Die Fibel für neue Mitarbeiter des FreeBSD-Dokumentationsprojekts

Die Fibel für neue Mitarbeiter des FreeBSD-Dokumentationsprojekts

Version: 43184
2013-11-13 von hrs.
Copyright © 1998-2012 The FreeBSD Documentation Project
Copyright © 1998-2012 The FreeBSD German Documentation Project

Zusammenfassung

Vielen Dank für Ihr Interesse und Ihre Mitarbeit an der FreeBSD-Dokumentation. Jeder Beitrag ist für uns sehr wichtig.

In dieser Fibel wird von der eingesetzten Software bis hin zu den Vorstellungen des FreeBSD-Dokumentationsprojekts alles behandelt, was Sie wissen müssen, wenn Sie sich am FreeBSD-Dokumentationsprojekt beteiligen wollen.

Bitte beachten Sie, dass diese Fibel *jederzeit* unter Bearbeitung und noch nicht vollständig ist.

Redistribution and use in source (SGML DocBook) and 'compiled' forms (SGML, HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code (SGML DocBook) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

2. Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Wichtig

THIS DOCUMENTATION IS PROVIDED BY THE FREEBSD DOCUMENTATION PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FREEBSD DOCUMENTATION PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Inhaltsverzeichnis

Benutzungshinweise vii
1. Die Eingabeaufforderungen vii
2. Typographische Festlegungen vii
3. Anmerkungen, Tipps, wichtige Hinweise, Warnungen und Beispiel viii
4. Danksagungen ix
1. Überblick 1
1.1. Die FreeBSD-Dokumentationsreihe 1
1.2. Bevor es losgeht 2
1.3. Der Schnellstart 2
2. Die Werkzeuge 5
2.1. Notwendige Werkzeuge 6
2.2. Optionale Werkzeuge 7
3. Die XML-Fibel 9
3.1. Überblick 9
3.2. Von Elementen, Tags und Attributen 11
3.3. Die DOCTYPE-Deklaration 18
3.4. Die Rückkehr zu SGML 21
3.5. Kommentare 22
3.6. Entitäten 23
3.7. Dateien mit Entitäten einbinden 26
3.8. Markierte Bereiche 30
3.9. Schlussbemerkung 34
4. XML-Dokumente erstellen 35
4.1. XHTML 35
4.2. Die DocBook DTD 46
5. Stylesheets 81
5.1. DSSSL 81
5.2. CSS 81
6. Verzeichnisstruktur unter `doc/` 83
6.1. `doc/` als höchste Ebene 83
6.2. Die Verzeichnisse `Sprache.Kodierung/` 84
6.3. Dokumentenspezifische Informationen 84
7. Die Erzeugung der Zieldokumente 87
7.1. Für den Bau der FreeBSD-Dokumentation benötigte Werkzeuge 87
7.2. Die `Makefile`s des Dokumentationsbaums verstehen 87
7.3. Make-Includes des FreeBSD Documentation Projects 90
8. Die Webseite 95
8.1. Vorbereitung 95
8.2. Die Webseiten bauen 96
8.3. Installieren der Webseiten auf Ihrem Server 96
8.4. Umgebungsvariablen 97
9. Übersetzungen 99
10. Der Schreibstil 105
10.1. Anleitungen für einen korrekten Schreibstil 106

10.2. Häufig verwendete Wörter 110
11. `sgml-mode` und Emacs 113
12. Weiterführende Quellen 115
12.1. Das FreeBSD-Dokumentationsprojekt 115
12.2. XML 115
12.3. HTML 115
12.4. DocBook 115
12.5. Das Linux-Dokumentationsprojekt 115
A. Beispiele 117
A.1. DocBook-Buch (`book`) 117
A.2. DocBook-Artikel (`article`) 118
A.3. Ausgabeformate erzeugen 119
Stichwortverzeichnis 123

Liste der Beispiele

1. Ein Beispiel ix
3.1. Verwendung eines Elements (Start- und Endtag) 12
3.2. Verwendung eines Elements (nur Starttag) 13
3.3. Verschachtelte Elemente: em 13
3.4. Elemente mit Attributen nutzen 14
3.5. Attribute mit einfachen Anführungszeichen 14
3.6. .profile, für sh(1) und bash(1) Benutzer 15
3.7. .cshrc, für csh(1)- und tcsh(1)-Benutzer 16
3.8. Beispiele für Kommentare in SGML 22
3.9. Fehlerhafte SGML-Kommentare 22
3.10. Allgemeine Entitäten festlegen 24
3.11. Parameterentitäten festlegen 25
3.12. Dateien mit Allgemeinen Entitäten einbinden 27
3.13. Dateien mit Parameterentitäten einbinden 28
3.14. Aufbau eines markierten Bereiches 30
3.15. CDATA als Inhaltsmodell für markierte Bereiche 31
3.16. Anwendung von INCLUDE und IGNORE in markierten Abschnitten 32
3.17. Kontrolle von markierten Bereichen über Parameterentitäten 33
4.1. Die Struktur eines XHTML-Dokumentes 36
4.2. h1, h2... 37
4.3. Falsche Verschachtelung von Überschriften 37
4.4. Absätze mit dem Element p 38
4.5. Blockzitat 38
4.6. Listen mit ul und ol erstellen 39
4.7. Definitionslisten mit dl erstellen 40
4.8. Vorformatierten Text mit pre erstellen 40
4.9. Einfache Tabelle mit table 41
4.10. Anwendung des Attributes rowspan 42
4.11. Anwendung des Attributes colspan 42
4.12. Gemeinsame Anwendung der Attrbute rowspan und colspan 43
4.13. Text mit em und strong hervorheben 44
4.14. Nicht-proportionale Schrift mit tt 44
4.15. <a href="..."> benutzen 45
4.16. Anwendung von <a id="..."> 45
4.17. Auf einen Abschnitt eines anderen Dokumentes verweisen 45
4.18. Buchvorlage book mit bookinfo 48
4.19. Artikelvorlage article mit articleinfo 49
4.20. Ein einfaches Kapitel 49
4.21. Ein leeres Kapitel 50
4.22. Unterkapitel 50
4.23. Absatz mit para 52
4.24. blockquote 52
4.25. warning 54
4.26. itemizedlist, orderedlist und procedure 55

4.27. programlisting 56
4.28. Das co- und das calloutlist-Element 57
4.29. Tabellen mittels informaltable auszeichnen 58
4.30. Tabelle mit Attribut frame="none" 59
4.31. screen, prompt und userinput 60
4.32. Das Element emphasis 61
4.33. Richtig zitieren 62
4.34. Tasten, Maustasten und Tastenkombinationen 62
4.35. Anwendungen, Befehle und Optionen 64
4.36. Das Element filename 65
4.37. Portsnamen und das Element filename 66
4.38. Gerätenamen per devicename auszeichnen 67
4.39. role und hostid 68
4.40. Das Element username 69
4.41. maketarget und makevar 70
4.42. literal 71
4.43. Das Element replaceable 72
4.44. Das Element errorname 72
4.45. chapter und section mit dem Attribut id 77
4.46. Querverweise und das Element anchor 77
4.47. Einsatz von xref 78
4.48. link beutzen 79
4.49. Verweise mit ulink 80
A.1. Ein DocBook-Buch (book) 117
A.2. Ein DocBook-Artikel (article) 118
A.3. Ein DocBook-Dokument in eine einzelne HTML-Datei umwandeln 119
A.4. Ein DocBook-Dokument in mehrere kleine HTML-Dateien umwandeln 120
A.5. Ein DocBook-Dokument nach Postscript umwandeln 121
A.6. Eine PDF-Datei aus einem DocBook-Dokument erzeugen 122

Benutzungshinweise

1. Die Eingabeaufforderungen

Die folgende Tabelle zeigt die normale Eingabeaufforderung des Systems und die Eingabeaufforderung des Superusers. Die in diesem Buch vorkommenden Beispiele benutzen die jeweilige Eingabeaufforderung, um zu zeigen, unter welchem Benutzer die Beispiele ausgeführt werden sollten.

Benutzer	Eingabeaufforderung
`Normaler Benutzer`	`%`
`Superuser`	`#`

2. Typographische Festlegungen

Um die Lesbarkeit zu erhöhen, werden in diesem Dokument die im folgenden genannten typographischen Festlegungen verwendet:

Bedeutung	Beispiel
Kommandonamen	Geben Sie `ls -a` ein, um alle Dateien anzuzeigen.
Datei- und Verzeichnisnamen	Bearbeiten Sie `.login`.
Bildschirmein- und ausgaben	`You have mail.`
Referenzen auf Hilfeseiten	Mit su(1) können Sie sich als ein anderer Benutzer anmelden.
Benutzer- und Gruppennamen	Ich bin `root`, ich darf das.
Hervorhebungen	Hier *müssen* Sie vorsichtig sein.
Argumente auf der Kommandozeile, die durch existierende Namen, Dateien oder Variablen ersetzt werden müssen	Dateien können Sie mit dem Befehl `rm Dateiname` löschen.
Umgebungsvariablen	`$HOME` ist Ihr Benutzerverzeichnis.

3. Anmerkungen, Tipps, wichtige Hinweise, Warnungen und Beispiel

An einigen Stellen innerhalb dieses Buchs werden wichtige oder nützliche Hinweise gegeben, die besonders hervorgehoben sind. Hier ein kurzer Überblick über die verwendeten Darstellungen.

Anmerkung

Anmerkungen werden so dargestellt. Sie enthalten Informationen die Sie nur zu lesen brauchen, wenn Sie direkt davon betroffen sind.

Tipp

Tipps sind Informationen, die vielleicht hilfreich sein könnten oder aufzeigen, wie bestimmte Dinge einfacher zu bewerkstelligen sind.

Wichtig

Besonders wichtige Punkte werden so hervorgehoben. Meist enthalten sie Hinweise auf vielleicht zusätzlich auszuführende Schritte oder Dinge, die besonders zu beachten sind.

Warnung

Warnungen werden wie dieser Abschnitt dargestellt und weisen auf mögliche Schäden hin, die entstehen können, falls die beschriebenen Schritte nicht genau befolgt oder Hinweise nicht beachtet werden. Die Palette der möglichen Schäden reicht von Hardwareschäden bis hin zu Datendatenverlust durch ein versehentliches Löschen von wichtigen Dateien oder ganzen Verzeichnissen.

Beispiel 1. Ein Beispiel

Beispiele, die so wie hier dargestellt werden, enthalten meist kleine Übungen, die nachvollzogen werden sollten, um das vorher beschriebene besser zu verinnerlichen oder mit den erzeugten Ausgaben vertraut zu werden.

4. Danksagungen

Ich möchte mich bei Sue Blake, Patrick Durusau, Jon Hamilton, Peter Flynn und Christopher Maden bedanken, die sich die Zeit genommen haben, die frühen Entwürfe dieses Dokuments zu lesen und viele hilfreiche Hinweise und Ratschläge gegeben haben.

Kapitel 1. Überblick

Herzlich Willkommen beim FreeBSD-Dokumentationsprojekt. Qualitativ hochwertige Dokumentation ist ein wichtiger Erfolgsfaktor und sehr bedeutend für die Verbreitung von FreeBSD. Die wichtigste Quelle dafür ist das FreeBSD-Dokumentationsprojekt (FDP). Jeder Beitrag, der zu diesem Projekt geleistet wird, ist ungemein wertvoll.

Es ist das Anliegen dieser Fibel, den Leser mit dem FDP vertraut zu machen und zu erklären, *wie das FDP organisiert ist, wie man selber Dokumente erstellt und an das FDP einreicht* und *wie die verfügbaren Werkzeuge effektiv beim Schreiben eingesetzt werden können.*

Wie jedes Open-Source-Projekt, ist auch das FDP auf die Mithilfe vieler angewiesen. Deshalb ist jeder herzlich eingeladen mitzuarbeiten. Die dafür erforderlichen Voraussetzungen sind gering und es gibt keine Verpflichtung eine bestimmte Menge an Dokumenten pro Monat oder Jahr beizusteuern. Das Einzige, was Sie tun müssen, ist sich auf der Mailingliste FreeBSD documentation project einzutragen.

Nach dem Lesen der FDP-Fibel sollte man wissen:

- welche Dokumente durch das FDP betreut werden,
- wie man SGML-Dokumente liest und den SGML-Quellcode der durch das FDP betreuten Dokumente versteht,
- wie man selbst Änderungen an Dokumenten vornehmen kann und
- wie man Änderungen zur Begutachtung durch das FDP einreichen kann.

1.1. Die FreeBSD-Dokumentationsreihe

Das FDP umfasst vier verschiedene Kategorien:

Manualpages
: Die englischen Manualpages wurden nicht vom FDP geschrieben, da sie ein Teil des Basissystems sind. Jedoch können bzw. wurden bereits Teile von existierenden Manualpages umformuliert, um sie verständlicher zu machen oder um Fehler zu beheben.

  Für die Übersetzung der Manualpages des Systems in die verschiedenen Sprachen sind die einzelnen Übersetzergruppen verantwortlich. Alle dabei entstandenen Übersetzungen gehören zum FDP.

Die FAQ
: Das Ziel der FAQ ist es, Fragen, die auf den verschiedenen Mailinglisten und in Newsgruppen regelmäßig diskutiert werden, nach einem einfachen Frage- und Ant-

wort-Muster zu behandeln. Das schließt nicht aus, das auf bestimmte Fragen ausführlich und umfassend eingegangen wird.

Das Handbuch
: Das Ziel des Handbuches ist es, die umfassende Quelle und Referenz im Netz für FreeBSD-Benutzer zu sein.

Die Webseite
: Die Webseite http://www.FreeBSD.org und ihre vielen Spiegel auf der ganzen Welt vertreten das FreeBSD-Projekt im WWW. Für viele Menschen ist sie der erste Kontakt mit FreeBSD.

Die Quellen für die FreeBSD-Website, das FreeBSD Handbuch sowie die FreeBSD FAQ werden im `doc/` Subversion-Repository von FreeBSD verwaltet, das Sie über `https://svn.FreeBSD.org/doc/` erreichen können.

Manualpages werden im `src/` Subversion-Repository von FreeBSD verwaltet, das Sie über `https://svn.FreeBSD.org/base/` erreichen können.

Das bedeutet, dass alle Änderungen an den Dateien für jeden verfügbar sind und sich jeder mit svn eine lokale Kopie der Dokumentation anlegen kann.

Parallel zum FDP haben viele Menschen Anleitungen geschrieben und Webseiten mit Bezug zu FreeBSD erstellt. Einige davon werden im Subversion-Archiv verwaltet, sofern der Autor dem zugestimmt hat. In anderen Fällen hat sich der Autor entschlossen, seine Dokumentation außerhalb des zentralen FreeBSD-Archivs zu verwalten. Das FDP bemüht sich, so viele Verweise wie möglich auf solche Quellen bereitzustellen.

1.2. Bevor es losgeht

Zum Verständnis der folgenden Kapitel sollte folgendes bereits bekannt sein:

- Wie eine aktuelle lokale Kopie des FreeBSD Subversion-Repository mit Hilfe von svn angelegt und gepflegt werden kann.
- Wie neue Programme mit Hilfe des FreeBSD-Portsystems oder mittels pkg_add(1) heruntergeladen und installiert werden.

1.3. Der Schnellstart

Falls man einfach loslegen möchte und sich sicher genug fühlt, um alles weitere erst bei Bedarf nachzusehen, kann man einfach den folgenden Anweisungen folgen:

1. Zuerst muß der Metaport textproc/docproj auf dem betreffenden Arbeitsrechner installiert werden.

```
# cd /usr/ports/textproc/docproj
# make JADETEX=no install
```

2. Laden Sie mit svn eine lokale Kopie des FreeBSD-doc-Verzeichnisbaumes herunter.

   Selbst wenn Sie nur über eine geringe Bandbreite oder wenig freien Plattenplatz verfügen, müssen Sie mindestens die Verzeichnisse `head/share` sowie `head/language/share` auschecken, um an der Dokumentation arbeiten zu können. Dazu ein Beispiel:

   ```
   % mkdir -p head/share
   % mkdir -p head/en_US.ISO8859-1/share
   % svn checkout https://svn0.us-
   east.FreeBSD.org/doc/head/share head/share
   % svn checkout https://svn0.us-
   east.FreeBSD.org/doc/head/en_US.ISO8859-1/share head/en_US.ISO8859-1/share
   ```

   Für den Fall, dass ausreichend Platz auf der Festplatte vorhanden ist, kann auch eine vollständige Arbeitskopie des gesamten Subversion-Baumes anlegt werden.

   ```
   % svn checkout https://svn0.us-
   east.FreeBSD.org/doc/head head
   ```

   ### Anmerkung

   svn0.us-east.FreeBSD.org ist ein öffentlicher Server. Wählen Sie einen Mirror in Ihrer Nähe und überprüfen Sie das Serverzertifikat auf der Seite Subversion mirror sites.

3. Sollte geplant sein, ein existierendes Buch oder einen existierenden Artikel zu ändern, muß natürlich noch zusätzlich das betreffende Verzeichnis aus dem CVS-Archiv geholt werden. Soll hingegen ein neues Buch oder ein neuer Artikel geschrieben werden, empfiehlt es sich, auf bestehende Bücher und Artikel zurückzugreifen und diese als Vorlage zu nutzen.

