereit waren, umständlich HTML einzugeben.*
> —Tim Berners-Lee

18.1 Verfügbare Stylesheets

Für die Transformation von DocBook nach (X)HTML gibt es die folgenden Stylesheets:

Tabelle 18.1. *DocBook-Stylesheets für (X)HTML*

Dateiname	*Beschreibung*
`docbook.xsl`	erzeugt eine einzige (X)HTML-Datei, die alles enthält
`chunk.xsl`	erzeugt Teildateien in XHTML
`profile-docbook.xsl`	erzeugt (X)HTML wie `docbook.xsl`, jedoch mit Profiling (siehe Kapitel 17, *Konditionale Elemente (Profiling)* (Seite 419))
`profile-chunk.xsl`	erzeugt (X)HTML wie `chunk.xsl`, jedoch mit Profiling
`onechunk.xsl`	erzeugt wie `docbook.xsl` eine Datei, die jedoch nach dem `id`-Attribut des Wurzelelements benannt ist
`chunktoc.xsl`	erlaubt die manuelle Kontrolle über den Aufteilungsvorgang (siehe Abschnitt 18.5.6, „Manuelle Kontrolle über den Aufteilungsvorgang“ (Seite 444))

Die Dateinamen in Tabelle 18.1, „DocBook-Stylesheets für (X)HTML“ (Seite 435) sind sowohl für HTML als auch für XHTML verfügbar. Wenn also von HTML gesprochen wird, ist das Gesagte analog auch auf XHTML anwendbar.

18.2 Eine Beispieltransformation

Dieser Abschnitt zeigt Ihnen, was Sie für eine Umwandlung eines DocBook 4-Dokuments in XHTML benötigen. Die Umwandlung von DocBook 5 nach XHTML ist bis auf Kleinigkeiten (Namensräume, Pfade, usw.) identisch.

18.2.1 Das Dokument

Es wird von folgenden kurzen Dokument ausgegangen:

Beispiel 18.1. *pinguin.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC
  "-//OASIS//DTD DocBook XML V4.5//EN"
  "http://www.docbook.org/xml/4.5/docbookx.dtd">
<book lang="de">
  <title>Über Pinguine</title>
  <chapter>
    <title>Anatomie und Aussehen</title>
    <para>Der Größen- und Gewichtsunterschied der verschiedenen
      Pinguinarten ist beträchtlich, Körperbau und Gefieder sind
      in der Familie dagegen sehr homogen. [...]</para>
  </chapter>
  <chapter>
    <title>Verbreitung</title>
    <para>Pinguine leben in den offenen Meeren der südlichen
    Hemisphäre. [...]</para>
  </chapter>
  <!-- ... -->
</book>
```

Umfangreiche Dokumente wie Bücher werden Sie aus Komfortgründen wahrscheinlich in einzelne Kapitel aufteilen. Hinweise hierzu finden Sie in Kapitel 11, *Modulare Dokumente* (Seite 221).

18.2.2 Die Transformation

Um Ihr DocBook-Dokument nach XHTML umzuwandeln, benötigen Sie einen XSLT-Prozessor (`xsltproc`, Saxon oder Xalan) und die DocBook-Stylesheets. Hierzu haben Sie folgende Möglichkeiten (vgl. Abschnitt 16.2, „DocBook XSL Stylesheets anpassen“ (Seite 376)):

1. Sie verwenden die Originalstylesheets. Ersetzen Sie den Platzhalter *DB* durch Ihr Installationsverzeichnis der DocBook-Stylesheets und rufen Sie auf:

Kapitel 19
FO (PDF) erzeugen

Die DocBook-Stylesheets erzeugen XSL-FO, ein Zwischenformat um üblicherweise PDF zu erzeugen. Dieses Kapitel zeigt Ihnen, wie Sie eine Anpassungsdatei schreiben, Seitenformate für verschiedene Dokumenttypen erstellen, Titelseiten anpassen, mit Kopf- und Fußzeilen arbeiten, Inhaltsverzeichnisse formatieren und noch vieles mehr. Da die Fülle des Materials sehr umfangreich ist, musste Vieles weggelassen werden. Weitere Informationen finden Sie in den Büchern von Dave Pawson zu XSL-FO und von Bob Stayton zu den DocBook-Stylesheets.

Körper und Stimme leiht die Schrift dem stummen Gedanken
Durch der Jahrhunderte Strom trägt ihn das redende Blatt.
—Friedrich Schiller

19.1 Verfügbare Stylesheets

Anders als bei der (X)HTML-Ausgabe gibt es für XSL-FO nur zwei Stylesheets, die direkt aufgerufen werden können (siehe Tabelle 19.1, „DocBook-Stylesheets für XSL-FO“ (Seite 471)).

Tabelle 19.1. *DocBook-Stylesheets für XSL-FO*

Dateiname	*Beschreibung*
`fo/docbook.xsl`	erzeugt XSL-FO; das heißt, das gesamte Dokument befindet sich nach der Transformation in einer Datei.
`fo/profile-docbook.xsl`	erzeugt XSL-FO wie `docbook.xsl`, jedoch mit Profiling (siehe Kapitel 17, *Konditionale Elemente (Profiling)* (Seite 419))

Teildateien wie für (X)HTML sind für XSL-FO sinnlos, da die Spezifikation dieses nicht vorsieht. Wenn Sie nur ein Kapitel oder einen Abschnitt verarbeiten möchten, verwen-

den Sie den Parameter *rootid* (siehe Abschnitt 16.3, „Teilweises Transformieren“ (Seite 380)).

19.2 Anpassungsdatei für XSL-FO anlegen

Ebenso wie in (X)HTML werden Parameter entweder über den XSLT-Prozessor verändert oder Sie schreiben Sie in eine Anpassungsdatei. Eine Auflistung aller Parameter für XSL-FO finden Sie unter `http://docbook.sourceforge.net/release/xsl/current/doc/fo/`. Komplexere Änderungen brauchen stets eine Anpassungsdatei. Ein minimaler Aufbau hat folgende Struktur:

Beispiel 19.1. *Eine Anpassungsdatei für XSL-FO*

```
<xsl:stylesheet version="1.0"
   xmlns:fo="http://www.w3.org/1999/XSL/Format"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:import href="DBBASISURI/fo/docbook.xsl"/>
 <xsl:param name="paper.type" select="'A4'"/>
 <!-- Fügen Sie hier weitere Optionen ein -->
</xsl:stylesheet>
```

Ersetzen Sie den Platzhalter *DBBASISURI* durch den entsprechenden offiziellen URI, vgl Tabelle 16.1 (Seite 378).

19.3 Seitenformate ändern

Die DocBook-XSL-FO-Stylesheets erlauben das Seitenformat an Ihre Wünsche anzupassen. Sowohl die bekannten DIN-Formate als auch selbstdefinierte sind unterstützt.

19.3.1 Verwenden vordefinierter Seitenformate

Standardmäßig ist bei den DocBook-XSL-FO-Stylesheets das amerikanische Seitenformat USletter eingestellt. Für europäische bzw. deutsche Drucksachen ist dieses Format unüblich. Daher erlauben die DocBook-Stylesheets ebenso die DIN-Formate von A0 bis A10, B0 bis B10 und C0 bis C10. Um das DIN A4-Format auszuwählen, geben Sie ein:

```
xsltproc --stringparam paper.type 'A4' ...
```

Entsprechend geben Sie in Ihrer Anpassungsdatei ein:

```
<xsl:param name="paper.type">A4</xsl:param>
```

19.3.2 Parameter für das Seitenlayout

Standardmäßig werden Ränder auf bestimmte Vorgaben gesetzt. Mit den folgenden Parametern in Tabelle 19.2 lässt sich das Seitenlayout einer Seite beeinflussen.

Kapitel 20
E-Book-Format EPUB erzeugen

E-Books sind modern: Erlauben Sie durch ein Lesegerät schnell und bequem Ihre Lieblingsbücher ohne viel Aufwand mitzunehmen. Dieses Kapitel zeigt Ihnen, wie Sie eine Anpassungsdatei für EPUB erzeugen, einem Format für E-Books. Des Weiteren erfahren Sie, welche Schritte notwendig sind um eine EPUB-Datei zu erstellen und wie Sie das Cover an eigene Wünsche anpassen.

Verlage sind Geburtskliniken des Geistes.
—John Irving

Wie machen wir's, daß alles frisch und neu
und mit Bedeutung auch gefällig sei?
—Johann Wolfgang von Goethe (Faust I)

20.1 Was sind E-Books und EPUB?

E-Books sind „digitale Bücher" die Sie mit einem entsprechenden Gerät lesen können. Der Vorteil von E-Books ist, dass auf solch einem Gerät viele Bücher ihren Platz finden und Sie somit immer Ihre Lieblingswerke zur Hand haben.

Aus historischen Gründen gibt es eine verwirrende Formate-Vielfalt.[1] Mittlerweile unterstützen die Geräte weitere Formate, die nicht direkt als „E-Book" entwickelt wurden, jetzt aber unter diesem Begriff eingeordnet werden. Um diese Vielfalt besser zu verstehen, werden E-Books in diesem Buch anhand ihrer primären Eigenschaft eingeordnet, der Zeilenanpassung:

„Statische Zeilenanpassung"
: E-Books in diesem Format haben ein „Papierformat", das heisst, eine feste Höhe und Breite (üblicherweise DIN A4). Wenn Sie die Darstellung vergrößern, ragen

1 Einen Vergleich zwischen verschiedenen E-Book-Formaten finden Sie unter der URL `http://en.wikipedia.org/wiki/Comparison_of_e-book_formats`.

Zeilen über den sichtbaren Ausschnitt Ihres Bildschirms hinaus. Dadurch erschwert sich das Lesen in diesen Formaten, da Sie Ihren sichtbaren Ausschnitt horizontal verschieben müssen und es mühsam wird, den Anfang einer Zeile wiederzufinden.

Die gängigsten „statischen" Formate sind PDF, RTF, Text, Doc und andere.

„Dynamische Zeilenanpassung"
E-Books in diesem Format passen sich der Bildschirmgröße Ihres Lesegerätes an. Vergrößern Sie die Darstellung, wird die Zeile automatisch umgebrochen. Ein horizontales Verschieben Ihres Ausschnittes entfällt, außer das Format enthält übergroße Grafiken. Dadurch ergibt sich eine viel natürlichere Leseart (von oben nach unten) als dies bei statischen Formaten der Fall ist.

Gängige „dynamische" Formate sind (X)HTML, EPUB, Plucker, MobiPocket und andere.

Aus den vielfältigen E-Book-Formaten scheint sich mittlerweile eine gemeinsame Spezifikation herauszubilden, das sogenannte *EPUB-Format* (Stand Februar 2009). Dieses Format wird vom International Digital Publishing Forum (IDPF) entwickelt und ist aufgeteilt in drei Spezifikationen:

- *OEBPS Container Format (OCF) [42]* OCF fasst alle Dateien als ZIP-Archiv zusammen. Es beschreibt, wie die Struktur des ZIP-Archivs aussieht und welche Metainformationen darin zu finden sind. Optional enthält es Informationen zu digitalen Signaturen, Verschlüsselung oder digitaler Rechteverwaltung (DRM).
- *Open Packaging Format (OPF)[43]* OPF beschreibt und referenziert alle Bestandteile einer Publikation als XML-Datei wie Markupdateien, Bilder und Navigationsstrukturen. Des Weiteren bestimmt es die Lesereihenfolge Ihres Inhalts und definiert ein Inhaltsverzeichnis.
- *Open Publication Structure (OPS) [44]* OPS beschreibt den Inhalt der Publikation auf Basis von XHTML 1.1.

Alle drei Spezifikationen sind frei verfügbar und basieren auf offenen Spezifikationen oder Standards. Seit 2008 gibt es Stylesheets von Keith Fahlgren, um DocBook in EPUB umzuwandeln.

Der Vorteil von EPUB gegenüber anderen E-Book-Formaten sind die folgenden Punkte:

- CSS wird vollständig unterstützt, wodurch Ihr Inhalt wie eine HTML-Seite gestaltet werden kann.
- Schriftarten dürfen in das EPUB-Format eingebettet werden. Dadurch ergibt sich eine breite typografische Bandbreite.
- Durch so genannte *erweitere Navigationsstellen* (*navigation center extended*) ist es möglich, die Hauptstruktur des Dokuments einzublenden ohne dass das komplette Dokument gelesen werden muss. Weitere Verzeichnisse wie Abbildungen, Tabellen, Fußnoten und andere kann der Leser über eine separate Liste anzeigen – sofern die Datei dies enthält.

- Ein Rückfallsystem, falls ein Medientyp auf einem Lesegerät nicht angezeigt werden kann. Wenn beispielsweise die EPUB-Datei eine SVG-Grafik enthält, die vom Lesegerät nicht angezeigt werden kann, wird stattdessen versucht, ein alternatives Grafikformat auszuwählen. Dies ist nur möglich, falls die EPUB-Datei die entsprechende Auswahl an Medientypen enthält.

20.2 Anpassungsdatei für EPUB anlegen

Eine Anpassungsdatei für EPUB ähnelt einer für (X)HTML bzw. XSL-FO. Da die Stylesheets für EPUB auf denen für XHTML 1.1 basieren, sind die selben Parameter sowohl für EPUB als auch für (X)HTML möglich. Eine Auflistung finden Sie unter `http://docbook.sourceforge.net/release/xsl/current/doc/html/`. Eine minimale Struktur sieht wie folgt aus:

Beispiel 20.1. *Anpassungsdatei für EPUB*

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="DBBASISURI/epub/docbook.xsl"/>
  <!-- Fügen Sie hier Ihre Parameter ein: -->
</xsl:stylesheet>
```

Ersetzen Sie den Platzhalter *DBBASISURI* durch den entsprechenden offiziellen URI, vgl Tabelle 16.1 (Seite 378).

20.3 EPUB als E-Book erzeugen

Seit Ende 2008 enthalten die DocBook-Stylesheets EPUB als experimentelles Ausgabeformat. Obwohl sie bereits erstaunlich gute Ergebnisse erzielen, sind die Stylesheets für EPUB dennoch als „vorläufig“ bezeichnet (Stand März 2009).

Um aus Ihrem DocBook-Dokument ein E-Book zu erzeugen, sind zwei Verfahren möglich: Sie verwenden ein mitgeliefertes Ruby-Skript oder Sie erzeugen das EPUB auf manuellem Weg. Versuchen Sie zuerst die erste, da einfachere Methode.

20.3.1 EPUB mit Ruby-Skript erzeugen

Die DocBook-Stylesheets enthalten ein Ruby-Skript, das Ihr Dokument in XHTML 1.1 umwandelt, Grafiken, CSS und Schriftarten sucht und das Ganze als eine ZIP-Datei zusammenpackt.