   Ein Artikel über die Konfiguration eines VPNs zwischen FreeBSD und Windows 2000 kann wie folgt erstellt werden:

   1. Zuerst wird das Verzeichnis `articles` aus dem FreeBSD-CVS-Archiv lokal angelegt:

      ```
      % svn checkout https://svn0.us-
      east.FreeBSD.org/doc/head/en_US.ISO8859-1/articles
      ```

2. Anschließend kopiert man einen bereits existierenden Artikel und nutzt ihn als Vorlage. In diesem Beispiel soll der neue Artikel im Verzeichnis `vpn-w2k` liegen:

```
% cd head/en_US.ISO8859-1/articles
% svn export committers-guide vpn-w2k
```

Bereits existierende Dokumente, die geändert werden sollen, können direkt aus dem CVS-Archiv geholt werden. Das folgende Beispiel zeigt das für die FAQ aus dem Verzeichnis `head/en_US.ISO8859-1/books/faq` :

```
% svn checkout https://svn0.us-
east.FreeBSD.org/doc/head/en_US.ISO8859-1/books/faq
```

4. Jetzt können die `.xml` Dateien mit einem beliebigen Texteditor bearbeitet werden.

5. Danach ist `make` mit dem Ziel `lint` aufzurufen, um das gesamte Dokument auf Auszeichnungsfehler hin zu untersuchen, ohne dass zeitaufwändige Transformationen vorgenommen werden.

```
% make lint
```

Soll anschließend das Zieldokument erstellt werden, kann mit Hilfe der Variable `FORMATS` bestimmt werden, welche Ausgabeformate erzeugt werden sollen. Unterstützt werden momentan `html`, `html-split` , `txt`, `ps`, `pdf` und `rtf`. Die aktuelle Liste der unterstützten Formate befindet sich am Anfang der Datei `head/share/mk/doc.docbook.mk` . Bei der Verwendung dieser Variable ist es wichtig, darauf zu achten, dass die Angabe der gewünschten Formate in Anführungszeichen eingeschlossen wird, sofern mehr als nur ein Format gleichzeitig erstellt werden soll.

Wenn das Dokument beispielsweise nach `HTML` konvertiert werden soll, kann dies so vorgenommen werden:

```
% make FORMATS=html
```

Soll es hingegen in den Formaten `html` und `txt` erzeugt werden, kann man entweder make(1) zweimal hintereinander aufrufen:

```
% make FORMATS=html
% make FORMATS=txt
```

oder beide Formate mit einem Aufruf von make(1) erzeugen:

```
% make FORMATS="html txt"
```

6. Zum Schluss müssen die Änderungen an das FDP mittels send-pr(1) eingesandt werden.

Kapitel 2. Die Werkzeuge

Innerhalb des FDPs werden verschiedene Programme für die Verwaltung der FreeBSD-Dokumentation, ihrer Transformation in verschiede Formate und weitere Aufgaben eingesetzt. Wer an der FreeBSD-Dokumentation mitarbeiten möchte, wird diese Programme benötigen.

Doch dies ist kein Grund zur Angst, da alle notwendigen Programme als FreeBSD-Ports und fertige Pakete vorhanden sind, wodurch sich die Installation drastisch vereinfacht.

Allerdings müssen diese Programme installiert werden, bevor alle Beispiele der folgenden Kapitel ausprobiert werden können.

Wenn es möglich ist, sollte der Port textproc/docproj verwendet werden

Durch die Installation des Ports textproc/docproj kann die Installation vereinfacht und eine Menge Zeit gespart werden. Bei diesem Port handelt es sich um einen *Metaport*, der selbst keine Programme oder ähnliches installiert. Stattdessen enthält er eine Vielzahl von Abhängigkeiten zu anderen Ports und setzt deren korrekte Installation voraus. Durch seine Installation *sollten* automatisch alle Pakete, die in diesem Kapitel genannt werden, auf den Rechner geladen und dort installiert werden.

Ein nur unter bestimmten Umständen benötigter Port ist das JadeTeX-Makro-Paket, das seinerseits eine TeX-Installation voraussetzt. TeX ist ein ziemlich großes Programmpaket und sollte nur installiert werden, sofern Zieldokumente im PostScript- oder PDF-Format generiert werden sollen.

Um den Platz und die Zeit für die Installation von JadeTeX und TeX zu sparen, muß bei der Installation angeben werden, ob JadeTeX (und damit auch TeX) installiert werden soll oder nicht.

Daher sollte der docproj-Port entweder mit

```
# make JADETEX=yes install
```

oder mit

```
# make JADETEX=no install
```

installiert werden, je nachdem was gewünscht wird. Alternativ können Sie auch direkt die Ports textproc/docproj-jadetex oder

textproc/docproj-nojadetex installieren. Die Variable `JADETEX` wird von diesen Ports automatisch entsprechend gesetzt. Ohne JadeTeX können Sie nur die Formate HTML und ASCII erzeugen. Die Formate PostScript und PDF erfordern TeX.

2.1. Notwendige Werkzeuge

2.1.1. Software

Die folgenden Programme sind notwendig, um sinnvoll an der FreeBSD-Dokumentation arbeiten und diese in andere Formate wie HTML, reinen Text und RTF umwandeln zu können. Sie müssen diese aber nicht separat installieren, da alle Programme automatisch durch den Metaport textproc/docproj installiert werden.

Jade (textproc/jade)
: Eine DSSSL-Implementierung. Sie wird gebraucht, um Dokumente in andere Formate wie HTML und TeX zu übersetzen.

Tidy (www/tidy)
: Ein Formatierer, mit dem man Teile der automatisch generierten HTML-Dateien neuformatieren kann, um ihre Lesbarkeit zu erhöhen.

Links (www/links)
: Ein Textbrowser, der HTML-Dateien in einfache Textdateien umwandeln kann.

peps (graphics/peps)
: Einige der Dokumente enthalten Grafiken, die nur im EPS-Format vorliegen. Damit diese von dem meisten Webbrowsern angezeigt werden können, müssen sie nach PNG konvertiert werden.

2.1.2. Die DTDs und die Entitäten

Das FDP benutzt verschiedene DTDs und Entitätensätze, die installiert sein müssen, bevor mit der Arbeit an einem beliebigen Dokument begonnen werden kann.

HTML DTD (textproc/html)
: HTML ist die bevorzugte Auszeichnungssprache des World Wide Web und wird durchgängig für die FreeBSD-Webseite genutzt.

DocBook DTD (textproc/docbook)
: DocBook ist als Auszeichnungssprache für technische Dokumentationen entwickelt worden. Die gesamte FreeBSD-Dokumentation wird mittels DocBook erstellt.

ISO 8879-Entitäten (textproc/iso8879)
: 19 der ISO 8879:1986-Zeichensätze, die von vielen DTDs benötigt werden. Darin enthalten sind mathematische Symbole, zusätzliche Zeichen, die für auf dem lateinischen beruhende Alphabete benötigt werden sowie griechische Zeichen.

2.1.3. Die Stilvorlagen

Die Stilvorlagen werden während der Transformation und der Formatierung von Dokumenten, beispielsweise für die Bildschirmdarstellung oder den Druck, benutzt.

Modular DocBook Stylesheets (textproc/dsssl-docbook-modular)
: Die Modular DocBook Stylesheets werden benötigt, wenn mittels DocBook erstellte Dokumente in Formate wie HTML oder RTF konvertiert werden sollen.

2.2. Optionale Werkzeuge

Die in diesem Kapitel genannten Programme müssen nicht unbedingt installiert werden. Allerdings können sie die Arbeit an der Dokumentation erleichtern und die Anzahl an möglichen Ausgabeformaten erhöhen.

2.2.1. Software

JadeTeX und teTeX (print/jadetex und print/teTeX)
: Jade und teTeX werden eingesetzt, um DocBook-Dokumente nach DVI, Postscript und PDF zu konvertieren. Hierfür müssen die JadeTeX Makros installiert sein.

  Ist es nicht geplant, die Dokumente in einem dieser Formate zu erzeugen, wenn also HTML, Text und RTF ausreichend sind, brauchen JadeTeX und teTeX nicht installiert zu werden. Da die Installation von teTeX insgesamt 30 MB benötigt, kann so Zeit und Plattenplatz gespart werden.

Wichtig

Wird sich für die Installation von JadeTeX und teTeX entschieden, muß teTeX anschließend noch eingerichtet werden. Die Datei `print/jadetex/pkg-message` enthält detailierte Angaben zu den dafür notwendigen Schritten.

Emacs oder XEmacs (editors/emacs oder editors/xemacs)
: Beide Texteditoren haben einen speziellen Modus zur Bearbeitung von SGML-Dokumenten entsprechend den Vorgaben einer SGML-DTD. Zusätzlich bieten sie Funktio-

nen an, mit denen sich der Tippaufwand reduzieren und Fehlerwahrscheinlichkeit senken läßt.

Natürlich muß nicht mit einem dieser Texteditoren gearbeitet werden; jeder andere Editor kann dafür genausogut genutzt werden, doch vielleicht wird die Arbeit durch sie als effektiver empfunden werden.

Sofern Sie Vorschläge haben, welche andere Software für die Verarbeitung oder Bearbeitung von SGML-Dokumenten in diese Liste mitaufgenommen werden sollte, senden Sie diese bitte an Documentation Engineering Team <doceng@FreeBSD.org >.

Kapitel 3. Die XML-Fibel

Die Mehrzahl der Dokumente des FDPs sind in XML geschrieben. Ziel dieses Kapitels ist es, genau zu erklären, was das bedeutet und wie man die XML-Quellen liest und versteht. Ebenso werden die in den Quellen genutzten Kniffe erklärt, auf die man beim Lesen der Dokumente stoßen wird.

Teile dieses Kapitels basieren auf Mark Galassis „Get Going With DocBook“.

3.1. Überblick

In den guten alten Zeiten war der Umgang mit „elektronischem“ Text einfach. Man musste lediglich wissen, welcher Zeichensatz (ASCII, EBCDIC oder ein anderer) vorlag. Text war einfach Text und sah so aus, wie man ihn sah. Keine Extras, keine Formatierungen und kein sonstiger Schnickschnack.

Für viele Zwecke war dies allerdings nicht ausreichend. Von einem maschinenlesbaren Text wird erwartet, dass er auch von Maschinen gelesen und intelligent weiterverarbeitet werden kann. Einzelne Stellen sollen hervorgehoben werden, andere sollen in ein Glossar aufgenommen werden oder auf andere Textstellen verweisen. Dateinamen wiederum sollen in einer „schreibmaschinenähnlichen“ Schrift auf dem Bildschirm dargestellt werden, der Ausdruck soll jedoch in „Schrägschrift“ oder in einer beliebigen anderen Darstellungsform erfolgen.

Anfänglich gab es die Hoffnung, dass die Künstliche Intelligenz (KI) helfen würde, dieses Ziel zu erreichen. Computer sollte den Text lesen und dazu in der Lage sein, selbstständig wichtige Formulierungen, Dateinamen, Benutzereingaben oder Beispiele zu erkennen. Leider verlief die Entwicklung in diesem Bereich nicht wie gewünscht und Computer benötigen nach wie vor etwas Unterstützung, bevor sie Texte vernünftig verarbeiten können.

Genauer gesagt, man muss ihnen sagen, was was ist. Sehen wir uns folgende Zeilen an:

> Löschen Sie `/tmp/foo` mittels rm(1).
>
> ```
> % rm /tmp/foo
> ```

Es fällt uns leicht, zu erkennen, was ein Dateiname, ein einzugebender Befehl oder ein Verweis auf eine Hilfeseite ist. Das kann ein Computer, der einen Text verarbeitet, nicht. Aus diesem Grund ist es notwendig, Texte mit weiteren Informationen „auszuzeichnen“.

Der Begriff „Auszeichnung[1]“ bedeutet, dass sich der Wert eines Textes erhöht, aber auch seine Kosten. Durch Auszeichnungen wird einem Dokument zusätzlicher Text hinzuge-

[1] Im angelsächsischen Sprachraum wird von ‚markup‘ gesprochen.

fügt, der aber von dem eigentlichen Dokumenteninhalt auf eine bestimmte Art und Weise unterschieden werden kann, so dass Programme die Auszeichnung erkennen können und mittels dieser Informationen während der Verarbeitung in der Lage sind, Entscheidungen zu treffen. Texteditoren können diese Auszeichnungselemente vor dem Benutzer verbergen, um zu vermeiden, dass er durch sie abgelenkt wird.

Die durch die Auszeichnungselemente im Textdokument zusätzlich abgelegten Informationen erhöhen den Wert des Dokuments. Allerdings muss diese Arbeit in den meisten Fällen von einem Menschen getan werden – wären Maschinen dazu fähig, wären zusätzliche Auszeichnungselemente unnötig. Der damit verbundene Aufwand *erhöht die Kosten*, die durch die Erstellung des Dokuments entstehen.

Das etwas weiter oben gegebene Beispiel sieht im Quelltext so aus:

```
<para>Löschen Sie <filename>/tmp/foo</filename> mittels &man.rm.1;.</
para>

<screen>&prompt.user; <userinput>rm /tmp/foo</userinput></screen>
```

Die Auszeichnungselemente sind deutlich vom eigentlichen Inhalt zu unterscheiden.

Die Einführung von Auszeichnungselementen setzt voraus, dass festgelegt wird, welche Bedeutung einzelne Elemente haben und wie diese interpretiert werden. Sie brauchen daher eine Auszeichnungssprache, der Sie folgen, wenn Sie eigene Dokumente verfassen.

Natürlich kann es keine universelle Auszeichnungssprache geben und eine einzige mag nicht ausreichend für alle möglichen Anwendungsfälle sein. Eine Sprache für technische Dokumente wird sich wahrscheinlich stark von einer für Kochrezepte unterscheiden. Die universelle Lösung ist eine Basissprache, mit deren Hilfe weitere Sprachen entwickelt werden können – eine *Meta-Auszeichnungssprache* also.

Genau diese Anforderung wird von der Standard Generalized Markup Language (SGML) erfüllt. Mit ihrer Hilfe wurden viele andere Auszeichnungssprachen wie beispielsweise HTML und DocBook, welche beide von FDP genutzt werden, entwickelt.

Die eigentliche Sprachdefinition erfolgt in einer Dokumenten-Typ-Definition (DTD). Innerhalb dieser DTD werden die Namen der einzelnen Elemente, deren mögliche Reihenfolge und Verschachtelung sowie weitere Informationen festgelegt.

Eine DTD ist eine *vollständige* Definition aller möglichen Sprachelemente, ihrer Reihenfolge[2], optionaler Elemente und so weiter und so weiter. Dank dieser recht formalen Festlegung ist es möglich, SGML-*Parser* zu entwickeln, die sowohl ein Dokument als auch sei-

[2]Bei natürlichen Sprachen spricht man vom Satzbau – demjenigen Konstrukt, das unter anderem die Position des Subjekts, Objekts und Prädikats in einem Satz festlegt.

ne DTD einlesen und anhand dieser DTD prüfen können, ob das Dokument allen Anforderungen der DTD entspricht. Dieser Vorgang wird allgemein als *Validierung des Dokuments* bezeichnet.

Anmerkung

Das Validieren eines SGML-Dokuments gegen eine DTD überprüft lediglich die korrekte Syntax des Dokuments, dass heißt, ob nur gültige Auszeichnungselemente verwendet wurden und ihre Reihenfolge stimmt. Dabei wird *nicht* geprüft, ob die Elemente der DTD *sinngemäß* verwandt wurden. Sollten beispielsweise alle Dateinamen als Funktionsnamen ausgezeichnet worden sein, so würde der Parser keinen Fehler signalisieren. Formaler ausgedrückt: Der Parser prüft die Syntax, aber nicht die Semantik.

Es ist anzunehmen, dass, wenn man selber vor hat Dokumentation für das FDP zu schreiben, der größte Teil davon mit Hilfe von HTML oder DocBook geschrieben werden wird. Aus diesem Grunde wird an dieser Stelle nicht erklärt, wie eine DTD entwickelt wird.

3.2. Von Elementen, Tags und Attributen

Alle in SGML geschriebenen DTDs haben bestimmte gemeinsame Eigenschaften. Das ist nicht verwunderlich, da sich die hinter SGML stehende Idee unweigerlich bemerkbar macht. Zwei der markantesten Merkmale dieser Idee sind die Begriffe *Inhalt* und *Element*.

Von einem Dokument, unabhängig, ob es sich um eine einzelne Webseite oder ein langes Buch handelt, wird angenommen, dass es einen wie auch immer gearteten Inhalt hat. Dieser lässt sich selbst wiederum in Teilelemente aufspalten, die ebenso zerlegbar sind. Durch die Aufnahme von Auszeichnungselementen in einen Text, werden diese einzelnen Elemente eindeutig benannt und voneinander abgegrenzt.

Nimmt man zum Beispiel ein typisches Buch, so kann man es auf der obersten Ebene als ein Ganzes, als ein Element betrachten. Dieses „Buch“-Element enthält nun Kapitel, die wiederum selbst als Elemente bezeichnet werden können. Jedes einzelne Kapitel enthält weitere Elemente. So gibt es beispielsweise Absätze, Zitate und Fußnoten. Jeder Absatz kann wiederum selbst Elemente enthalten, die helfen, den Absatzinhalt als direkte Rede oder als Namen eines der Protagonisten einer Geschichte zu identifizieren.

Wenn man möchte, kann man sich das als „Unterteilung“[3] des Inhalts vorstellen. Auf der obersten Ebene gibt es ein Element: das Buch selbst. Schaut man ein wenig tiefer, findet

[3] Im angelsächsischen Sprachraum wird hier von „chunking“ gesprochen.

man weitere Teilelemente: die einzelnen Kapitel. Diese sind wiederum unterteilt in Absätze, Fußnoten, Namen und so weiter und so weiter.

Anzumerken ist an dieser Stelle, dass das eben gesagte ohne weiteres auf jeden Inhaltstyp angewandt werden kann, auch ohne dass von SGML die Rede ist. Man könnte beispielsweise einfach verschiedene Stifte nehmen und einen Ausdruck dieser Fibel vor sich hinlegen und dann mit verschiedenen Farben die einzelnen Abschnitte des Buchinhalts markieren.

Leider gibt es keinen elektronischen Stift, um das zu tun. Deshalb muss ein anderer Weg gewählt werden, um zu bestimmen, zu welchem Element die einzelnen Inhalte gehören. In SGML-basierten Auszeichnungssprachen wie HTML und DocBook werden dafür so genannte *Tags* eingesetzt.

Mit einem solchen Tag wird eindeutig festgelegt, wo ein bestimmtes Element beginnt und wo es endet. *Allerdings gehört der Tag selber nicht zum Element.* Er legt lediglich die Grenzen des Elements fest. Da jede DTD mit dem Ziel entwickelt wurde, einen speziellen Inhaltstyp auszuzeichnen, wird jede DTD verschiedene Elemente kennen, die daher natürlich auch unterschiedlich benannt sein werden.

Der Starttag für ein imaginäres Element mit dem Namen *`elementname`* ist *`<elementname>`*. Sein Gegenstück, der schließende Endtag, ist *`</elementname>`*.