Korrekte Katalogeinstellungen notwendig
Die Hinweise in diesem Abschnitt benötigen eine XML-Umgebung mit XML-Katalogunterstützung, das heisst, der XML-Katalog muss die richtigen Einträge für die DocBook-DTDs und Stylesheets enthalten. Informieren Sie sich in Kapitel 5 (Seite 99).

IV
DocBook vertiefen

Kapitel 21
Tipps und Tricks für DocBook

Wie DocBook-Code geschrieben wird, war bereits Thema früherer Kapitel. Dieses Kapitel vergleicht die DocBook-Schemata, gibt Hilfestellungen bei Auszeichnungsproblemen, lüftet einige Geheimnisse, gibt Tipps und Tricks und zeigt, wie Sie Ihre Dokumente konsistent halten.

> *Ratschläge sind eine gefährliche Gabe, selbst von den Weisen an die Weisen.*
> *Und alle Wege mögen in die Irre führen.*
> —J. R. R. Tolkin (Der Herr der Ringe, Band 1)

21.1 Welches DocBook-Schema?

Wie in Abschnitt 6.3, „Schemadschungel“ (Seite 112) bereits erläutert, gibt es DocBook in vielerlei Kombinationen: zum einen die im Laufe der Zeit verschiedenen Versionen, zum anderen die verschiedenen Schemasprachen. Wie behält man den Überblick? Vor allem: Welche Version und welches Schema sollte man wann verwenden?

Eine allgemeingültige Antwort auf diese Fragen kann es nicht geben und muss von jedem Anwender individuell entschieden werden. Die folgende Zusammenstellung hilft Ihnen, eine Entscheidung zu treffen:

DocBook 4
: Die offizielle Schemasprache von DocBook 4 ist die DTD. Für interessierte Anwender sind inoffizielle Schemata für RELAX NG und W3C-Schema erhältlich. Für gewöhnlich wird in der 4er Serie nur die DTD eingesetzt. Achten Sie darauf, möglichst die letzte Version (gegenwärtig 4.5) zu verwenden. Frühere Versionen sollten Sie nur einsetzen, wenn Sie ältere Dokumente besitzen und diese mit anderen Personen austauschen.

DocBook 5
: Die offizielle Schemasprache von DocBook 5 ist RELAX NG. Andere Schemata sind aus Kompatibilitätsgründen für ältere Programme vorhanden. Neuere Dokumente

sollten DocBook 5 verwenden, um die gesamte Palette an Vorteilen auszunutzen. Des Weiteren passieren zukünftige Entwicklungen nur in DocBook 5 während die 4er Serie eingefroren wird und nur noch Fehlerkorrekturen zugelassen werden.

Eine detailliertere Aufstellung zeigt die folgende Tabelle:

Tabelle 21.1. *Vergleich zwischen DocBook 4 und DocBook 5*

Thema	*4.5*	*5.0*
Offizielles Schema	DTD	RELAX NG
Elementanzahl	406	385
Datentypen	nein	ja
Enthält Schematron	nein	ja
Enthält XInclude-Elemente im Schema	nein	nein[a]
Leichte Anpassung	nein	ja
Anmerkungen	nein	ja
Unterstützt Namensräume	nein	ja
Universelle Verlinkung durch XLinks	nein	ja
Zukünftige Entwicklung	nein	ja
Wurzelelement auswählen	nein	ja
Vordefinierte Entities	ja	nein[b]
Entwicklung eingefroren	ja[c]	nein
Vereinheitlichte Metainformationen (`info` vs. `appendixinfo`, `chapterinfo`, …)	nein	ja

a XInclude-Element sind in den Dateien `docbookxi.rnc` bzw. `docbookxi.rng` vorhanden

b Lässt sich einbauen.

c Nur noch Fehlerkorrekturen

21.2 Was und wieviel auszeichnen?

Beim Arbeiten mit DocBook werden Sie feststellen, dass Sie sich entscheiden müssen, welches Element Sie verwenden. Soll das Wort „URL“ mit `acronym` ausgezeichnet werden? Ist es sinnvoll, alle Abkürzungen im Dokument in `abbrev` einzuschließen? Muss ein Name mit `personname` umschlossen werden?

Für diese Fragen gibt es kein richtig oder falsch, sondern die Antwort hängt von einigen Faktoren ab (diese werden gleich gezeigt). Grundsätzlich müssen Sie *nicht* das komplette Arsenal an Elementen von DocBook nutzen. Sie können Dokumente

Kapitel 22
Migrieren nach DocBook

Sie möchten DocBook und dessen Vorteile nutzen, jedoch liegen Ihre Dokumente in einem anderen Format vor. Da es eine Vielzahl an Formaten gibt, ist dieses Kapitel als Ideengeber gedacht, um die Migration nach DocBook zu vereinfachen.

Es fürchtet jemand die Umwandlung?
Was kann denn ohne Umwandlung geschehen.
—Mark Aurel, römischer Kaiser

22.1 Von DocBook 4 nach DocBook 5

DocBook 5 bietet einige Vorteile gegenüber der 4er Serie: leichtere Anpassung, verbesserte Verlinkung mittels XLink, genauere Validierung über RELAX NG und einige andere (siehe Abschnitt 6.5, „Unterschiede zwischen DocBook 4 und 5“ (Seite 116) und Abschnitt 21.1, „Welches DocBook-Schema?“ (Seite 517)). Falls Sie einige dieser Vorteile benötigen, Sie jedoch Altdokumente besitzen, lässt sich die Migration sehr einfach über eine XSLT-Transformation erledigen. Gehen Sie wie folgt vor:

1. Validieren Sie Ihr DocBook 4.x-Dokument und korrigieren Sie eventuelle Fehler. Die nachfolgende Transformation erwartet ein valides Dokument.
2. Laden Sie das Stylesheet für die Transformation von Version 4 nach Version 5 herunter. Die neueste Version erhalten Sie von der Adresse `https://docbook.svn.sourceforge.net/svnroot/docbook/trunk/docbook/relaxng/tools/db4-upgrade.xsl`.
3. Falls Sie Entities in Ihrem Dokument haben, die Sie erhalten möchten, gehen Sie wie folgt vor, ansonsten fahren Sie fort mit Schritt 4:
   a. Öffnen Sie Ihr Dokument mit einem Text-Editor Ihrer Wahl, verwenden Sie allerdings keinen XML-Editor.
   b. Schneiden Sie die DOCTYPE-Deklaration aus und sichern Sie sie. Sie wird später wieder benötigt.

   c. Suchen Sie alle selbstdefinierten Entities und ersetzen Sie diese durch eine Zeichenkette, die nicht in Ihrem Dokument vorhanden ist. Der *Transition Guide* [8] empfiehlt, ein Entity `&Product;` durch `[[[Product]]]` zu ersetzen. Sie können beliebige Zeichen verwenden, jedoch müssen sie eindeutig sein.
   d. Wiederholen Sie den letzten Schritt für alle Entities, die Sie erhalten möchten.
   e. Speichern Sie das veränderte Dokument.

4. Transformieren Sie Ihr Dokument mit Hilfe des Stylesheets `db4-upgrade.xsl`:

   ```
   xsltproc --output doc5.xml db5-upgrade.xsl doc4.xml
   ```

5. Öffnen Sie das neue Dokument und fügen Sie die DOCTYPE-Deklaration von Schritt 3.b nach der XML-Deklaration ein.
6. Entfernen Sie alle externen Bezeichner wie PUBLIC oder SYSTEM. Abhängig vom Wurzelelement bleibt folgendes Gerüst übrig:

   ```
   <!DOCTYPE chapter [
     <!-- Der Inhalt -->
   ]>
   ```

7. Falls Sie Ihre selbstdefinierten Entities im Schritt 3 geschützt haben, müssen Sie diese wieder zurückwandeln, das heißt aus `[[[Product]]]` muss wieder `&Product;` werden.
8. Validieren Sie das Ergebnis aus dem letzten Schritt mit dem RELAX NG-Schema von DocBook 5:

   ```
   jing -c docbook.rnc doc5.xml
   ```

   Da das Stylesheet möglicherweise noch nicht alle Fälle abdeckt, sollten Sie mögliche Fehler den Entwicklern mitteilen. Damit ermöglichen Sie, dass die Umwandlung in Zukunft besser funktioniert. Melden Sie Fehler auf der DocBook-Seite unter SourceForge.net (siehe `http://sourceforge.net/projects/docbook/`). Sie müssen sich zuvor anmelden, bevor Sie einen Fehlerbericht abschicken.

22.2 Migrieren nach DocBook

Sie sind von den Vorteilen von DocBook überzeugt und möchten es für sich oder Ihr Team nutzen. Es stellt sich noch eine Frage: Wie bekommen Sie Ihre vorhandenen Dokumentationen in das DocBook-Format? Um spätere Enttäuschungen gleich vorzubeugen: Je nach Ausgangsformat wird die Lösung mehr oder weniger überzeugen.

Je nach Format wird die Lösung immer individuell sein. Bei einigen etablierten Formaten haben sich Methoden herausgebildet, die erfolgsversprechend sind. Dennoch ist es sehr wahrscheinlich, dass Sie das Ergebnis manuell nacharbeiten müssen. Ein Trost an dieser Stelle: Normalerweise müssen Sie diese Arbeit nur einmal erledigen.

Das Konvertieren nach DocBook bereitet aus den folgenden Gründen Probleme:

1. *Das Format ist binär* Viele Programme schreiben Ihr Dateiformat in einem Binärformat. Das ist per se noch kein Problem. Jedoch wird der Aufbau von den

Kapitel 23
DocBook — Ein Ausblick

Das vorliegende Kapitel stellt einige Entwicklungen im XML- und DocBook-Umfeld vor, die für manche interessant sein mögen. Vieles davon ist noch im Fluß, daher ist dieses Kapitel eher für Entwickler interessant.