Beispiel 3.1. Verwendung eines Elements (Start- und Endtag)

HTML kennt das Element `p`, um festzulegen, dass ein bestimmter abgegrenzter Bereich einen Absatz darstellt. Dieses Element hat sowohl einen Start- als auch einen Endtag.

```
<p>Das ist ein Absatz. Er beginnt mit Starttag
  für das Element 'p' und endet mit dem Endtag für
  das Element 'p'.</p>

<p>Das ist ein etwas kürzerer Absatz.</p>
```

Elemente müssen nicht notwendigerweise einen Endtag haben. Ebenso ist es nicht notwendig, dass Elemente einen Inhalt haben. Beispielsweise kann in HTML-Dokumenten mittels eines speziellen Elements festgelegt werden, dass eine horizontale Linie an einer bestimmten Stelle erscheinen soll. Da dieses Element offensichtlich keinen Inhalt hat, wird auch kein Endtag benötigt.

Beispiel 3.2. Verwendung eines Elements (nur Starttag)

In HTML kann man mit dem Element `hr` festlegen, dass an einer bestimmten Stelle eine horizontale Linie angezeigt werden soll. Da dieses Element keinen Inhalt umschließt, hat es nur einen Starttag.

```
<p>Das ist ein Abschnitt.</p>

<hr>

<p>Das ist ein weiterer Absatz. Eine horizontale Linie
  trennt ihn vom vorherigen Absatz.</p>
```

Elemente können andere Elemente enthalten. Im anfangs erwähnten Buch enthielt das Buch-Element alle Kapitel-Elemente, die wiederum alle Absatz-Elemente enthielten und so fort.

Beispiel 3.3. Verschachtelte Elemente: `em`

```
<p>Das ist ein einfacher <em>Abschnitt</em>, in dem
  einige <em>Worte</em> <em>hervorgehoben</em> wurden.
```

Welche Elemente andere Elemente enthalten können und welche das sind, wird innerhalb der DTD eines Dokuments festgelegt.

Wichtig

Viele Leute sind oft verwirrt, wenn es um die richtige Benutzung der Begriffe Tag und Element geht. Im Ergebnis werden sie oft so genutzt, als wären sie austauschbar. Allerdings sind sie das nicht.

Ein Element ist ein konzeptioneller Teil eines Dokuments und hat einen festgelegten Anfang und ein festgelegtes Ende. Ein Tag hingegen markiert die Stelle, an der ein Element beginnt und endet.

Wenn in diesem Dokument vom „Tag `p`“ gesprochen wird, ist damit der Text gemeint, der aus den drei Zeichen `<`, `p` und `>` besteht. Wird

hingegen von dem „Element `p`“ gesprochen, ist damit das gesamte Element gemeint.

Diese Unterscheidung ist sicherlich subtil. Trotzdem sollte man sie sich vergegenwärtigen.

Elemente können selber Attribute haben, die aus einem Namen und einem Wert bestehen. Die Attribute haben die Aufgabe, dem Element zusätzliche Informationen hinzuzufügen. Denkbar sind hier Festlegungen über die Darstellung, Bezeichner, über die das Element eindeutig identifiziert werden kann, oder beliebige andere Informationen.

Elementattribute werden in den *Starttag* eingefügt und haben die Form `Attributename="Wert"` .

Bei einigen HTML-Versionen kennt das Element `p` das Attribut `align`, mit dessen Hilfe die Textausrichtung eines Absatzes bestimmt werden kann.

`align` akzeptiert einen von vier vorgegebenen Werten: `left`, `center`, `right` und `justify`. Ist `align` nicht angegeben, wird vom Standardwert `left` ausgegangen.

Beispiel 3.4. Elemente mit Attributen nutzen

```
<p align="left">Die Verwendung des align-Attributs
  für diesen Absatz ist überflüssig, da left
  der Standardwert ist.</p>

<p align="center">Dieser Absatz wird hoffentlich mittig ↺
dargestellt.</p>
```

Einige Attribute akzeptieren nur bestimmte Werte, wie beispielsweise `left` oder `justify`. Andere akzeptieren jeden beliebigen Wert. Enthält Attributwert doppelte Anführungszeichen (`"`), wird der Wert in einfachen Anführungszeichen eingeschlossen.

Beispiel 3.5. Attribute mit einfachen Anführungszeichen

```
<p align='right'>Ich stehe rechts!</p>
```

Manchmal können die Anführungszeichen um den Attributwert weggelassen werden. Allerdings sind die Regeln, die festlegen wann dies zulässig ist, sehr spitzfindig. Am besten schließen Sie Attributwerte *immer* in Anführungszeichen ein.

Die Informationen über Attribute, Elemente und Tags sind in SGML-Katalogen abgelegt und werden von den verschiedenen Werkzeugen des Dokumentationsprojektes genutzt, um die geschriebenen Dokumente zu validieren. Die Programme die durch textproc/docproj installiert werden, bringen ihre eigenen Katalogvarianten mit, zudem pflegt das FDP seine eigenen Kataloge. Beide Katalogarten müssen von allen Programmen gefunden werden können.

3.2.1. Was dafür getan werden muss;...

Damit die Beispiele dieser Fibel ausgeführt werden können, ist es notwendig, dass einige Programme auf dem Rechner installiert sind und das eine Umgebungsvariable korrekt gesetzt wird.

1. Der erste Schritt ist die Installation des Ports textproc/docproj über das FreeBSD-Portsystem. textproc/docproj ist ein *Metaport*, der alle vom FDP benötigten Programme und Daten aus dem Netz laden und installieren sollte.

2. Anschließend muss in den Shellkonfigurationsdateien die Variable `SGML_CATALOG_FILES` [4] gesetzt werden.

Beispiel 3.6. **`.profile`**, für sh(1) und bash(1) Benutzer

```
SGML_ROOT=/usr/local/share/xml
SGML_CATALOG_FILES=${SGML_ROOT}/jade/catalog
SGML_CATALOG_FILES=${SGML_ROOT}/docbook/4.1/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/html/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/iso8879/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=/usr/doc/share/xml/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=/usr/doc/en_US.ISO8859-1/share/xml/
catalog:$SGML_CATALOG_FILES
export SGML_CATALOG_FILES
```

[4] Sofern man nicht an der deutschen Dokumentation arbeitet, müssen die Verzeichnisangaben entsprechend angepasst werden.

Beispiel 3.7. `.cshrc`, für csh(1)- und tcsh(1)-Benutzer

```
setenv SGML_ROOT /usr/local/share/xml
setenv SGML_CATALOG_FILES ${SGML_ROOT}/jade/catalog
setenv SGML_CATALOG_FILES ${SGML_ROOT}/docbook/4.1/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/html/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/iso8879/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES /usr/doc/share/xml/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES /usr/doc/en_US.ISO8859-1/share/
xml/catalog:$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES /usr/doc/de_DE.ISO8859-1/share/
xml/catalog:$SGML_CATALOG_FILES
```

Damit die Änderungen wirksam werden, meldet man sich ab und anschließend wieder an – oder man führt die obigen Anweisungen direkt in der Shell aus und setzt so die benötigten Umgebungsvariablen.

1. Nun sollte man eine Datei `beispiel.xml` anlegen, die den folgenden Text enthält:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
  <head>
    <title>Eine Beispieldatei in HTML</title>
  </head>

  <body>
    <p>Das ist ein Absatz mit etwas Text.</p>

    <p>Das ist ein Absatz mit anderem Text.</p>

    <p align="right">Dieser Absatz wird rechtsbündig
      ausgerichtet.</p>
  </body>
</html>
```

2. Nachdem die Datei abgespeichert wurde, kann sie mit Hilfe eines SGML-Parsers validiert werden.

Bestandteil von textproc/docproj ist `onsgmls`- ein validierender Parser [10]. `onsgmls` liest ein Dokument entsprechend einer SGML-DTD ein und gibt anschließend

ein Element-Structure-Information-Set (ESIS) aus. Allerdings ist das an dieser Stelle nicht weiter wichtig.

Wird `onsgmls` mit der Option `-s` aufgerufen, werden nur Fehlermeldungen ausgegeben. Dadurch kann leicht geprüft werden, ob ein Dokument gültig ist oder nicht.

So prüft man mit `onsgmls`, ob die neuangelegte Beispieldatei gültig ist:

```
% onsgmls -s beispiel.xml
```

Sofern das Beispiel korrekt abgetippt wurde, wird sich `onsgmls` ohne jegliche Ausgabe beenden. Das bedeutet, dass das Dokument erfolgreich validiert werden konnte und somit gültig ist.

3. Jetzt sollten die Tags `title` und `/title` aus dem Dokument gelöscht und das Dokument erneut validiert werden:

```
% onsgmls -s beispiel.xml
onsgmls:beispiel.xml:5:4:E: character data is not allowed here
onsgmls:beispiel.xml:6:8:E: end tag for "HEAD" which is not ↺
finished
```

Die Fehlermeldungen, die von `onsgmls` ausgegeben werden, sind in durch Doppelpunkte getrennte Spalten unterteilt.

Spalte	Bedeutung
1	Der Name des Programms, das den Fehler meldet. Hier wird immer `onsgmls` stehen.
2	Der Name der fehlerhaften Datei.
3	Die Zeilennummer des Fehlers.
4	Die Spaltenummer des Fehlers.
5	Ein einbuchstabiger Code, der über die Art des Fehlers informiert. `I` steht für eine informelle Meldung, `W` für eine Warnung und `E` für Fehler[a] und `X` für einen Querverweis. Bei den oben stehenden Ausgaben handelt es sich also um Fehlermeldungen.
6	Die Meldung.

[a]Nicht immer besteht eine Meldung aus fünf Spalten. Die Ausgabe von `onsgmls -sv` ist beispielsweise `onsgmls:I: SP version "1.3"` (natürlich abhängig von der Version). Wie man sehen kann, handelt es sich hier um eine informelle Meldung.

Durch das Weglassen des Tags `title` sind zwei unterschiedliche Fehler entstanden.

Der erste Fehler besagt, dass Inhalt (in diesem Falle Zeichen anstatt eines Starttags) an einer Stelle gefunden wurde, an der der Parser etwas anderes erwartet hat. Genauer gesagt wurde der Starttag eines Elements erwartet, das innerhalb von `head` auftreten kann.

Der zweite Fehler wurde dadurch verursacht, dass das Element `head` ein Element `title` enthalten *muss* und `onsgmls` nicht berücksichtigt, dass dieser Fehler auf dem vorhergehenden beruht. Es wird lediglich festgestellt, dass der Endtag von `head` auftritt, obwohl nicht alle notwendigen Elemente vorhanden sind.

4. Zum Schluss sollte der Tag `title` wieder in die Beispieldatei eingefügt werden.

3.3. Die DOCTYPE-Deklaration

Am Anfang jedes Dokuments muss der Name der dem Dokument zugrundeliegenden DTD angegeben werden. Mit Hilfe dieser Information können SGML-Parser die verwendete DTD feststellen und prüfen, ob das Dokument zu ihr konform ist.

Üblicherweise steht diese Information in einer Zeile, die als DOCTYPE-Deklaration bezeichnet wird.

Eine Deklaration für ein HTML-Dokument, das nach den Vorgaben der DTD für HTML 4.0 geschrieben wurde, sieht so aus:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

und besteht aus verschiedenen Teilen.

`<!`
: Die Zeichenkette `<!` dient hier als *Indikator*, dass es sich bei diesem Ausdruck um eine SGML-Deklaration handelt und diese Zeile den Dokumententyp festlegt.

`DOCTYPE`
: Zeigt an, dass dies die SGML-Deklaration für den Dokumententyp ist.

`html`
: Nennt das erste Element, das im Dokument auftaucht.

`PUBLIC "-//W3C//DTD HTML 4.0//EN"`
: Nennt den Formalen Öffentlichen Bezeichner [5] der DTD des Dokuments. Diese Information wird von SGML-Parsern ausgewertet, um die von dem Dokument referenzierte DTD zu bestimmen.

[5] auf Englisch *Formal Public Identifier (FPI)*

Das Schlüsselwort `PUBLIC` gehört nicht zum öffentlichen Bezeichner, sondern legt fest, wie ein SGML-Parser die DTD finden kann. Alternative Wege eine DTD zu referenzieren werden später gezeigt.

`>`

Schließt den mit `<!` begonnenen Ausdruck ab.

3.3.1. Formale Öffentliche Bezeichner

Anmerkung

Dieser Abschnitt braucht nicht unbedingt zu gelesen zu werden. Dennoch ist es empfehlenswert, da er nützliche Hintergrundinformationen enthält, die hilfreich sein können, falls der SGML-Prozessor die genutzte DTD nicht finden kann.

Jeder öffentliche Bezeichner muss eine bestimmte Syntax haben, die wie folgt lautet:

```
"Besitzer //Schlüsselwort  Beschreibung //Sprache"
```

`Besitzer`

Nennt den Besitzer des öffentlichen Bezeichners.

Falls diese Zeichenkette mit „ISO“ beginnt, gehört der Bezeichner dem ISO-Komitee. Der Bezeichner `"ISO 8879:1986//ENTITIES Greek Symbols//EN"` nennt „ISO 8879:1986“ als den Besitzer des Satzes von Entitäten für griechische Zeichen. ISO 8879:1986 ist die ISO-Bezeichnung für den SGML-Standard.

Beginnt die Zeichenkette nicht mit „ISO“, sieht sie entweder so `-//Besitzer` oder so `+//Besitzer` aus. Beide Varianten unterscheiden sich also nur durch das anfängliche `+` bzw. `-`.

Sofern am Anfang ein `-` steht, ist der Bezeichner nicht öffentlich registriert, steht hingegen ein `+` am Anfang, ist er registriert.

Im ISO-Standard ISO 9070:1991 wurde festgelegt, wie registrierte Namen erzeugt werden können. Unter anderem können sie von den Bezeichnungen von ISO-Publikationen, von ISBN-Nummern oder einer Organisationsbezeichnungen entsprechend ISO 6523 abgeleitet werden. Anträge für neue offiziell registrierte Bezeichner werden vom ISO-Komitee an das American National Standards Institute (ANSI) weitergeleitet.

Da das FreeBSD-Projekt seine Bezeichner nicht hat registrieren lassen, ist der Besitzer `-//FreeBSD`. Unter anderem kann man daran auch sehen, dass das W3C sich nicht hat registrieren lassen.

Schlüsselwort
: Es gibt verschiedene Schlüsselwörter mit denen man die Art der gegebenen Informationen beschreiben kann. Einige der üblichsten sind `DTD`, `ELEMENT`, `ENTITIES` und `TEXT`. `DTD` wird nur für Dateien mit DTDs verwandt, `ELEMENT` findet für Dateien mit Fragmenten von DTDs Verwendung, die nur Deklarationen für Entitäten und Elemente enthalten. `TEXT` wird für SGML-Inhalte (Texte und Tags) verwendet.

Beschreibung
: Eine frei wählbare Beschreibung des Inhalts der referenzierten Datei. Möglich sind hier Versionsnummern oder ein kurzer und sinnvoller Text, der innerhalb der SGML-Welt eindeutig ist.

Sprache
: Ein ISO-Code aus zwei Buchstaben, der die für die Datei verwendete Sprache nennt. `EN` steht hier für Englisch, `DE` für Deutsch.

3.3.1.1. Die `catalog`-Dateien

Wenn man die oben beschriebene Syntax für Bezeichner verwendet und ein Dokument durch einen SGML-Prozessor schickt, muss dieser die Möglichkeit haben, den Bezeichner auf eine real existierende Datei abzubilden, die die benötigte DTD enthält.

Einer der möglichen Wege hierfür sind Katalogdateien. Eine solche Datei, die üblicherweise `catalog` heißt, besteht aus einzelnen Zeilen, die Bezeichner auf Dateinamen abbilden. Enthält ein Katalog beispielsweise die Zeile

```
PUBLIC "-//W3C//DTD HTML 4.0//EN"              "4.0/strict.dtd"
```

kann ein SGML-Prozessor darüber feststellen, dass die benötigte DTD in der Datei `strict.dtd` im Unterverzeichnis `4.0` des Verzeichnisses des Katalogs zu finden ist.

Ein gutes Beispiel für einen Katalog ist `/usr/local/share/xml/html/catalog` . Diese Datei enthält den Katalog für alle HTML DTDs, die im Zuge der Installation von textproc/docproj installiert wurden.

3.3.1.2. Die Variable `SGML_CATALOG_FILES`

Natürlich muss einem SGML-Prozessor noch mitgeteilt werden können, wo er seine Kataloge finden kann. Viele Programme bieten hierfür Kommandozeilenoptionen an, über die man einen oder mehrere Kataloge angeben kann.

Zusätzlich besteht noch die Möglichkeit mit der Umgebungsvariablen `SGML_CATALOG_FILES` auf SGML-Kataloge zu verweisen. Die Einträge von `SGML_CATALOG_FILES` müssen aus den vollständigen Pfadnamen der Kataloge, jeweils durch Komma getrennt, bestehen.

Üblicherweise werden die folgenden Kataloge über `SGML_CATALOG_FILES` für die Arbeit an den Dokumenten des FDPs eingebunden:

- `/usr/local/share/xml/docbook/4.1/catalog`
- `/usr/local/share/xml/html/catalog`
- `/usr/local/share/xml/iso8879/catalog`
- `/usr/local/share/xml/jade/catalog`

Allerdings sollte das schon geschehen sein.

3.3.2. Alternativen zu Formalen Öffentlichen Bezeichnern

Anstatt mit einem Bezeichner die zum Dokument gehörende DTD zu referenzieren, kann auch explizit auf die Datei der DTD verwiesen werden.

Die Syntax der DOCTYPE-Deklaration ist in diesem Falle anders:

```
<!DOCTYPE html SYSTEM "/pfad/zur/dokumenten.dtd">
```

Das Schlüsselwort `SYSTEM` legt fest, dass ein SGML-Prozessor die DTD auf „systemspezifische“ Art und Weise bestimmen soll. Meistens, aber nicht immer, wird so auf eine Datei im Dateisystem verwiesen.

Allerdings sollte man öffentliche Bezeichner aus Gründen der Portabilität bevorzugen, da man so nicht eine Kopie der DTD mit dem Dokument selber verteilen muss, beziehungsweise da man, wenn man mit `SYSTEM` arbeitet, nicht davon ausgehen kann, dass die benötigte DTD auf anderen Systemen genau unter dem gleichen Pfad verfügbar ist.

3.4. Die Rückkehr zu SGML

An einer früheren Stelle wurde erwähnt, dass man SGML nur benötigt, falls man selbst eine DTD entwickeln möchte. Genaugenommen ist das nicht 100%ig richtig. Einige Teile der SGML-Syntax können auch in normalen Dokumenten verwendet werden, falls dies gewünscht oder notwendig ist.

In diesem Falle muss dafür Sorge getragen werden, dass ein SGML-Prozessor feststellen kann, dass ein bestimmter Abschnitt des Inhalts SGML ist, das er verarbeiteten muss.

Solche SGML-Abschnitte werden mittels `<! ... >` in Dokumenten besonders gekennzeichnet. Alles, was sich zwischen diesen Begrenzungen befindet, ist SGML, wie es auch in DTDs gefunden werden kann.

Demnach ist die DOCTYPE-Deklaration ein gutes Beispiel für SGML, das in Dokumenten verwendet werden muss...

3.5. Kommentare

Kommentare sind SGML-Konstrukte, die normalerweise nur in DTDs gültig sind. Dennoch ist es, wie in Abschnitt 3.4, „Die Rückkehr zu SGML“ gezeigt, möglich Fragmente mit SGML-Syntax in Dokumenten zu verwenden.

Zum Abgrenzen von SGML-Kommentaren wird ein doppelter Bindestrich „`--`“ verwendet. Mit seinem ersten Auftreten öffnet er einen Kommentar, mit seinem zweiten schließt er ihn wieder.

Beispiel 3.8. Beispiele für Kommentare in SGML

```
<!-- Testkommentar -->
```

```
<!-- Text innerhalb eines Kommentars -->

6lt;!-- Ein anderer Kommentar    -->

6lt;!-- So können mehrzeilige Kommentare
     genutzt werden -->

<!-- Eine andere Möglichkeit für --
  -- mehrzeilige Kommentare.     -->
```

Hat man früher schon Erfahrungen mit HTML gesammelt, wird man vielleicht andere Regeln für den Gebrauch von Kommentaren kennengelernt haben. Beispielsweise wird oft angenommen, dass Kommentare mit `<!--` begonnen und nur mit `-->` beendet werden.

Dies ist *nicht* der Fall. Viele Webbrowser haben fehlerhafte HTML-Parser, die dies akzeptieren. Die SGML-Parser, die vom FDP verwendet werden, halten sich strenger an die SGML-Spezifikation und verwerfen Dokumente mit solchen Fehlern.

Beispiel 3.9. Fehlerhafte SGML-Kommentare

```
<!-- Innerhalb eines Kommentars --

     DIES IST NICHT TEIL EINES KOMMENTARS

  -- Wieder innerhalb eines Kommentars -->
```

SGML-Parser würden die mittlere Zeile wie folgt interpretieren:

```
<!DIES IST NICHT TEIL EINES KOMMENTARS>
```

Da es sich hierbei nicht um gültiges SGML handelt, kann diese Zeile zur verwirrenden Fehlermeldungen führen.

```
<!----- Eine wirklich schlechte Idee  ----->
```

Wie das Beispiel zeigt, sollten solche Kommentare tunlichst vermieden werden.

```
<!--================================================-->
```

Ein solcher Kommentar ist (ein wenig) besser, kann aber jemanden, der mit SGML noch nicht so vertraut ist, verwirren.

3.5.1. Fingerübungen...

1. Zur Übung können Sie einige Kommentare in die Datei `beispiel.xml` einfügen und überprüfen, ob die Datei nun noch erfolgreich von `onsgmls` validiert werden kann.

2. Zu Testzwecken sollten Sie auch noch ein paar fehlerhafte Kommentare hinzufügen und sich die resultierenden Fehlermeldungen von `onsgmls` ansehen.

3.6. Entitäten

Entitäten stellen einen Mechanismus dar, mit dem einzelnen Dokumententeilen ein Name zugewiesen werden kann. Findet ein SGML-Parser während des Parsens eine Entität, ersetzt er diese durch den ihr zugewiesenen Inhalt.

Dadurch steht eine einfache Möglichkeit zur Verfügung, mit der variabler Inhalt in SGML-Dokumenten eingebunden werden kann. Zusätzlich sind Entitäten der einzige Weg, über den eine Datei in eine andere Datei mit SGML-Mitteln eingebunden werden kann.

Es werden zwei Arten von Entitäten unterschieden: *Allgemeine Entitäten* und *Parameterentitäten*.

3.6.1. Allgemeine Entitäten

Allgemeine Entitäten können nur in Dokumenten benutzt werden. Sie können zwar im SGML-Kontext definiert aber dort nicht benutzt werden. Vergleichen Sie dies mit Im Parameterentitäten.

Jede allgemeine Entität hat einen Namen, über den sie angesprochen werden kann, um den von ihr referenzierten Inhalt in ein Dokument einzubinden. Dafür muss an der betref-

fenden Stelle der Namen der Entität per `&entitaetenname;` einfügt werden. Eine Entität `current.version` könnte beispielsweise durch die aktuelle Versionsnummer eines Programms ersetzt werden. Man könnte also schreiben:

```
<para>Die aktuelle Version des
  Programms ist &current.version;.</para>
```

Wenn sich die Versionsnummer ändert, muss nur die Entität angepasst und anschließend das Dokument neu erzeugt werden.

Eine weitere Einsatzmöglichkeit für Allgemeine Entitäten ist das Einbinden von Zeichen, die auf andere Weise nicht in ein SGML-Dokument eingefügt werden könnten. Ein Beispiel für solche Zeichen sind < und &, die normalerweise nicht direkt in SGML-Dokumenten erlaubt sind. Stößt ein SGML-Parser bei seiner Arbeit auf das Symbol <, nimmt er an, dass der Anfang eines Start- oder Endtags gefunden wurde. Bei einem & wird er annehmen, den Anfang einer Entität gefunden zu haben.

Wenn eines der beiden Zeichen benötigt wird, werden daher die allgemeinen Entitäten `<` und `&` verwendet.

Allgemeine Entitäten können nur in einem SGML-Kontext definiert werden. Üblich ist es, dies direkt nach der DOCTYPE-Deklaration zu tun:

Beispiel 3.10. Allgemeine Entitäten festlegen

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY current.version    "3.0-RELEASE">
<!ENTITY last.version       "2.2.7-RELEASE">
]>
```

Wichtig ist an dieser Stelle, dass die DOCTYPE-Deklaration durch eine eckige Klammer am Ende der ersten Zeile erweitert wurde. Die beiden Entitäten selber werden in den folgenden zwei Zeilen definiert, bevor in der letzten Zeile die eckige Klammer und die DOCTYPE-Deklaration wieder geschlossen werden.

Die eckigen Klammern sind notwendig um festzulegen, dass man die über DOCTYPE genannte DTD erweitern möchte.

3.6.2. Parameterentitäten

Genau wie Allgemeine Entitäten werden Parameterentitäten eingesetzt um wiederverwendbare Inhaltsteile mit Namen zu versehen. Im Gegensatz zu Allgemeinen Entitäten können sie aber nur innerhalb eines SGML-Kontextes genutzt werden.

Die Definition von Parameterentitäten erfolgt ähnlich zu der Allgemeiner Entitäten. Sie werden lediglich mit `%entitaetenname;` anstelle von `&entitaetenname;` referenziert[6]. Wichtig ist, dass das `%`-Zeichen zwischen `ENTITY` und dem Entitätennamen ein Teil der Definition ist.

Beispiel 3.11. Parameterentitäten festlegen

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % param.etwas "etwas">
<!ENTITY % param.text "Text">
<!ENTITY % param.neu  "%param.etwas mehr %param.text">
]>
```

3.6.3. Fingerübungen...

1. Fügen Sie in `beispiel.xml` eine Allgemeine Entität ein.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" [
<!ENTITY version "1.1">
]>

<html>
  <head>
    <title>Eine HTML-Beispieldatei</title>
  </head>

  <body>
    <p>Das ist ein Absatz mit etwas Text.</p>

    <p>Das ist ein Absatz mit anderem Text.</p>

    <p align="right">Dieser Absatz wird rechtsbündig
      ausgerichtet.</p>

    <p>Die aktuelle Version ist: &amp;version;</p>
  </body>
</html>
```

2. Validieren Sie diese Datei mit `onsgmls`

3. Öffnen Sie nun `beispiel.xml` mit Ihrem Webbrowser. Es kann notwendig sein, dass Sie die Datei vorher in `beispiel.html` umbenennen müssen, damit die Datei auch als HTML-Dokument erkannt wird.

[6] Es wird das Prozentzeichen anstelle des kaufmännischen Unds verwendet.

Nur wenn Sie einen sehr modernen Browser haben, werden Sie sehen können, dass `&version;` durch die Versionsnummer ersetzt wurde. Leider haben die meisten Webbrowser sehr einfache SGML-Parser, die nicht richtig mit SGML umgehen können[7].

4. Die Lösung hierfür ist, das Dokument zu *normalisieren*. Zu diesem Zweck liest ein Normer das Dokument ein und gibt anschließend semantisch gleichwertiges SGML wieder aus, dass auf verschiedene Arten transformiert worden sein kann. Eine dieser möglichen Transformationen ist das Ersetzen der Referenzen auf Entitäten mit dem von ihnen präsentierten Inhalt.

   Versuchen Sie, die Beispieldatei mittels `osgmlnorm` zu normalisieren:

   ```
   % osgmlnorm beispiel.xml > beispiel.html
   ```

   Anschließend sollten Sie eine normalisierte Version, dass heißt eine, bei der die Entitäten gegen ihren Inhalt ersetzt wurden, in der Datei `beispiel.html` finden. Diese Datei können Sie sich nun mit Ihrem Browser ansehen.

5. Wenn Sie sich die Ausgaben von `osgmlnorm` ansehen, werden Sie feststellen, dass die DOCTYPE-Deklaration am Anfang der Datei nicht mehr enthalten ist. Möchten Sie die Deklaration behalten, muss `osgmlnorm` mit der Option `-d` aufrufen werden:

   ```
   % osgmlnorm -d beispiel.xml > beispiel.html
   ```

3.7. Dateien mit Entitäten einbinden

Sowohl Allgemeine als auch Parameterentitäten sind nützliche Helfer, wenn es darum geht, eine Datei in eine andere einzubinden.

3.7.1. Dateien mit Allgemeinen Entitäten einbinden

Angenommen man hat ein Buch geschrieben, dessen Inhalt auf mehrere Dateien aufgeteilt und mittels SGML ausgezeichnet. Jedes Kapitel wurde dazu in einer eigenen Datei (`kapitel1.xml` , `kapitel2.xml` usw.) abgelegt und über eine Datei `buch.xml` sollen alle physischen Dateien wieder mit der Hilfe von Entitäten zu einem logischen Dokument zusammengeführt werden.

Damit der Inhalt der Dateien mit Entitäten eingebunden werden kann, muss die Deklaration der Entitäten das Schlüsselwort `SYSTEM` enthalten. SGML-Parser werden so angewiesen, den Inhalt der referenzierten Datei als Wert für die jeweilige Entität zu nehmen.

[7] Eigentlich ist das eine Schande. Man stelle sich vor, welche Probleme und Hacks, wie beispielsweise Server Side Includes, man an dieser Stelle hätte vermeiden können.

Beispiel 3.12. Dateien mit Allgemeinen Entitäten einbinden

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY kapitel.1 SYSTEM "kapitel1.xml">
<!ENTITY kapitel.2 SYSTEM "kapitel2.xml">
<!ENTITY kapitel.3 SYSTEM "kapitel3.xml">
]>

<html>

  &amp;kapitel.1;
  &amp;kapitel.2;
  &amp;kapitel.3;
</html>
```

Warnung

Wenn man Allgemeine Entitäten benutzt, um andere Dateien einzubinden, dürfen diese Dateien (`kapitel1.xml` , `kapitel2.xml` , ...) *keine* eigene DOCTYPE-Deklaration haben.

3.7.2. Dateien mit Parameterentitäten einbinden

Wie bereits festgestellt, können Parameterentitäten nur innerhalb eines SGML-Kontexts genutzt werden. Warum möchte man aber Dateien innerhalb eines SGML-Kontexts einbinden? Der Vorteil liegt in der Möglichkeit, die Deklaration von Entitäten in eine andere Datei auslagern zu können, wodurch diese leichter wiederverwendbar sind.

Angenommen das Buch aus dem vorherigen Kapitel besteht aus sehr vielen Kapiteln und diese sollen auch in einem anderen Buch, aber in einer anderen Reihenfolge, verwendet werden. Eine Möglichkeit wäre es, die dafür notwendigen Entitäten am Anfang jedes Buches einzeln festzulegen – was allerdings mit der Zeit unhandlich und fehlerträchtig wird.

Alternativ bietet sich dazu an, die Deklarationen in eine separate Datei auszulagern und deren Inhalt anschließend in beide Bücher über Parameterentitäten einzubinden.

Beispiel 3.13. Dateien mit Parameterentitäten einbinden

Zuerst werden die Entitäten in einer separaten Datei namens `kapitel.ent` deklariert. `kapitel.ent` enthält für dieses Beispiel die folgenden Zeilen:

```
<!ENTITY kapitel.1 SYSTEM "kapitel1.xml">
<!ENTITY kapitel.2 SYSTEM "kapitel2.xml">
<!ENTITY kapitel.3 SYSTEM "kapitel3.xml">
```

Im zweiten Schritt fügt man in beide Bücher eine Parameterentität ein, die den Inhalt von `kapitel.ent` referenziert, und lädt über diese dann die Deklarationen. Anschließend können die so geladenen Entitäten wie gewohnt genutzt werden.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % kapitel SYSTEM "kapitel.ent">
%kapitel;
]>

<html>
  &amp;kapitel.1;
  &amp;kapitel.2;
  &amp;kapitel.3;
</html>
```

3.7.3. Fingerübungen...

3.7.3.1. Binden Sie Dateien über Allgemeine Entitäten ein

1. Legen Sie drei Dateien (`absatz1.xml` , `absatz2.xml` und `absatz3.xml`) mit jeweils einer Zeile wie

   ```
   <p>Erster Absatz.</p>
   ```

   an.

2. Ändern Sie `beispiel.xml` so ab, dass sie wie folgt aussieht:

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
   <!ENTITY version "1.1">
   <!ENTITY absatz1 SYSTEM "absatz1.xml">
   <!ENTITY absatz2 SYSTEM "absatz2.xml">
   <!ENTITY absatz3 SYSTEM "absatz3.xml">
   ]>
   ```

```
<html>
  <head>
    <title>Eine HTML-Beispieldatei</title>
  </head>

  <body>
    <p>Die aktuelle Version dieses Dokuments ist &version;</p>

    &absatz1;
    &absatz2;
    &absatz3;
  </body>
</html>
```

3. Erzeugen Sie nun die Datei `beispiel.html` , indem Sie `beispiel.xml` normalisieren:

```
% osgmlnorm -d beispiel.xml > beispiel.html
```

4. Öffnen Sie `beispiel.html` nun mit einem Webbrowser und vergewissern Sie sich, dass der Inhalt der Dateien `absatzN.xml` in `beispiel.html` übernommen wurde.

3.7.3.2. Binden Sie Dateien mit Parameterentitäten ein

Anmerkung

Hierfür müssen Sie die vorherige Fingerübung gemacht haben.

1. Ändern Sie `beispiel.xml` so ab, dass es wie folgt aussieht:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % entitaeten SYSTEM "entitaeten.xml"> %entitaeten;
]>

<html>
  <head>
    <title>Eine HTML-Beispieldatei</title>
  </head>

  <body>
    <p>Die aktuelle Version dieses Dokuments ist &version;</p>

    &absatz1;
    &absatz2;
    &absatz3;
  </body>
</html>
```

2. Legen Sie eine weitere Datei `entitaeten.xml` an, die folgenden Inhalt hat:

```
<!ENTITY version "1.1">
<!ENTITY absatz1 SYSTEM "absatz1.xml">
<!ENTITY absatz2 SYSTEM "absatz2.xml">
<!ENTITY absatz3 SYSTEM "absatz3.xml">
```

3. Erzeugen Sie die Datei `beispiel.html` , indem Sie `beispiel.xml` normalisieren:

```
% osgmlnorm -d beispiel.xml > beispiel.html
```

4. Öffnen Sie `beispiel.html` nun mit einem Webbrowser und vergewissern Sie sich, dass der Inhalt der Dateien `absatzN.xml` in `beispiel.html` übernommen wurde.

3.8. Markierte Bereiche

SGML erlaubt es, dass bestimmte Dokumentabschnitte während der Verarbeitung besonders behandelt werden sollen. Diese Abschnitte werden als „markierte Bereiche“ [8] bezeichnet.

Beispiel 3.14. Aufbau eines markierten Bereiches

```
<![ SCHLÜSSELWORT  [
  Inhalt des markierten Bereiches
]]>
```

Da es sich bei markierten Bereichen um SGML-Konstrukte handelt, werden sie mit `<!` eingeleitet. Der eigentliche Anfang des markierten Bereiches wird von der folgenden eckigen Klammer bestimmt. Das darauf folgende *`SCHLÜSSELWORT`* legt fest, wie der „markierte Inhalt“ durch einen SGML-Prozessor während der Verarbeitung behandelt werden soll. Der „markierte“ Inhalt selbst beginnt erst nach der zweiten eckigen Klammer und erstreckt sich bis zu den zwei schließenden eckigen Klammern am Ende des Bereiches. Mit Hilfe des `>` Zeichens wird der mit `<!` begonnene SGML-Kontext wieder verlassen.

3.8.1. Schlüsselworte für markierte Bereiche

3.8.1.1. `CDATA` und `RCDATA`

Die Schlüsselworte `CDATA` und `RCDATA` bestimmen das *Inhaltsmodell* für markierte Bereiche. Dadurch ist es möglich, vom Standardmodell abzuweichen.