> *Wenn der Mensch nicht über das nachdenkt,*
> *was in ferner Zukunft liegt,*
> *wird er das schon in naher Zukunft bereuen.*
> —Konfuzius

23.1 Entwicklung von DocBook 5

Die letzte Version der DocBook-DTD war zum Zeitpunkt der Bucherstellung Version 4.5. Außer Fehlerkorrekturen ist die Entwicklung der 4er Serie eingefroren, neuere Elemente und Konzepte werden nur in DocBook 5 und höher Einzug halten.

Wenn Sie an der Entwicklung von DocBook interessiert sind, können Sie die *Request for Enhancement* (RFE) für die Version 5 und höher unter der URL `http://sourceforge.net/tracker/?group_id=21935&atid=384107` einzusehen.

23.2 Einflüsse von anderen Dokumentenformaten

DocBook war und ist nie isoliert gewesen. Durch seinen modularen Aufbau kann es mit anderen Schemata kombiniert werden. Die folgenden Abschnitte stellen andere Dokumentformat vor und wie diese in Beziehung zu DocBook stehen.

23.2.1 DITA und DocBook

DocBook hat knapp 400 Elemente[1]. Falls Sie für Ihre Dokumentation ein bestimmtes Element suchen, ist es sehr wahrscheinlich, dass DocBook es bereits enthält.

1 Für die ganz genauen: DocBook 4.5 hat 406 und DocBook 5 hat 382 Elemente.

Anhang B
XML-Umgebung einrichten

Dieser Anhang zeigt Ihnen, wie Sie eine einfache, kommandozeilenbasierte Umgebung für Linux und Windows einrichten, um die verschiedenen Schritte des Verarbeitungsprozesses von DocBook-Dokumenten zu ermöglichen. Ferner zeigt Ihnen dieses Kapitel die XML-Editoren jEdit und Emacs und wie sie bedient werden.

Mein Heim ist meine Burg.
—Sprichwort aus England

B.1 XML-Tools installieren

In diesem Abschnitt werden Sie Programme kennen lernen, die unter einer freien Lizenz stehen und kostenfrei genutzt werden dürfen. Zusammen ergeben sie eine leistungsfähige und flexible XML-Umgebung. Entsprechende Kenntnisse vorausgesetzt, lässt sie sich auch nachträglich nach Ihren Wünschen anpassen oder erweitern.

Vorgefertigte Pakete erhältlich
Möchten Sie sich die Mühe ersparen, gibt es vorgefertigte Pakete für Ihr jeweiliges System unter der Adresse `http://www.vorkon.de`.

Linuxanwender haben es einfach: Sie brauchen lediglich die vorgestellten Programme installieren. Je nach Distribution kann der Name geringfügig abweichen. Verwenden Sie Ihren Paketmanager oder vergleichbare Programme, um die entsprechenden Pakete zu finden.

Windowsanwender haben die folgenden Optionen:

- Falls Sie einen (kommerziellen) XML-Editor bereits besitzen, wird eine Verarbeitungsmaschinerie meist mitgeliefert. Das heisst, Sie brauchen nur die entsprechenden Funktionen Ihres XML-Editors aufzurufen, um Ihr Dokument zu verarbeiten.

Anhang C
Übersicht aller DocBook-Elemente

In diesem Anhang finden Sie eine Übersicht über alle DocBook-Elemente, und zwar nach Themen gruppiert wie auch in alphabetischer Reihenfolge.

C.1 Thematische Sortierung

Absatz-Elemente (Seite 196)
formalpara, para, simpara

Abschnitte (Seite 141)
bridgehead, sect1, sect2, sect3, sect4, sect5, section, simplesect, refnamediv, refsynopsisdiv, refsect1, refsect2, refsect3, refsection

Adressen
address, city, country, email, fax, orgname, otheraddr, phone, pob, postcode, state, street

Anmerkungen (Seite 190)
annotation, alt

Benutzerschnittstellen (Seite 202)
accel, guibutton, guiicon, guilabel, guimenu, guimenuitem, guisubmenu, keycap, keycode, keycombo, keysym, menuchoice, mousebutton, shortcut

Befehlselemente (Seite 183)
arg, cmdsynopsis, command, group, option, optional, parameter, prompt, refsynopsisdiv, replaceable, sbr, synopfragment, synopfragmentref, type

Betriebssystem (Seite 200)
computeroutput, envar, filename, hardware, medialabel, prompt, property, systemitem, userinput

Bibliografie-Struktur (Seite 136)
bibliodiv, biblioentry, bibliomisc, bibliomixed, bibliomset, biblioset

Technische Auszeichnungen
: application, database, email, envar, filename, hardware, medialabel, systemitem, varname

Titel
: caption, segtitle, subtitle, title, titleabbrev

Versionsgeschichten (Seite 143)
: revhistory, revision, revnumber, releaseinfo, revremark, printhistory