[8] auf Englisch *marked sections*

Ein SGML-Prozessor muss während der Verarbeitung eines Dokuments zu jedem Zeitpunkt wissen, welches *Inhaltsmodell* gerade anzuwenden ist.

Was ist ein Inhaltsmodell? Kurz gesagt beschreibt das Inhaltsmodell, welche Art von Inhalt der Parser zu erwarten und wie er damit umzugehen hat.

Bei `CDATA` und `RCDATA` handelt es sich wahrscheinlich um die nützlichsten Inhaltsmodelle. `CDATA` steht für Zeichendaten[9]. Trifft ein Parser auf dieses Inhaltsmodell, wird er annehmen, dass sich im zugehörigen Dokumentenbereich nur „gewöhnliche" Zeichen befinden. Das bedeutet, dass < und & ihre besondere Bedeutung verlieren und als einfache Zeichen behandelt werden.

`RCDATA` steht für Entitätenreferenzen und Zeichendaten[10]. Für einen Bereich mit diesem Inhaltsmodell, wird ein Parser davon ausgehen, dass er sowohl Zeichen als auch Enitätenreferenzen finden kann. < verliert hier zwar auch seine besondere Bedeutung, doch & wird weiterhin als Anfang einer Entität interpretiert.

Nützlich ist das `CDATA`-Modell vor allem dann, wenn es darum geht Texte eins-zu-eins zu übernehmen, in denen < und & gehäuft auftreten. Zwar kann man solche Texte überarbeiten und jedes < durch ein `<` und jedes & durch ein `&` ersetzen, doch es wird in den meisten Fällen einfacher sein, für den betreffenden Text `CDATA` als Inhaltsmodell festzulegen. Ein SGML-Parser wird dann, sobald er auf < oder & trifft, diese als Zeichen in einem Text betrachten.

Anmerkung

Bei der Verwendung von `CDATA` und `RCDATA` als Inhaltsmodell für SGML-Beispiele, wie sie in diesem Dokument enthalten sind, muss bedacht werden, dass der Inhalt eines `CDATA`-Bereiches nicht validiert wird. dass das SGML in diesen Bereichen gültig ist, muss auf andere Weise sichergestellt werden. Denkbar ist beispielsweise, es in einem separaten Dokument zu erstellen, dort zu prüfen und erst dann in das eigentliche Dokument einzufügen.

Beispiel 3.15. CDATA als Inhaltsmodell für markierte Bereiche

```
<para>Das ist ein Beispiel, wie man einen Text,
```

[9] auf Englisch *character data*

[10] auf Englisch *Entity references and character data*

```
  der viele &lt;- und &amp;-
  Entitäten enthält, in ein Dokument einbinden kann.
  Das Beispiel selbst, das sich innerhalb des markierten ↺
Bereiches befindet,
  ist ein HTML-Fragment. Der diesen Text umschließende Tag, ↺
beginnend mit
  mit para und endend mit /para, stammt aus der DocBook DTD.</
para>

<programlisting>
  <![RCDATA[
    <p>Dieses Beispiel demonstriert die Verwendung von HTML-
Elementen.
      Da spitze Klammern so oft vorkommen, ist es einfacher, ↺
das
      gesamte Beispiel als CDATA Abschnitt auszuweisen, als die
      entsprechenden Entitäten zu nutzen.</p>

    <ul>
      <li>Das ist ein Listenelement.</li>
      <li>Das ist ein zweites Listenelement.</li>
      <li>Das ist ein drittes Listenelement.</li>
    </ul>

    <p>Und das hier, das ist das Ende des Beispiels.</p>
  -]]>
</programlisting>
```

Liest man die Quellen dieser Fibel, wird man feststellen, dass diese Technik durchgängig angewandt wurde.

3.8.1.2. `INCLUDE` und `IGNORE`

Das Schlüsselwort `INCLUDE` legt fest, dass der Inhalt des betreffenden Abschnittes mitverarbeitet wird. Demgegenüber bestimmt `IGNORE`, dass er ignoriert wird, dass heißt, dass er bei der Verarbeitung übergangen wird und in der Ausgabe nicht enthalten ist.

Beispiel 3.16. Anwendung von `INCLUDE` und `IGNORE` in markierten Abschnitten

```
<![ INCLUDE [
  Dieser Text wird verarbeitet und eingebunden.
]]>

<![ IGNORE [
  Dieser Text wird weder verarbeitet noch eingebunden.
```

```
]]>
```

Für sich alleine ist `IGNORE` als Anweisung nicht besonders nützlich, da ein Bereich, der von der Verarbeitung ausgenommen sein soll, auch auskommentiert werden kann.

Kombiniert man `IGNORE` hingegen mit Parameterentitäten, steht so ein Weg zur Verfügung, um dessen Anwendung besser steuern zu können. Zwar können Parameterentitäten nur in einem SGML-Kontext einsetzt werden, da aber markierte Bereiche ebenfalls SGML-Konstrukte sind, ist diese Einschränkung irrelevant.

Soll beispielsweise ein und dasselbe Dokument in zwei unterschiedlichen Varianten produziert werden, einer gedruckten und einer digitalen, und soll nur die digitale zusätzliche Informationen enthalten, kann dies mit einem Trick erreicht werden.

Man definiert eine Parameterentität, der man als Wert die Zeichenkette `INCLUDE` zuweist und deklariert den betreffenden Bereich, der nur in der digitalen Variante erscheinen soll, als markierten Abschnitt und setzt als Schlüsselwort die zuvor definierte Parameterentität ein.

Soll anstelle der digitalen die gedruckte Variante produziert werden, muss lediglich der Entität `IGNORE` als Wert zugewiesen und das Ursprungsdokument erneut durch den SGML-Prozessor geschickt werden.

Beispiel 3.17. Kontrolle von markierten Bereichen über Parameterentitäten

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % digitale.kopie "INCLUDE">
]]>

...

<![ %digitale.kopie [
  Dieser Satz sollte nur in der digitalen Version enthalten ↺
sein.
]]>
```

Bei der Produktion der gedruckten Variante muss der Wert der Entität geändert werden.

```
<!ENTITY % digitale.kopie "IGNORE">
```

Bei der Verarbeitung wird als Schlüsselwort in beiden Fällen der von `%digitale.kopie` repräsentierte Wert verwendet. Im ersten Fall wird der Inhalt des markierten Bereichs mitverarbeitet, im zweiten Fall nicht.

3.8.2. Fingerübung...

1. Legen Sie eine neue Datei `abschnitt.xml` an, die folgenden Inhalt hat:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % text.ausgabe "INCLUDE">
]>

<html>
  <head>
    <title>Ein Beispiel mit markierten Abschnitten</title>
  </head>

  <body>
    <p>Dieser Absatz <![CDATA[beinhaltet viele <
      Zeichen (< < < < <). Weshalb es einfacher ist,
      ihn als CDATA Bereich auszuweisen. -]]></p>

    <![ IGNORE [
    <p>Dieser Absatz wird NICHT in der Ausgabe enthalten sein.</p>
    -]]>

    <![ %text.ausgabe [
     <p>Dieser Absatz wird in Abhängigkeit von %text.ausgabe
       mitausgegeben.</p>
    -]]>
  </body>
</html>
```

2. Normalisieren Sie den Inhalt dieser Datei mit Hilfe von `osgmlnorm` und sehen Sie sich das Ergebnis an. Achten Sie dabei darauf, welche Absätze enthalten beziehungsweise nicht enthalten sind und was aus den `CDATA`-Bereichen geworden ist.

3. Ändern Sie die Definition von `text.ausgabe` so, dass es den Wert `IGNORE` zugewiesen bekommt. Verarbeiten Sie dann die Datei erneut mit `osgmlnorm` und vergleichen die Ausgabe mit der vom ersten `osgmlnorm` Lauf.

3.9. Schlussbemerkung

Aus Platzgründen, und um der Verständlichkeit Willen, wurden viele Gesichtspunkte nicht in aller Tiefe beziehungsweise gar nicht besprochen. Trotzdem sollte in den bisherigen Kapiteln genügend Wissen über SGML vermittelt worden sein, um den Aufbau der Dokumentation des FDPs zu verstehen.

Kapitel 4. XML-Dokumente erstellen

In diesem Kapitel werden die beiden vom FDP eingesetzten Auszeichnungssprachen XHTML und DocBook behandelt. Hierbei beschränkt sich dieses Kapitel auf die Elemente, die bei der täglichen Arbeit am ehesten zum Einsatz kommen werden.

Beide Sprachen besitzen eine große Anzahl von Elementen. Das erschwert es, das richtige Element in der richtigen Situation auszuwählen. Aus diesem Grund werden zu jedem Element auch immer Beispiele angeboten, die den richtigen Einsatz des Elements verdeutlichen sollen.

Es ist nicht das Ziel dieses Kapitels möglichst viele Elemente beider Sprachen zu behandeln – dies wäre nur eine Wiederholung der eigentlichen Sprachreferenz. Sofern es Unklarheiten zur Verwendung einzelner Elemente und Auszeichnung von bestimmten Sachverhalten gibt, können diese an FreeBSD documentation project geschickt werden.

Fluß- kontra Blockelemente

Wenn im Folgenden von *Flußelementen* die Rede ist, sind damit Elemente gemeint, die in einem Blockelement auftreten können und keinen Zeilenumbruch hervorrufen. *Blockelemente* hingegen erzeugen unter anderem einen Zeilenumbruch[1].

4.1. XHTML

XHTML ist die XML Version der HyperText Markup Language, der Auszeichnungssprache der Wahl im World Wide Web. Weitere Informationen zu XHTML finden sich unter `http://www.w3.org/` .

XHTML kommt bei der Erstellung der Webseiten des FreeBSD-Projektes zum Einsatz. Für technische Dokumentationen sollte XHTML jedoch nicht eingesetzt werden, da DocBook eine größere und bessere Auswahl an Elementen bietet. Folglich sollte XHTML nur für die FreeBSD-Webseiten verwendet werden.

[1] Die englische Bezeichnung *inline element* wurde in Anlehnung an das Wort „Fließtext“ mit „Flußelement“ übersetzt.

Die HTML-Spezifikation liegt bis jetzt in mehreren Versionen vor: 1, 2, 3.0, 3.2, 4.0 sowie eine XML konforme Version namens XHTML in seiner neuesten Version XHTML 1.0 (verfügbar als *strict* und *transitional* Variante.

Die XHTML-DTDs sind über den Port textproc/xhtml verfügbar und werden automatisch als Teil des Metaports textproc/docproj installiert.

4.1.1. Formale Öffentliche Bezeichner

Da es mehrere Version von XHTML gibt, existieren auch mehrere FÖPs, zu denen ein XHTML-Dokument konform erklärt werden kann. Die Mehrzahl der sich auf der FreeBSD-Webseite befindenden XHTML-Seiten sind zu der lockeren Version von XHTML 1.0 konform.

```
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

4.1.2. Die Elemente `head` und `body`

Ein XHTML-Dokument unterteilt sich normalerweise in zwei Bereiche: „head“ und „body“. Der Kopf (*head*) enthält Metadaten wie den Dokumententitel und Angaben zum Autor. Der Rumpf (*body*) umfasst den eigentlichen Dokumenteninhalt, der für den Leser bestimmt ist. In einem XHTML-Dokument werden diese Bereiche über die Elemente `head` und `body` voneinander abgegrenzt. Beide sind Kinder des Wurzelelementes `html`.

Beispiel 4.1. Die Struktur eines XHTML-Dokumentes

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Der Dokumententitel </title>
  </head>

  <body>

    …

  </body>
</html>
```

4.1.3. Blockelemente

4.1.3.1. Überschriften

XHTML kennt sechs verschiedene Elemente, mit denen Überschriften ausgezeichnet werden können. Das bekannteste Element ist `h1`, das sich am Anfang der Überschriftenhier-

archie befindet. `h1` folgen die Überschriftenelemente `h2` bis `h6`. Der Inhalt von `hN` stellt den Text der Überschrift dar.

Beispiel 4.2. `h1`, `h2`...

Fügen Sie in eine der existierenden Übungsdateien folgendes ein:

```
<h1>Erstes Kapitel</h1>

<!-- Hier steht die Einführung -->

<h2>Das ist die Überschrift des ersten Kapitels</h2>

<!-- Hier steht der Inhalt des ersten Kapitels -->

<h3>Das ist die Überschrift des ersten Unterkapitels</h3>

<!-- Hier steht der Inhalt des ersten Unterkapitels -->

<h2>Das ist die Überschrift des zweiten Kapitels</h2>

<!-- Hier steht der Inhalt des zweiten Kapitels -->
```

Eine XHTML-Seite sollte immer nur eine Überschrift `h1` haben. Dieser Überschrift können beliebig viele Kapitel mit einer Überschrift `h2` folgen, die selbst wiederum eine beliebige Anzahl von Kapiteln mit einer Überschrift `h3` enthalten können. Diese Verschachtelung setzt sich bis zu Kapiteln mit einer `h6`-Überschrift fort. Es sollte vermieden werden, Elemente in der Überschriftenhierarchie auszulassen.

Beispiel 4.3. Falsche Verschachtelung von Überschriften

Fügen Sie in eine der existierenden Übungsdateien folgendes ein:

```
<h1>Erstes Kapitel</h1>

<!-- Allgemeines zum Dokument -->

<h3>Unterkapitel</h3>

<!-- h3 folgt direkt auf h1. h2 wurde ausgelassen -->
```

4.1.3.2. Absätze

Absätze können in XHTML mit Hilfe des Elementes `p` ausgezeichnet werden.

Beispiel 4.4. Absätze mit dem Element `p`

Fügen Sie in eine der existierenden Übungsdateien folgendes ein:

```
<p>Das hier, das ist ein Absatz. Absätze können
  andere Elemente enthalten.</p>
```

4.1.3.3. Blockzitate

Ein Blockzitat ist ein etwas umfangreicheres Zitat aus einem anderen Text, das nicht zum aktuellen Absatz gehört.

Beispiel 4.5. Blockzitat

Fügen Sie in eine der existierenden Übungsdateien folgendes ein:

```
<blockquote>
  <p>Artikel 1: Menschenwürde; Grundrechtsbindung der
    staatlichen Gewalt</p>

  <ol>
    <li>
      <p>Die Würde des Menschen ist unantastbar. Sie zu achten
        und zu schützen ist Verpflichtung aller staatlichen
        Gewalten.</p>
    </li>
    <li>
      <p>Das Deutsche Volk bekennt sich darum zu ↺
unverletzlichen
        und unveräußerlichen Menschenrechten als Grundlage ↺
jeder
        menschlichen Gemeinschaft, des Friedens und der
        Gerechtigkeit in der Welt.</p>
    </li>
    <li>
      <p>Die nachfolgenden Grundrechte binden Gesetzgebung,
        vollziehende Gewalt und Rechtsprechung als
        unmittelbar geltendes Recht.</p>
    </li>
  </ol>
```

```
</blockquote>
```

4.1.3.4. Listen

XHTML kennt drei Arten von Listen: sortierte, unsortierte und Definitionslisten. Ein Eintrag in einer sortierten Liste wird üblicherweise mit einer Nummer versehen, Einträge in unsortierten Listen hingegen mit einem Aufzählungspunkt. Definitionslisten wiederum bestehen aus zwei Teilen: Der erste enthält den Begriff der definiert werden soll und der zweite dessen Erläuterung.

Sortierte Listen werden mit dem Element `ol` (für *ordered list*) ausgezeichnet, unsortierte Listen mit `ul` (für *unordered list*) und Definitionslisten mit `dl`.

Listenpunkte sortierter und unsortierter Listen werden mit dem Element `li` ausgezeichnet, welches Text oder andere Blockelemente enthalten kann. Begriffe, die in einer Definitionslisten enthalten sind, werden mit dem Element `dt` (für *definition term*) ausgezeichnet. Die Erklärung zu diesem Begriff wird mit Hilfe des Elementes `dd` (für *definition description*) markiert. So wie `li`, kann das Element `dd` ebenfalls andere Blockelemente aufnehmen.

Beispiel 4.6. Listen mit `ul` und `ol` erstellen

Fügen Sie in eine der existierenden Übungsdateien folgendes ein:

```
<p>Jetzt folgt eine unsortierte Liste. Wahrscheinlich werden
  die einzelnen Einträge mit einem vorangehenden Punkt ↺
dargestellt.</p>

<ul>
  <li>Erster Eintrag</li>

  <li>Zweiter Eintrag</li>

  <li>Dritter Eintrag</li>
</ul>

<p>Die zweite Liste ist sortiert und ihre Einträge bestehen ↺
aus mehreren Absätzen.
  Jeder Listeneintrag ist nummeriert.</p>

<ol>
  <li><p>Das ist der erste Eintrag mit nur einem Absatz.</p></
li>

  <li><p>Das ist der erste Absatz des zweiten Eintrags.</p>
```

```
    <p>Und das ist der zweite Absatz des zweiten Eintrags.</
p></li>

  <li><p>Der dritte Eintrag besteht ebenfalls nur aus einem ↺
Eintrag.</p></li>
</ol>
```

Beispiel 4.7. Definitionslisten mit `dl` erstellen

Fügen Sie in eine der existierenden Übungsdateien folgendes ein:

```
<dl>
  <dt>Erster Begriff</dt>

  <dd><p>Erster Absatz der Erklärung</p>

    <p>Zweiter Absatz der Erklärung.</p></dd>

  <dt>Zweiter Begriff</dt>

  <dd><p>Erster Absatz der Erklärung.</p></dd>

  <dt>Dritter Begriff</dt>

  <dd>Erster Absatz der Erklärung zum dritten Begriff.</dd>
</dl>
```

4.1.3.5. Vorformatierter Text

In einigen Fällen ist es gewollt, dass die Formatierung eines Textes im Quelldokument erhalten bleibt, damit der Leser diesen genau so sieht, wie ihn der Autor erstellt hat. In der XHTML-Spezifikation ist dafür das Element `pre` vorgesehen, welches dafür sorgt, dass Zeilenumbrüche erhalten bleiben und Leerzeichen nicht zusammengefasst werden. Browser verwenden für den Inhalt des Elementes `pre` üblicherweise eine Fixschrift.

Beispiel 4.8. Vorformatierten Text mit `pre` erstellen

Der Originaltext einer E-Mail läßt sich beispielsweise wie folgt einbinden:

```
<pre>  From: nik@FreeBSD.org
  To: freebsd-doc@FreeBSD.org
```

```
  Subject: Neue Version verfügbar

  Es ist eine neue Version der Fibel für neue Mitarbeiter am
  FreeBSD-Dokumentationsprojekt verfügbar:

    &lt;URL:http://people.FreeBSD.org/~nik/primer/index.↺
html&gt;

  Kommentare und Anmerkungen sind willkommen.

  N</pre>
```

Beachten Sie, dass < und & nach wie vor als Sonderzeichen erkannt werden. Daher wird in diesem Beispiel auch `<` an Stelle von < verwendet. Aus dem gleichen Grund wurde auch `>` an Stelle von > verwendet. Achten Sie also stets auf Sonderzeichen, wenn Sie normalen Text aus E-Mails, Programmcode oder einer anderen Quelle kopieren.

4.1.3.6. Tabellen

Anmerkung

Die meisten Textbrowser, beispielsweise Lynx, können Tabellen nicht besonders gut darstellen. Deshalb sollten Auszeichnungsalternativen in Betracht gezogen werden, um eine angemessene Darstellung sicherzustellen.

Tabellen lassen sich in XHTML mit Hilfe des Elements `table` auszeichnen. Eine Tabelle setzt sich aus einer oder mehreren Zeilen (`tr`) zusammen, von denen jede mindestens eine Zelle (`td`) enthält. Zellen können wiederum andere Blockelemente, wie Absätze oder Listen, enthalten. Auch können sie auch andere Tabellen aufnehmen, wobei die Verschachtelungstiefe unbegrenzt ist. Soll die Tabellenzelle nur einen Textabsatz enthalten, ist es nicht notwendig den Text mit einem `p` zu umschließen.

Beispiel 4.9. Einfache Tabelle mit `table`

Fügen Sie in eine der existierenden Übungsdateien folgendes ein:

```
<p>Eine einfache 2x2 Tabelle.</p>

<table>
  <tr>
```

```
    <td>Obere linke Zelle</td>

    <td>Obere rechte Zelle</td>
  </tr>

  <tr>
    <td>Untere linke Zelle</td>

    <td>Untere rechte Zelle</td>
  </tr>
</table>
```

XHTML kennt die Möglichkeit, dass sich eine Zelle mehrere Zeilen und/oder Spalten erstrecken kann. Sollen beispielsweise mehrere Spalten zusammenfassen werden, kann dies mit Hilfe des Attributes `colspan` erreicht werden, indem man ihm die Anzahl der zusammenzufassenden Spalten zuweist. Ähnliches gilt für die Zusammenfassung von Zeilen: Hierfür wird dem Attribut `rowspan` die Anzahl der zusammenzufassenden Zeilen zugewiesen.

Beispiel 4.10. Anwendung des Attributes `rowspan`

```
<p>Diese Tabelle besteht aus einer langen Zelle auf der
  linken Seite und zwei kleineren Zellen auf der rechten.</p>

<table>
  <tr>
    <td rowspan="2">Lang und dünn</td>
  </tr>

  <tr>
    <td>Obere Zelle</td>

    <td>Untere Zelle</td>
  </tr>
</table>
```

Beispiel 4.11. Anwendung des Attributes `colspan`

```
<p>Eine breite Zeile oben und zwei schmalere Zeilen
  darunter.</p>
```

```
<table>
  <tr>
    <td colspan="2">Obere Zelle</td>
  </tr>
  <tr>
    <td>Linke untere Zelle</td>

    <td>Rechte untere Zelle</td>
  </tr>
</table>
```

Beispiel 4.12. Gemeinsame Anwendung der Attrbute **rowspan** und **colspan**

```
<p>Eine Tabelle mit 3-mal-3 Zellen. Oben links
  werden 2 mal 2 Zelle zusammengezogen.</p>

<table>
  <tr>
    <td colspan="2" rowspan="2">Große obere linke Zelle</td>

    <td>Obere rechte Zelle</td>
  </tr>

  <tr>
    <td>Mittlere rechte Zelle</td>
  </tr>

  <tr>
    <td>Untere linke Zelle</td>

    <td>Untere mittlere Zelle</td>

    <td>Untere rechte Zelle</td>
  </tr>
</table>
```

4.1.4. Flußelemente

4.1.4.1. Hervorheben von Information

Sollen sich bestimmte Informationen von anderen optisch abheben, kann dies mit den HTML-Tags `strong` und `em` erreicht werden. `strong` stellt dabei eine stärkere Hervorhebung als `em` dar, wobei mit `strong` ausgezeichnete Elemente fett und mit `em` ausgezeich-

nete Elemente kursiv dargestellt werden. Allerdings ist diese Aussage nicht verlässlich, da die Darstellung vom Browser abhängig ist. In der Praxis ist es so, dass Webseiten nur strukturelle und semantische Informationen enthalten und Stylesheets später auf diese Informationen angewendet werden. Beachten Sie also die Semantik und nicht die Formatierung wenn Sie mit diesen Tags arbeiten.

Beispiel 4.13. Text mit `em` und `strong` hervorheben

```
<p><em>Dieses</em> Wort ist hervorgehoben,
  während <strong>dieses noch stärker hervorgehoben ist.</p>
```

4.1.4.2. Nicht-proportionale Schrift für Texte

Der Tag `tt` erlaubt es, Text in einer schreibmaschinenähnlichen Schrift darzustellen.

Beispiel 4.14. Nicht-proportionale Schrift mit `tt`

```
<p>Dieses Dokument wurde ursprünglich von
  Nik Clayton geschrieben. Nick Clayton kann unter der E-Mail-
Adresse
  <tt>nik@FreeBSD.org</tt> erreicht werden.</p>
```

4.1.5. Links

Anmerkung

Bei Links handelt es sich ebenfalls Flußelemente.

4.1.5.1. Auf andere Dokumente im WWW verweisen

Um auf ein anderes Dokument im WWW zu verweisen, müssen Sie die URL dieses Dokuments kennen.

Links auf andere Dokumente im WWW werden in HTML durch den Tag `a` und dessen Attribute `href`, das die Zieladresse enthält, angelegt. Der Inhalt des Elementes wird selbst zum

Link und seine Darstellung erfolgt verschieden vom übrigen Text. Meist geschieht das durch eine andere Schriftfarbe oder dadurch, dass der Linktext unterstrichen wird.

Beispiel 4.15. `<a href="...">` benutzen

```
<p>Weitere Informationen stehen auf der
  <a href="http://www.FreeBSD.org/">FreeBSD-Webseite</a> zur ↺
Verfügung.</p>
```

Beim Aufruf dieses Links wird das referenzierte Dokument vom Browser geladen und mit dessen Seitenanfang dargestellt.

4.1.5.2. Auf bestimmte Dokumentenabschnitte verweisen

XHTML unterstützt neben einfachen Links auch solche, die auf einen bestimmten Abschnitt innerhalb eines Dokumentes verweisen. Dazu müssen die Abschnitte, auf die verwiesen werden soll, mit Hilfe von sogenannten „Ankern“ markiert werden. Diese Anker können ebenfalls mit Hilfe des Tags `a` gesetzt werden, nur das anstelle von `href` das Attribut `name` gesetzt werden muss.

Beispiel 4.16. Anwendung von `<a id="...">`

```
<p><a name="absatz1">Auf</a> diesen Absatz kann mit
  Hilfe seines Namens (<tt>absatz1</tt>) verwiesen werden.</p>
```

Um auf einen so gekennzeichneten Abschnitt zu verweisen, muss die URL des Dokumentes um das Zeichen `#` und den Namen des Zielankers erweitert werden.

Beispiel 4.17. Auf einen Abschnitt eines anderen Dokumentes verweisen

Für dieses Beispiel wird davon ausgegangen, dass der mit `absatz1` gekennzeichnete Absatz sich in der XHTML-Datei `foo.html` befindet.

```
<p>Weitere Informationen können im
```

```
<a href="foo.html#absatz1">ersten Absatz</a> der Datei
<tt>foo.html</tt> gefunden werden.</p>
```

4.2. Die DocBook DTD

DocBook wurde ursprünglich von HaL Computer Systems und O'Reilly & Associates als DTD für das Erstellen von technischen Dokumenten entwickelt [2]. Seit 1998 wird es vom DocBook Technical Committee gewartet. DocBook ist sehr stark auf die Beschreibung von Inhalten, und nicht auf die Darstellung des Inhalts ausgerichtet. Damit steht es im Gegensatz zu LinuxDoc und XHTML.

Formelle und informelle Elemente

Einige Elemente der DocBook DTD sind in zwei Varianten vorhanden: *formell* und *informell*. Üblicherweise besitzt die formelle Variante einen Titel, dem der eigentliche Elementinhalt folgt. Die informelle Variante hingegen hat keinen Titel.

Die DocBook DTD ist in der Ports-Sammlung im Port textproc/docbook enthalten und wird bei der Installation des Metaports textproc/docproj automatisch installiert.

4.2.1. Die FreeBSD-Erweiterungen

Für das FDP wurde die DocBook DTD durch das FreeBSD-Dokumentationsproject um zusätzliche Elemente erweitert, um damit präzisiere Auszeichnungsmöglichkeiten zur Verfügung zu haben. Sofern im folgenden FreeBSD-spezifische Elemente genutzt werden, wird explizit darauf hingewiesen werden.

Wenn nachfolgend im Text der Begriff „DocBook“ verwendet wird, ist damit die durch das FDP erweiterte Version der DocBook DTD gemeint.

Anmerkung

Die durch das FDP vorgenommenen Erweiterungen sind nicht FreeBSD-spezifisch. Sie wurden lediglich vorgenommen, da sie für die Arbeit des FDPs als nützlich erschienen. Für den Fall, das in den anderen *nix-Lagern (NetBSD, OpenBSD, Linux,...) Interesse dar-

[2] Einen kurzen historischen Abriss finden Sie unter http://www.oasis-open.org/docbook/intro.shtml#d0e41.

> an besteht, gemeinsam eine Standarderweiterung für die DocBook DTD zu entwickeln, kann mit dem Documentation Engineering Team <doceng@FreeBSD.org > Verbindung aufgenommen werden.

Zum jetzigen Zeitpunkt sind die FreeBSD-Erweiterungen nicht Bestandteil der Ports-Sammlung. Sie werden im FreeBSD-Subversion-Repository (head/share/xml/freebsd.dtd) verwaltet.

4.2.2. Formelle Öffentliche Bezeichner

In Übereinstimmung mir der DocBook-Richtlinie zur Erstellung von Bezeichnern für DocBook-Erweiterungen lautet der Bezeichner der erweiterten FreeBSD-Variante:

```
PUBLIC "-//FreeBSD//DTD DocBook V4.2-Based Extension//EN"
```

4.2.3. Die Struktur von DocBook-Dokumenten

DocBook erlaubt es, Dokumente auf verschiedene Weise zu strukturieren. Innerhalb des FDPs werden hauptsächlich zwei Arten von DocBook-Dokumenten verwendet: Buch und Artikel. Beide unterscheiden sich darin, dass ein Buch auf der obersten Ebene durch `chapter`-Elemente strukturiert wird. Sollte das noch nicht ausreichend sein, können die einzelnen Kapitel eines Buches mit Hilfe des Elementes `part` in Teile aufgespalten werden. Das Handbuch ist beispielsweise auf diese Weise aufgebaut.

Kapitel (`chapter`) können weiterhin in Unterkapitel unterteilt werden. Diese werden durch die Elemente `sect1` ausgezeichnet. Soll ein Unterkapitel selbst weitere Unterkapitel enthalten, kann das über das Element `sect2` geschehen. Diese Unterteilung kann bis zur Tiefe von fünf Unterkapiteln – über die Elemente `sect3`, `sect4` und `sect5` – fortgeführt werden. Der eigentliche Inhalt, um den es ja in dem Artikel oder Buch geht, wird unterhalb der hier genannten Elemente eingefügt.

Vom Aufbau her ist ein Artikel einfacher strukturiert als ein Buch. So kann ein Artikel beispielsweise keine Kapitel (`chapter`) enthalten. Stattdessen kann der Inhalt eines Artikels nur durch die schon bekannten `sect`*N*-Elemente in einen oder mehrere Abschnitte gegliedert werden. Überlegen Sie sich vor dem Schreiben eines Textes, ob der zu schreibende Text am besten als Buch oder als Artikel angelegt wird. Artikel eignen sich besser für Texte, die nicht in mehrere Kapitel aufgeteilt werden müssen und mit einem Umfang von ungefähr 20 bis 25 Seiten vergleichsweise kurz sind. Natürlich ist das nur eine Richtlinie. Bücher sind dementsprechend am besten für lange Texte geeignet, die sich sinnvoll in Kapitel unterteilen lassen und möglichweiser noch Anhänge und ähnliches enthalten können.

Alle Tutorien von FreeBSD sind als Artikel verfasst, während hingegen die FreeBSD-FAQ und das FreeBSD-Handbuch als Bücher verfasst wurden.

4.2.3.1. Bücher schreiben

Der Inhalt eines Buches wird in einem `book`-Element abgelegt. Neben dem Textteil des Buches kann dieses Element weitergehende Informationen über das Buch selbst, wie Meta-Informationen zum Erstellen eines Stichwortverzeichnisses oder zusätzliche Inhalte zum Erstellen einer Titelei, enthalten. Diese zusätzlichen Inhalte sollten in einem `bookinfo`-Element abgelegt werden.

Beispiel 4.18. Buchvorlage **`book`** mit **`bookinfo`**

```
<book>
  <bookinfo>
    <title>Titel</title>
    <author>
      <firstname>Vorname</firstname>
      <surname>Nachname</surname>
      <affiliation>
        <address><email>E-Mail-Adresse</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>1998</year>
      <holder role="mailto:E-Mail-Adresse ">Vollständiger Name </
holder>
    </copyright>

    <releaseinfo>$FreeBSD$</releaseinfo>

    <abstract>
      <para>Kurze Zusammenfassung des Buchinhaltes.  </para>
    </abstract>
  </bookinfo>

  …

</book>
```

4.2.3.2. Artikel schreiben

Der Inhalt eines Artikels wird in einem `article`-Element abgelegt. Neben dem Textteil kann dieses Element weitere Teile, wie Meta-Informationen zum Erstellen eines Stichwortverzeichnisses oder zusätzliche Inhalte zum Erstellen einer Titelei, enthalten. Analog zu einem Buch, sollten diese Informationen in einem `articleinfo`-Element abgelegt werden.

Beispiel 4.19. Artikelvorlage `article` mit `articleinfo`

```
<article>
  <articleinfo>
    <title>Titel</title>

    <author>
      <firstname>Vorname</firstname>
      <surname>Nachname</surname>
      <affiliation>
        <address><email>E-Mail-Adresse</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>1998</year>
      <holder role="mailto:E-Mail-Adresse ">Vollständiger Name </
holder>
    </copyright>

    <releaseinfo>$FreeBSD$</releaseinfo>

    <abstract>
      <para>Kurze Zusammenfassung des Artikelinhalts.  </para>
    </abstract>
  </articleinfo>

  …

</article>
```

4.2.3.3. Kapitel

Kapitel werden mit dem `chapter`-Element angelegt und müssen ein `title`-Element enthalten. Verwendet werden können sie nur in Büchern.

Beispiel 4.20. Ein einfaches Kapitel

```
<chapter>
  <title>Kapitelüberschrift</title>

  &hellip;
```

```
</chapter>
```

Kapitel können nicht leer sein. Neben einem `title`-Element müssen sie weiteren Inhalt beinhalten. Falls ein leeres Kapitel benötig wird, kann dies durch das Einfügen eines leeren Absatzes (`para`) erreicht werden.

Beispiel 4.21. Ein leeres Kapitel

```
<chapter>
  <title>Das ist ein leeres Kapitel</title>

  <para></para>
</chapter>
```

4.2.3.4. Unterkapitel

Bücher werden auf der obersten Gliederungsebene durch `chapter`-Elemente in Kapitel unterteilt. Eine weitergehende Untergliederung kann durch das Anlegen von Unterkapiteln erreicht werden. Im Gegensatz zu Kapiteln, die durch `chapter`-Elemente ausgezeichnet werden, erfolgt die Auszeichnung von Unterkapitel mit dem Element `sect`*n*. Das *n* in Elementnamen trifft eine Aussage über die Gliederungstiefe, auf der sich das Unterkapitel befindet. Ein `sect1`-Element kann mehrere Elemente vom Typ `sect2` enthalten, die die Unterkapitel der nächsten Gliederungsebene darstellen. `sect5` ist das letzte Element, das auf diese Art zur Gliederung eingesetzt werden kann.

Beispiel 4.22. Unterkapitel

```
<chapter>
  <title>Ein Beispielkapitel</title>

  <para>Ein beliebiger Text.</para>

  <sect1>
    <title>Erster Abschnitt (1.1)</title>

    &hellip;
  </sect1>

  <sect1>
    <title>Zweiter Abschnitt (1.2)</title>
```

```
    <sect2>
      <title>Erster Unterabschnitt (1.2.1)</title>

      <sect3>
        <title>Erster Unterunterabschnitt (1.2.1.1)</title>

         &hellip;
      </sect3>
    </sect2>

    <sect2>
      <title>Zweiter Unterabschnitt (1.2.2)</title>

      &hellip;
    </sect2>
  </sect1>
</chapter>
```

Anmerkung

Die Unterkapitel dieses Beispiels wurden zu Demonstrationszwecken manuell durchnummeriert. In „normalen“ Dokumenten wird diese Aufgabe von den Stylesheets übernommen.

4.2.3.5. Bücher mittels `part` unterteilen

In den Fällen, in denen die Unterteilung eines Buches in Kapitel nicht ausreichend ist, können mehrere Kapitel mit dem Element `part` zu einem Teil zusammengefasst werden.

```
<part>
  <title>Einführung</title>

  <chapter>
    <title>Überblick</title>

     &hellip;
  </chapter>

  <chapter>
    <title>Was ist FreeBSD?</title>

    &hellip;
  </chapter>

  <chapter>
```

```
    <title>Die Geschichte von FreeBSD</title>

    &hellip;
  </chapter>
</part>
```

4.2.4. Blockelemente

4.2.4.1. Absätze

DocBook kennt drei Arten von Absätzen: Absätze mit Überschrift (`formalpara`), normale Absätze (`para`) und einfache Absätze (`simpara`).