Warnhinweise (Seite 192)
: caution, important, note, tip, warning

Zeichengetreue Ausgabe (Verbatim-Elemente) (Seite 172)
: address, classnamesynopsisinfo, computeroutput, funcsynopsisinfo, lineannotation, literal, literallayout, programlisting, screen, screenshot, synopsis, userinput

Zitate und Sinnsprüche (Seite 189)
: ackno, attribution, blockquote, epigraph, quote

C.2 Alphabetische Sortierung

Die Spalte Format beschreibt die grundsätzliche Darstellung des entsprechenden DocBook-Elements: Inline, Block oder unterdrückt. Dabei gibt es einige Kombinationsmöglichkeiten, die in der folgenden Tabelle angegeben werden:

Tabelle C.1. *Erklärung zur Legende (Spalte Format)*

Kürzel	*Bedeutung*
I	Inline. Das Element erscheint direkt im Text, evtl. mit einer Änderung des Schriftstils. Es erfolgt jedoch kein Umbruch.
B	Block. Das Element erzeugt einen Umbruch, das heißt, ein Abstand vor und danach wird eingefügt. Manchmal wird sogar eine neue Seite begonnen (je nach erzeugtem Zielformat).
I|B	Inline oder Block. Das Element wird kontextabhängig formatiert.
U	Unterdrückt. Das Element wird nicht angezeigt. Für gewöhnlich bei Elementen, die rein informativen Charakter besitzen oder keine sinnvolle Darstellung im Zielformat erlauben.
(U)	Möglicherweise unterdrückt. Das Element wird je nach Kontext und je nach erzeugtem Format entweder angezeigt oder unterdrückt.
–	Keine Einteilung möglich sinnvoll oder als Block bzw. Inline-Element zu unterscheiden.

Element	*Format*	*Anwendung*	*Beschreibung*
abbrev[SD]	I	S. 137	Abkürzungen, die durch einen Punkt abgeschlossen werden

Element	*Format*	*Anwendung*	*Beschreibung*
abstract[SD]	B (U)	S. 133	Zusammenfassung eines Kapitels, Abschnitts usw.
accel	I	S. 204	Tastaturabkürzung in einer GUI
ackno[G]	B	S. 579	Danksagung in einem Artikel
acknowledgements[5]	B	S. 130	Danksagung
acronym[SD]	I	S. 577	Kunstwort aus den Anfangsbuchstaben mehrerer Wörter
action[G]	I	S. 577	Antwort auf ein Ereignis (meist von einem Benutzer)
address	B	S. 575	Postadresse
affiliation[SD]	I|B	S. 577	Institutszugehörigkeit eines Autors, einer Person oder eines Mitarbeiters
alt	I|B (U)	S. 186, S. 190	Textdarstellung eines grafischen Objekts
anchor	–	S. 578	Markierung, auf die verwiesen werden kann
annotation	B|U	S. 191	Anmerkung
answer	–	S. 168	Antwort auf eine Frage (von einem qandaset-Element)
appendix[SD]	B	S. 132	Anhang in einem Buch oder Artikel
appendixinfo[G]	U	S. 126	Metainformationen zu einem Anhang
application	I	S. 208	Name eines Softwareprogramms
arc[5]	U	S. 578	XLink Bogen in erweiterte Links
area	U	S. 177	Markierung für Callout-Listen
areaset	U	S. 578	Container für area-Elemente
areaspec	U	S. 177	Container für area/areaset-Elemente
arg	I|B	S. 183	Argument in einem cmdsynopsis-Element
article[SD]	B	S. 133	Artikel
articleinfo[SD,G]	U	S. 133	Metainformationen zu einem Artikel
artpagenums	I	S. 576	Seitenzahl eines veröffentlichten Artikels
attribution[SD]	I|B	S. 189	Quelle eines Zitats oder eines Mottos
audiodata[SD]	–	S. 576	Verweis auf externe Audiodaten

Glossar

A

Allgemeines Entity
: Platzhalter für einen beliebigen Inhalt. Man unterscheidet Entities mit gemischtem Inhalt (Markup und Text), Zeichen-Entities (vordefinierte wie `<`, namensdefinierte wie `&copyright;` und zahlendefinierte wie `&#x27ea;`) sowie ungeparste Entities (Binärdaten).
  Siehe auch Parameter Entity.

Anker
: In diesem Buch eine Bezeichnung für eine Markierung, die im gesamten Dokument einzigartig ist. Je nach DocBook-Version wird sie im Attribut `id` (DocBook 4) oder `xml:id` (DocBook 5) angegeben.

Anpassungsdatei (*Customization layer*)
: Mechanismus um benutzerdefinierte Anpassungen nicht in der Originaldatei, sondern in einer separate Datei (= Anpassungsdatei) vorzunehmen. Die Originaldatei wird in der Anpassungsdatei eingebunden („importiert“). Wird bei Stylesheets und Schemata angewendet.