Normale Absätze und einfache Absätze unterscheiden sich dadurch, dass innerhalb von `para` Blockelemente erlaubt sind, innerhalb von `simpara` hingegen nicht. Es ist empfehlenswert, `para` den Vorzug zu geben.

Beispiel 4.23. Absatz mit **para**

```
<para>Das ist ein Absatz. Absätze können fast jedes andere
  Element aufnehmen.</para>
```

Darstellung:

Das ist ein Absatz. Absätze können fast jedes andere Element aufnehmen.

4.2.4.2. Blockzitate

Blockzitate sind textlich umfangreichere Zitate aus einem anderen Text, die nicht innerhalb des aktuellen Absatzes angezeigt werden sollen. Wahlweise können Blockzitate eine Überschrift haben und die Zitatquelle nennen.

Beispiel 4.24. **blockquote**

```
<para>Ein Auszug aus dem Grundgesetz:</para>

<blockquote>
  <title>Menschenwürde; Grundrechtsbindung der staatlichen ↺
Gewalt</title>

  <attribution>Aus dem Grundgesetz</attribution>

  <orderedlist>
    <listitem>
```

```
      <para>Die Würde des Menschen ist unantastbar. Sie zu ↺
achten
        und zu schützen ist Verpflichtung aller staatlichen
        Gewalten.</para>
    </listitem>
    <listitem>
      <para>Das Deutsche Volk bekennt sich darum zu ↺
unverletzlichen
        und unveräußerlichen Menschenrechten als Grundlage ↺
jeder
        menschlichen Gemeinschaft, des Friedens und der
        Gerechtigkeit in der Welt.</para>
    </listitem>
    <listitem>
      <para>Die nachfolgenden Grundrechte binden Gesetzgebung,
        vollziehende Gewalt und Rechtsprechung als
        unmittelbar geltendes Recht.</para>
    </listitem>
  </orderedlist>
</blockquote>
```

Darstellung:

> Menschenwürde; Grundrechtsbindung der staatlichen Gewalt
>
> 1. Die Würde des Menschen ist unantastbar. Sie zu achten und zu schützen ist Verpflichtung aller staatlichen Gewalten.
>
> 2. Das Deutsche Volk bekennt sich darum zu unverletzlichen und unveräußerlichen Menschenrechten als Grundlage jeder menschlichen Gemeinschaft, des Friedens und der Gerechtigkeit in der Welt.
>
> 3. Die nachfolgenden Grundrechte binden Gesetzgebung, vollziehende Gewalt und Rechtsprechung als unmittelbar geltendes Recht.
>
> —Aus dem Grundgesetz

4.2.4.3. Tipps, Anmerkungen, Warnungen, wichtige Informationen und Randbemerkungen

In bestimmten Fällen kann es nützlich sein, dem Leser zusätzliche Informationen zu geben, die sich vom Haupttext abheben, damit der Leser sie besser wahrnimmt. Abhängig von der Art der Information, können solche Stellen mit einem der Elemente `tip` (für Tipps), `note` (für Anmerkungen), `warning` (für Warnungen), `caution` (für besonders ernstzunehmende Warnungen) und `important` (für wichtige Anmerkungen) ausgezeich-

net werden. Trifft keines dieser Elemente für die auszuzeichnende Stelle zu, sollte diese mit dem Element `sidebar` ausgezeichnet werden.

Da die richtige Einordnung einer auszuzeichnenden Textstelle nicht immer leicht zu treffen ist, werden in der DocBook-Dokumentation folgende Empfehlungen gegeben:

- Eine Anmerkung (`note`) ist eine Information, die von jedem Leser beachtet werden sollte.
- Eine wichtige Anmerkung (`important`) eine Variation einer Anmerkung.
- Eine Warnung (`warning`) betrifft einen möglichen Hardwareschaden oder weist auf eine Gefahr für Leib und Leben hin.
- Eine besonders ernstzunehmende Warnung (`caution`) betrifft einen möglichen Datenverlust oder Softwareschaden.

Beispiel 4.25. **`warning`**

```
<warning>
  <para>Wenn Sie FreeBSD auf Ihrer Festplatte installieren,
    kann es sein, da&amp;szlig; Sie Windows nie mehr benutzen ↺
wollen.</para>
</warning>
```

Eine Warnung wird wie folgt dargestellt:

Warnung

Wenn Sie FreeBSD auf Ihrer Festplatte installieren, kann es sein, dass Sie Windows nie mehr benutzen wollen.

4.2.4.4. Listen und Handlungsanweisungen

Listen sind ein oft gebrauchtes Hilfsmittel, wenn es darum geht, Informationen für den Benutzer übersichtlich darzustellen oder eine Abfolge von Arbeitsschritten zu beschreiben, die notwendig sind, um ein bestimmtes Ziel zu erreichen. Zur Auszeichnung von Listen stellt DocBook die Elemente `itemizedlist`, `orderedlist` und `procedure` zur Verfügung.[3].

[3] DocBook kennt noch andere Elemente für die Auszeichnung von Listen, die an dieser Stelle jedoch nicht behandelt werden.

`itemizedlist` und `orderedlist` ähneln sehr stark ihren HTML-Gegenstücken `ul` und `ol`. Beide Listenarten müssen mindestens ein Element `listitem` enthalten. Das `listitem` Element muss mindestens ein weiteres Blockelement enthalten.

`procedure` unterscheidet sich ein wenig von den vorhergehenden. Es enthält `step`-Elemente, die wiederum `step`- oder `substel`-Elemente enthalten können. Ein `step`-Element kann nur Blockelemente aufnehmen.

Beispiel 4.26. **`itemizedlist`** , **`orderedlist`** und **`procedure`**

```
<itemizedlist>
  <listitem>
    <para>Das ist das erste Listenelement.</para>
  </listitem>

  <listitem>
    <para>Das ist das zweite Listenelement.</para>
  </listitem>
</itemizedlist>

<orderedlist>
  <listitem>
    <para>Das ist das erste Aufzählungselement.</para>
  </listitem>

  <listitem>
    <para>Das ist das zweite Aufzählungselement.</para>
  </listitem>
</orderedlist>

<procedure>
  <step>
    <para>Machen Sie zuerst dies.</para>
  </step>

  <step>
    <para>Und dann machen Sie das..</para>
  </step>

  <step>
    <para>Und jetzt noch das&hellip;</para>
  </step>
</procedure>
```

Darstellung:

- Das ist das erste Listenelement.
- Das ist das zweite Listenelement.

1. Das ist das erste Aufzählungselement.

2. Das ist das zweite Aufzählungselement.

1. Machen Sie zuerst dies.

2. Und dann machen Sie das..

3. Und jetzt noch das...

4.2.4.5. Dateiinhalte auszeichnen

Technische Dokumente enthalten oft auch Konfigurationsbeispiele oder Quellcodeschnipsel. Zur Auszeichnung dieser Inhalte, stellt Docbook das Element `programmlisting` zur Verfügung. Im Gegensatz zu anderen DocBook-Elementen wird der Elementinhalt von `programmlisting` *nicht* normalisiert, das heißt, dass alle Leerzeichen, Tabulatoren und Zeilenumbrüche unverändert übernommen werden. Aus diesem Grund ist es unter anderem wichtig, dass sich der öffende Tag in der selben Zeile wie der Anfang des darzustellenden Textes befindet. Gleiches gilt für den schließenden Tag: Er muss sich am Ende der letzten Zeile befinden. Wird das nicht beachtet, kann es sein, dass unerwartete Leerzeichen und Leerzeilen in der Ausgabe auftauchen.

Beispiel 4.27. **`programlisting`**

```
<para>Am Ende sollte Ihr Programm wie folgt
  aussehen:</para>

<programlisting>#include &amp;lt;stdio.h&amp;gt;

int
main(void)
{
    printf("Hallo Welt!\n");
}</programlisting>
```

Die spitzen Klammern der `#include`-Anweisung können nicht direkt verwendet werden, sondern müssen über ihre Entitäten eingebunden werden.

Darstellung:

Am Ende sollte Ihr Programm wie folgt aussehen:

```
#include <stdio.h>

int
```

```
main(void)
{
    printf("Hallo Welt!\n");
}
```

4.2.4.6. Textanmerkungen

Textanmerkungen[4] sind ein Mechanismus, um auf bestimmte Stellen in einem vorhergehenden Beispiel oder Text zu verweisen.

Um solche Verweise anzulegen, müssen die betreffenden Stellen in den Beispielen (`programlisting`, `literallayout`, ...) mit `co`-Elementen markiert werden, wobei jedes Element ein eindeutiges `id`-Attribut besitzen muss. Anschließend sollte ein `calloutlist`-Element eingefügt werden, dessen Elemente sich auf die `co`-Elemente des Beispiels beziehen und die jeweiligen Anmerkungen enthalten.

Beispiel 4.28. Das `co`- und das `calloutlist` -Element

```
<para>Am Ende sollte Ihr Programm wie folgt
  aussehen:</para>

<programlisting>#include &amp;lt;stdio.h&amp;gt; <co id="co-ex-
include"/>

int <co id="co-ex-return"/>
main(void)
{
    printf("Hallo Welt\n"); <co id="co-ex-printf"/>
}</programlisting>

<calloutlist>
  <callout arearefs="co-ex-include">
    <para>Bindet die Headerdatei <filename>stdio.h</filename> ↺
ein.</para>
  </callout>

  <callout arearefs="co-ex-return">
    <para>Bestimmt den Typ des Rückgabewertes von ↺
<function>main()</function>.</para>
  </callout>

  <callout arearefs="co-ex-printf">
    <para>Ruft die Funktion <function>printf()</function> ↺
auf, die
```

[4] auf Englisch: *callout*

```
      <literal>Hallo Welt!</literal> auf der Standardausgabe ↺
ausgibt</para>
  </callout>
</calloutlist>
```

Darstellung:

Am Ende sollte Ihr Programm wie folgt aussehen:

```
#include &lt;stdio.h&gt; ❶

int ❷
main(void)
{
    printf("Hallo Welt\n"); ❸
}
```

❶ Bindet die Headerdatei `stdio.h` ein.
❷ Bestimmt den Typ des Rückgabewertes von `main()`.
❸ Ruft die Funktion `printf()` auf, die `Hallo Welt!` auf der Standardausgabe ausgibt

4.2.4.7. Tabellen

Im Gegensatz zu HTML ist es nicht notwendig, Tabellen zu Layoutzwecken einzusetzen, da die Layoutaufgabe von den Stylesheets übernommen wird. Stattdessen sollten Tabellen nur für die Auszeichnung von Daten in Tabellenform genutzt werden.

Vereinfacht betrachtet (für Details sollte die DocBook-Dokumentation zu Rate gezogen werden) besteht eine Tabelle, die entweder als formelle oder als informelle Tabelle angelegt werden kann, aus einem `table`-Element. Dieses Element selbst beinhaltet mindestens ein Element `tgroup`, das über ein Attribut die Spaltenanzahl der Tabelle bestimmt. Innerhalb des Elementes `tgroup` kann sich ein Element `thead` mit den Spaltenüberschriften und ein Element `tbody` mit dem eigentlichen Tabelleninhalt befinden. Beide Elemente beinhalten `row`-Elemente, die wiederum `entry`-Elemente beinhalten. Jedes `entry`-Element stellt eine einzelne Tabellenzelle dar.

Beispiel 4.29. Tabellen mittels `informaltable` auszeichnen

```
<informaltable pgwide="1">
  <tgroup cols="2">
    <thead>
      <row>
        <entry>Spaltenüberschrift 1</entry>
        <entry>Spaltenüberschrift 2</entry>
      </row>
```

```
    </thead>

    <tbody>
      <row>
        <entry>Zeile 1, Spalte 1</entry>
        <entry>Zeile 1, Spalte 2</entry>
      </row>

      <row>
        <entry>Zeile 2, Spalte 1</entry>
        <entry>Zeile 2, Spalte 2</entry>
      </row>
    </tbody>
  </tgroup>
</informaltable>
```

Darstellung:

Spaltenüberschrift 1	Spaltenüberschrift 2
Zeile 1, Spalte 1	Zeile 1, Spalte 2
Zeile 2, Spalte 1	Zeile 2, Spalte 2

Verwenden Sie stets das Attribut `pgwide` mit dem Wert `1`, wenn Sie das Element `informaltable` benutzen. Ein Bug des Internet Explorers verhindert ansonsten die korrekte Darstellung dieser Tabellen.

Soll die Tabelle keinen Rand haben, kann das Attribut `frame` mit dem Wert `none` dem Element `informaltable` hinzugefügt werden (`<informaltable frame="none">`)).

Beispiel 4.30. Tabelle mit Attribut `frame="none"`

Darstellung:

Spaltenüberschrift 1	Spaltenüberschrift 2
Zeile 1, Spalte 1	Zeile 1, Spalte 2
Zeile 2, Spalte 1	Zeile 2, Spalte 2

4.2.4.8. Beispiele für den Leser

Oft gilt es, für dem Benutzer Beispiele zu geben, die er dann selber nachvollziehen soll. Meist handelt es sich dabei um interaktive Dialoge zwischen Mensch und Maschine: Der

Benutzer gibt einen Befehl ein und erhält eine Antwort vom System. Ein Satz von speziellen Elementen und Entitäten unterstützt den Autor bei der Auszeichnung solcher Textstellen:

`screen`
: Gedacht zur Auszeichnung von Bildschirminhalten. Im Unterschied zu anderen Elementen werden Leerzeichen innerhalb des Elementes `screen` unverändert übernommen.

`prompt`, `&prompt.root;` und `&prompt.user;`
: Eingabeaufforderungen des Rechners (Betriebssystem, Shell oder Anwendung) sind ein häufig auftretender Teil dessen, was der Benutzer auf dem Bildschirm zu sehen bekommt. Sie sollten mit `prompt` ausgezeichnet werden.

  Ein Spezialfall sind die beiden Eingabeaufforderungen der Shell für normale Benutzer und den Superuser `root`. Jedes mal wenn auf eine von diesen beiden Nutzerrollen hingewiesen werden soll, sollte entweder `&prompt.root;` oder `&prompt.user;` eingesetzt werden. Beide Entitäten können auch außerhalb von `screen` verwendet werden.

Anmerkung

`&prompt.root;` und `&prompt.user;` sind FreeBSD-spezifische Erweiterungen der DocBook DTD und nicht in der originalen DocBook DTD enthalten.

`userinput`
: Das Element `userinput` ist für die Auszeichnung von Benutzereingaben gedacht.

Beispiel 4.31. `screen`, `prompt` und `userinput`

```
<screen>&prompt.user; <userinput>ls -1</userinput>
foo1
foo2
foo3
&prompt.user; <userinput>ls -1 | grep foo2</userinput>
foo2
&prompt.user; <userinput>su</userinput>
<prompt>Password: </prompt>
&prompt.root; <userinput>cat foo2</userinput>
This is the file called 'foo2'</screen>
```

Darstellung:

```
% ls -1
foo1
foo2
foo3
% ls -1 | grep foo2
foo2
% su
Password:
# cat foo2
This is the file called 'foo2'
```

Anmerkung

Obgleich der Inhalt der Datei `foo2` in dem obigen Beispiel angezeigt wird, sollte dieser nicht mit `programlisting` ausgezeichnet werden. Vielmehr sollte `programlisting` einzig und allein für die Darstellung von Dateifragmenten außerhalb von Benutzeraktionen gewählt werden.

4.2.5. Flußelemente

4.2.5.1. Hervorhebungen

Wenn es darum geht bestimmte Wörter oder Textstellen hervorzuheben, sollte dafür das Element `emphasis` verwendet werden. Das so ausgezeichnete Text wird dann kursiv oder fett dargestellt; im Falle einer Sprachausgabe würde es anders betont werden.

Im Gegensatz zu den HTML mit seinen Elementen `b` und `i`, kennt DocBook keinen Weg, um diese Darstellung zu ändern[5]. Handelt es sich bei dem darzustellenden um eine wichtige Information, kann alternativ `important` verwendet werden.

Beispiel 4.32. Das Element `emphasis`

```
<para>FreeBSD ist zweifelslos <emphasis>das</emphasis> führende
  Unix-artige Betriebssystem für die Intel-Plattform.</para>
```

Darstellung:

[5] Anmerkung des Übersetzers: Hier sollte man sich noch einmal ins Gedächtnis rufen, dass mittels der DocBook DTD nur Inhalte ausgezeichnet werden und nicht das Layout bestimmt wird.

FreeBSD ist zweifelslos *das* führende Unix-artige Bestriebssystem für die Intel-Plattform.

4.2.5.2. Zitate

Um einen Auszug aus einer anderen Quelle zu zitieren oder kenntlich zu machen, dass eine bestimmte Wendung im übertragenen Sinne zu verstehen ist, kann der betreffende Text mit Hilfe des Elementes `quote` ausgezeichnet werden. Innerhalb von `quote` können die meisten der normalerweise zur Verfügung stehenden Elemente genutzt werden.

Beispiel 4.33. Richtig zitieren

```
<para>Es sollte immer sichergestellt werden, dass die Suche ↺
die Grenzen
  zwischen <quote>lokaler und öffentlicher Administration</
quote>
  (RFC 1535) einhält.</para>
```

Darstellung:

Es sollte immer sichergestellt werden, das die Suche die Grenzen zwischen „lokaler und öffentlicher Administration“ (RFC 1535) einhält.

4.2.5.3. Tasten, Maustasten und Tastenkombinationen

Das Element `keycap` beschreibt eine bestimmte Taste der Tastatur. Für die Auszeichnung von Maustasten steht analog das Element `mousebutton` zur Verfügung. Mit Hilfe von `keycombo` können beliebige Tasten- und Maustastenkombinationen beschrieben werden.

Das Element `keycombo` besitzt ein Attribut `action`, dem einer der Werte `click`, `double-click`, `other`, `press`, `seq` oder `simul` zugewiesen werden kann. Die letzten beiden Werte deuten an, dass die genannte Kombination nacheinander oder gleichzeitig gedrückt werden soll. Die Stylesheets fügen zwischen die einzelnen Unterelemente von `keycombo` „+“-Zeichen ein.