Attribut
: Paar aus Attributnamen und Wert innerhalb eines Start-Tags, die durch ein Gleichheitszeichen (evtl. durch zusätzlichen Leerraum) getrennt werden. Der Wert wird in einfachen (') oder doppelten Anführungszeichen (") eingeschlossen. Der Datentyp wird durch das Schema bestimmt. Anführungszeichen innerhalb des Attributwerts werden durch `&apos;` bzw. `&quot;` eingegeben.
  Siehe auch Tag.

C

CSS (Cascading Stylesheet)
: W3C-Empfehlung. CSS wird zur Darstellung von Elementen aus XML oder (X)HTML in einem Browser verwendet. Umsortierungen, Ausblendungen bestimmter Elemente usw. sind mit CSS nicht möglich. Hierzu wird XSLT verwendet.

Literaturverzeichnis

Artikel und Informationen über DocBook

[1] Dominik Brettnacher. *Annotate*.
Der Autor beschreibt in diesem Artikel eine Anpassung der DocBook-Stylesheets, mit dem sich mit DocBook produziertes HTML mit Anmerkungen versehen lassen. Dadurch können Leser Ihre Kommentare direkt an der entsprechenden Stelle hinterlassen (für gewöhnlich auf Absatzebene).
http://www.brettnacher.org/annotate/

[2] Norman Walsh. *The Design of the DocBook XSL Stylesheets*.
In diesem Artikel beschreibt Norman Walsh, welche Designüberlegungen es bei den DocBook-Stylesheets gegeben haben und wie sie angewendet wurden.
http://nwalsh.com/docs/articles/dbdesign/

[3] Jiří Kosek. *DocBook for Eclipse: Reusing DocBook's Stylesheets*.
In diesem Artikel werden die Anpassungen beschrieben, um DocBook-Dokumente für das Hilfesystem der Eclipse Plattform zu schreiben und zu verarbeiten.
http://www.xml.com/lpt/a/1265

[4] Jiří Kosek. *Mastering DocBook Indexes*.
Indizes sind ein komplexes Thema, besonders wenn sie über XSLT verarbeitet werden. In dieser Abhandlung zeigt Jiří Kosek wie ein professioneller Index in DocBook erstellt wird und was es zu beachten gibt, wenn Indizes in einer anderen Sprache als Englisch benötigt werden.
http://www.xml.com/lpt/a/1446

[5] Austin King. *Creating an Online Help System with JavaHelp and DocBook*.
JavaHelp ist ein Rahmenwerk für die Online-Hilfe das von Sun Microsystems in Java entwickelt wurde. Dieser Tutorial zeigt, wie Sie Ihre Dokumente so verarbeiten, dass sie sich im JavaHelp-System integrieren lassen.
http://www.onjava.com/lpt/a/4285

[6] Jiří Kosek. *Show me your highlighted code*.
Programmiersprachen besitzen Merkmale wie Parameter, Zeichenketten, Schlüsselwörter, Funktionen usw. Dieser Artikel zeigt, wie Sie diese Merkmale in Ihren Listings automatisch hervorheben lassen. Dadurch sind sie für Ihre Leser ansprechender und einfacher zu erfassen.
http://xmlguru.cz/2006/07/docbook-syntax-highlighting
http://sourceforge.net/projects/xslthl

Index

Eine fettgedruckte Zahl weist auf eine Definition oder den Gebrauch hin.

Symbole

Kolophon

Über den Author

Nach Ausbildung zum Chemielaboranten und Studium der Chemie an der FH Reutlingen ist der Autor heute bei der SUSE Linux Products GmbH angestellt. Dort beschäftigt er sich im Rahmen seiner Tätigkeit als technischer Redakteur mit DocBook, XSL-FO, XSLT und Python.

Über das Buch

Das vorliegende Buch wurde in DocBook 5 mit Hilfe von oXygen 10.1 als XML-Editor geschrieben. Grundlage war openSUSE 10.3/11.0/11.1. In der ersten Auflage wurde der XML-Quellcode nach LaTeX transformiert und in PDF bzw. PostScript übersetzt.

Für diese Auflage wurde erstmals eine XSL-FO-basierte Lösung angewendet. Der Übersetzungsprozess von XSL-FO nach PDF wurde mit Hilfe eines Shell-Skripts vorgenommen. Zum Validieren wurden `xmllint` und `jing` verwendet, für den Transformationsprozess kam `xsltproc` zum Einsatz. Die XSL-FO-Datei wurde mit XEP von RenderX™ formatiert. Als Textschrift wird die Charis von SIL International verwendet, für Überschriften die TheSans und für Listings die TheSansMono Condensed; beide stammen von Lucas de Groot. Die Grafiken wurden entweder direkt in SVG gezeichnet oder in OpenOffice Draw erstellt, nach PDF exportiert und im DocBook-Code entsprechend referenziert.

Technisches

Der Quellcode des gesamten Buches beträgt über 2 MegaByte. Bilder wurden als PNG, SVG oder PDF eingetragen. Die fünf häufigsten Elemente in diesem Buch sind: `tag`, `para`, `entry`, `title` und `listitem`.