Beispiel 4.34. Tasten, Maustasten und Tastenkombinationen

Diese Eingaben zeichnen Sie wie folgt aus:

```
<para>Mit der Tastenkombination <keycombo ↺
action="simul"><keycap>Alt</keycap>
```

```
  <keycap>F1</keycap></keycombo> kann auf die zweite ↺
virtuelle Konsole
  umgeschaltet werden.</para>

<para>Um <command>vi</command> zu beenden, ohne die ↺
Änderungen zu
  speichern, muss <keycombo action="seq"><keycap>Esc</keycap>
  <keycap>:</keycap><keycap>q</keycap><keycap>!</keycap>
  </keycombo> eingegeben werden.</para>

<para>Der Fenstermanager ist so konfiguriert, dass mittels
  <keycombo action="simul"><keycap>Alt</keycap>
  <mousebutton>rechter Maustaste</mousebutton> </keycombo>
   Fenster verschoben werden können.</para>
```

Darstellung:

Mit der Tastenkombination Alt+F1 kann auf die zweite virtuelle Konsole umgeschaltet werden.

Um `vi` zu beenden, ohne die Änderungen zu speichern, muss Esc : q ! eingegeben werden.

Der Fenstermanager ist so konfiguriert, dass mittels Alt+rechter Maustaste Fenster verschoben werden können.

4.2.5.4. Anwendungen, Befehle, Optionen und Hilfeseiten

Oft besteht die Notwendigkeit auf bestimmte Anwendungen und Befehle zu verweisen. Der Unterschied zwischen einer Anwendung und einem Befehl liegt darin, dass eine Anwendung ein einzelnes oder eine Gruppe von Programmen ist, mit denen eine bestimmte Aufgabe erledigt werden kann. Ein Befehl hingegen ist der Name eines Programmes, dass der Benutzer aufrufen kann[6].

Desweiteren kann es auch vorkommen, dass die von einem Programm (in einem bestimmten Fall) akzeptierten Optionen genannt werden müssen.

Schlussendlich ist es oft gewünscht, zu einem Befehl dessen Abschnitt der Manualseiten im Unix-üblichen Stil „Befehl(Zahl)“ anzugeben.

Anwendungsnamen können mit `application` ausgezeichnet werden.

Befehle können zusammen mit der betreffenden Hilfeseite über das DocBook-Element `citerefentry` ausgezeichnet werden. `citerefentry` muss zwei weitere Elemente enthalten: `refentrytitle`, für den Befehlsnamen, und `manvolnum`, für die Kategorie der Hilfeseite.

[6] Der Befehl `mozilla` startet das Programm mozilla.

Diese Art auf Befehle zu verweisen kann sehr ermüdent sein. Daher gibt es einen Satz von Allgemeinen Entitäten, der diese Arbeit erleichtert. Er ist in der Datei `doc/share/xml/man-refs.ent` enhalten und kann über den folgenden Bezeichner eingebunden werden:

```
PUBLIC "-//FreeBSD//ENTITIES DocBook Manual Page Entities//EN"
```

Jede Entität in dieser Datei ist wie folgt aufgebaut: `&man.Hilfeseite.Kategorie;` .

Der Anfang eines Dokumentes, das diese Entitäten einbindet, könnte so aussehen:

```
<!DOCTYPE book PUBLIC "-//FreeBSD//DTD DocBook V4.1-Based Extension//
EN" [
<!ENTITY % man PUBLIC "-//FreeBSD//ENTITIES DocBook Manual Page ↺
Entities//EN">
%man; … -]>
```

Um Befehle innerhalb des Fließtextes auszuzeichnen, kann das Element `command` genutzt werden. Die Optionen eines Befehles können mit Hilfe von `option` ausgezeichnet werden.

Wenn man sich mehrmals hintereinander auf den gleichen Befehl bezieht, sollte man beim ersten Auftreten die Notation `&man.command.section;` verwenden. Für alle folgenden Referenzen sollte hingegen `command` verwendet werden. Dadurch verbessert sich das Erscheinungsbild, insbesondere von HTML, deutlich.

Die Unterscheidung zwischen `command` und `application` kann schwer sein, und manchmal ist die Entscheidung, welches Element das richtige ist, nicht leicht. Das folgende Beispiel soll diese Unterscheidung erleichtern.

Beispiel 4.35. Anwendungen, Befehle und Optionen

```
<para><application>Sendmail</application> ist der ↺
verbreiteteste
  UNIX-Mailserver.</para>

<para><application>Sendmail</application> besteht aus den ↺
Programmen
  <citerefentry>
    <refentrytitle>sendmail</refentrytitle>
    <manvolnum>8</manvolnum>
  </citerefentry>, &amp;man.mailq.1;, und &amp;man.↺
newaliases.1;.</para>

<para>Mittels der Option <option>-bp</option> kann
  <citerefentry><refentrytitle>sendmail</refentrytitle>
    <manvolnum>8</manvolnum>
  </citerefentry> den Status der Mailwarteschlange ausgeben.
  Der Status der Mailwarteschlange kann durch den Befehl
  <command>sendmail -bp</command> überprüft werden.</para>
```

Darstellung:

Sendmail ist der verbreitetste UNIX-Mailserver.

Sendmail besteht aus den Programmen sendmail(8), mailq(1) sowie newaliases(1).

Mittels der Option `-bp` kann sendmail(8) den Status der Mailwarteschlange ausgeben. Der Status der Mailwarteschlange kann durch den Befehl `sendmail -bp` überprüft werden.

Anmerkung

Die Schreibweise `&man.Hilfeseite.Kategorie;` ist leichter lesbar.

4.2.5.5. Dateien, Verzeichnisse und Erweiterungen

Immer wenn in einem Text der Name einer Datei, eines Verzeichnisses oder eine Dateierweiterung vorkommt, sollte die betreffende Stelle mit dem Element `filename` ausgezeichnet werden.

Beispiel 4.36. Das Element `filename`

```
<para>Die SGML-Quellen des
     englischen Handbuches befinden sich im Verzeichnis
     <filename
     class="directory">/usr/doc/en/handbook/</filename>. In
     diesem Verzeichnis befindet sich eine Datei
     <filename>handbook.xml</filename>. Desweiteren sollte
     sich eine Datei mit dem Namen
     <filenname>Makefile</filename> zusammen mit mehreren
     Dateien mit der Endung <filename>.ent</filename> in diesem
     Verzeichnis befinden.</para>
```

Darstellung:

Die SGML-Quellen des englischen Handbuches befinden sich im Verzeichnis `/usr/doc/en/handbook/` . In diesem Verzeichnis befindet sich eine Datei `handbook.xml` . Desweiteren sollte sich eine Datei mit dem Namen `Makefile` zusammen mit mehreren Dateien mit der Endung `.ent` in diesem Verzeichnis befinden.

4.2.5.6. Portnamen

FreeBSD-Erweiterung

Die hier genannten Elemente sind Bestandteil der FreeBSD-Erweiterung für DocBook und sind nicht in der originalen DocBook DTD enthalten.

An einigen Stellen ist es notwendig, den Namen eines Ports aus FreeBSDs Ports-Sammlung in Dokumenten zu verwenden. In diesem Fall sollte ebenfalls das Element `filename` eingesetzt werden, dabei aber dem Element das Attribut `role` mit dem Wert `package` zugewiesen werden. Da die Ports-Sammlung an jeder beliebigen Stelle im Dateisystem installiert werden kann, sollte `filename` nur die Kategorie und den Namen des Ports enthalten, aber nicht das Verzeichnis `/usr/ports` .

Beispiel 4.37. Portsnamen und das Element `filename`

```
<para>Wenn Sie Ihr Netz und dessen Datenverkehr analysieren
  möchten, dann installieren Sie bitte den Port <filename
  role="package">net/ethereal</filename>.</para>
```

Darstellung:

Wenn Sie Ihr Netz und dessen Datenverkehr analysieren möchten, dann installieren Sie bitte den Port net/ethereal.

4.2.5.7. Gerätedateien unterhalb von `/dev`

FreeBSD-Erweiterung

Die hier genannten Elemente sind Bestandteil der FreeBSD-Erweiterung für DocBook und sind nicht in der originalen DocBook DTD enthalten.

Wird in einem Dokument Bezug auf Gerätedateien unterhalb von `dev` genommen, dann gibt es zwei Möglichkeiten diese auszuzeichnen. Zum einen kann man sich auf das Gerät beziehen, so wie es unter `/dev` zu finden ist, und zum anderen kann man sich auf den

Gerätenamen beziehen, wie es innerhalb des Kerns verwendet wird. Für letztere Möglichkeit sollte das Element `devicename` genutzt werden.

Allerdings besteht nicht immer diese Wahlmöglichkeit. Einige Geräte, wie zum Beispiel Netzwerkkarten haben keinen Eintrag unter `/dev` oder werden anders dargestellt.

Beispiel 4.38. Gerätenamen per `devicename` auszeichnen

```
<para>Unter FreeBSD wird die serielle Datenübertragung über
  <devicename>sio</devicename> abgewickelt, das unterhalb von
  <filename>/dev</filename> eine Reihe von Einträgen anlegt.
  Zu diesen Einträgen gehören beispielsweise
  <filename>/dev/ttyd0</filename> und
  <filename>/dev/cuaa0</filename>.</para>

  <para>Andererseits erscheinen Geräte wie beispielsweise
    <devicename>ed0</devicename> nicht unterhalb von
    <filename>/dev</filename>.</para>

  <para>Unter MS-DOS wird das erste Diskettenlaufwerk als
    <devicename>a:</devicename> bezeichnet.  FreeBSD
     bezeichnet es als <filename>/dev/fd0</filename>.</para>
```

Darstellung:

Unter FreeBSD wird die serielle Datenübertragung über `sio` abgewickelt, das unterhalb von `/dev` eine Reihe von Einträgen anlegt. Zu diesen Einträgen gehören beispielsweise `/dev/ttyd0` und `/dev/cuaa0`.

Andererseits erscheinen Geräte wie beispielsweise `ed0` nicht unterhalb von `/dev`.

Unter MS-DOS wird das erste Diskettelaufwerk als `a:` bezeichnet. FreeBSD bezeichnet es als `/dev/fd0`.

4.2.5.8. Rechner, Domains, IP-Adressen und mehr

FreeBSD-Erweiterung

Die hier genannten Elemente sind Bestandteil der FreeBSD-Erweiterung für DocBook und sind nicht in der originalen DocBook DTD enthalten.

Bezeichner für Rechner können in Abhängigkeit der Bezeichnungsweise auf verschiedene Art und Weise ausgezeichnet werden. Gemeinsam ist allen, dass sie das Element `hostid` benutzen. Über das Attribut `role` wird die Art des Bezeichners genauer bestimmt.

Kein Rollenattribut oder `role="hostname"`
: Ohne Rollenattribut stellt der umschlossene Text einen normalen Rechnernamen wie `freefall` oder `wcarchive` dar. Wenn es gewünscht ist, kann mittels `role="hostname"` explizit angegeben werden, dass es sich um einen Rechnernamen handelt.

`role="domainname"`
: Ein Domainname wie `FreeBSD.org` oder `ngo.org.uk`. Er enthält keinen Rechnernamen.

`role="fqdn"`
: Vollqualifizierter Domainname wie `www.FreeBSD.org`. Enthält sowohl einen Domainnamen als auch einen Rechnernamen.

`role="ipaddr"`
: Eine IP-Adresse, meistens als durch Doppelpunkte getrenntes Tupel von vier Zahlen dargestellt.

`role="ip6addr"`
: Eine IPv6-Adresse.

`role="netmask"`
: Eine Netzwerkmaske, dargestellt als ein durch Doppelpunkte getrenntes Vierzahlentupel, einer Hexzahl oder als ein `/`, dem eine Zahl folgt.

`role="mac"`
: Eine MAC-Adresse, dargestellt durch zweistellige Hexzahlen, die durch Doppelpunkte getrennt sind.

Beispiel 4.39. `role` und `hostid`

```
<para>Der lokale Rechner kann immer über den Namen
  <hostid>localhost</hostid>  angesprochen werden, dem immer
  die IP-Adresse
  <hostid role="ipaddr">127.0.0.1</hostid> zugeordnet ist.</
para>

  <para>Zur Domain <hostid role="domainname">FreeBSD.org</
hostid>
    gehören verschiedene Rechner, inklusive <hostid
    role="fqdn">freefall.FreeBSD.org</hostid> und <hostid
    role="fqdn">pointyhat.FreeBSD.org</hostid>.</para>
```

```
  <para>Wenn eine IP-Adresse einer Netzwerkkarte zugeordnet ↺
wird,
    was mit der Hilfe von <command>ifconfig</command> ↺
geschieht,
    sollte <emphasis>immer</emphasis> die Netzmaske
    <hostid role="netmask">255.255.255.255</hostid>, die auch
    hexadezimal als <hostid role="netmask">0xffffffff</hostid>
    abgegeben werden kann, benutzt werden.</para>

  <para>Die MAC-Adresse ist für jede existierende Netzwerkkarte
    auf der Welt eindeutig. Eine typische MAC-Adresse ist
    beispielsweise <hostid
    role="mac">08:00:20:87:ef:d0</hostid>.</para>
```

Darstellung:

Der lokale Rechner kann immer über den Namen `localhost` angesprochen werden, dem immer die IP-Adresse `127.0.0.1` zugeordnet ist.

Zur Domain `FreeBSD.org` gehören verschieden Rechner, inklusive `freefall.FreeBSD.org` und `bento.FreeBSD.org` .

Wenn eine IP-Adresse einer Netzwerkkarte zugeordnet wird, was mit der Hilfe von `ifconfig` geschieht, sollte *immer* die Netzmaske `255.255.255.255` , die auch hexadezimal als `0xffffffff` abgegeben werden kann, benutzt werden.

Die MAC-Adresse ist für jede existierende Netzwerkkarte auf der Welt eindeutig. Eine typische MAC-Adresse ist beispielsweise `08:00:20:87:ef:d0` .

4.2.5.9. Benutzernamen

FreeBSD-Erweiterung

Die hier genannten Elemente sind Bestandteil der FreeBSD-Erweiterung für DocBook und sind nicht in der originalen DocBook DTD enthalten.

Namen von Benutzern, wie `root` oder `bib`, können mit dem Element `username` ausgezeichnet werden.

Beispiel 4.40. Das Element `username`

```
<para>Für die meisten Administrationsaufgaben müssen
```

```
    Sie als <username>root</username> angemeldet sein.</para>
```

Darstellung:

Für die meisten Administrationsaufgaben müssen Sie als `root` angemeldet sein.

4.2.5.10. Beschreibung von `Makefiles`

FreeBSD-Erweiterung

Die hier genannten Elemente sind Bestandteil der FreeBSD-Erweiterung für DocBook und sind nicht in der originalen DocBook DTD enthalten.

Zur Beschreibung von Teilen einer Makedatei stehen die beiden Elemente `marketarget` und `makevar` zur Verfügung. `maketarget` bezeichnet ein Ziel eines `Makefile`s, das als Parameter beim Aufruf von `make` angegeben werden kann. `makevar` hingegen bezeichnet eine Variable, die entweder in einem `Makefile` definiert oder `make` auf der Befehlszeile übergeben werden kann, um so den Bauprozess zu beeinflussen.

Beispiel 4.41. **`maketarget`** und **`makevar`**

```
<para>Zwei übliche Ziele in einem <filename>Makefile</filename>
  sind <maketarget>all</maketarget> und
  <maketarget>clean</maketarget>.</para>

<para>Üblicherweise wird, wenn das Ziel <maketarget>all</
maketarget>
  aufgerufen wird, die gesamte Anwendung neu erstellt. Der ↵
Aufruf
  des Zieles <maketarget>clean</maketarget> veranlasst das
  Löschen aller temporären Dateien (zum Beispiel
  <filename>.o</filename>), die während der Übersetzung erzeugt
  wurden.</para>

<para>Das genaue Verhalten von <maketarget>clean</maketarget>
  kann von einer Reihe von Variablen beeinflusst werden.
  Stellvertretend seien hier <makevar>CLOBBER</makevar> und
  <makevar>RECURSE</makevar> genannt.</para>
```

Darstellung:

Zwei übliche Ziele in einem `Makefile` sind `all` und `clean`.

Üblicherweise wird, wenn das Ziel `all` aufgerufen wird, die gesamte Anwendung neu erstellt. Der Aufruf des Zieles `clean` veranlaßt das Löschen aller temporären Dateien (zum Beispiel `.o`), die während der Übersetzung erzeugt wurden.

Das genaue Verhalten von `clean` kann von einer Reihe von Variablen beeinflusst werden. Stellvertretend seien hier `CLOBBER` und `RECURSE` genannt.

4.2.5.11. Text buchstabengetreu übernehmen

Für das Handbuch ist es oft notwendig, Textausschnitte buchstabengetreu darzustellen. Hierbei kann es sich um Texte handeln, die aus einer anderen Datei stammen oder die der Leser eins-zu-eins aus dem Handbuch kopieren können soll.

In einigen Fällen ist zu diesem Zwecke `programlisting` ausreichend, jedoch nicht immer. So ist `programlisting` zum Beispiel nicht einsetzbar, wenn es darum geht, einen Auszug aus einer Datei innerhalb eines Absatzes einzufügen. In solchen Fällen sollte das Element `literal` zum Einsatz kommen.

Beispiel 4.42. **`literal`**

```
<para>Die Zeile <literal>maxusers 10</literal> in der
  Kernelkonfigurationsdatei beeinflußt die Größe vieler
  Systemtabellen und kann als ungefähr als Richtwert dafür
  gelten, wie viele parallele Anmeldungen das System handhaben
  kann.</para>
```

Darstellung:

Die Zeile `maxusers 10` in der Kernelkonfigurationsdatei beeinflußt die Größe vieler Systemtabellen und kann als ungefähr als Richtwert dafür gelten, wie viele paralle Anmeldungen das System handhaben kann.

4.2.5.12. Benutzerspezifische Eingaben darstellen

Es kommt oft vor, dass der Leser Beispiele, Dateinamen oder Kommandozeilen verändern muss. Für einen solchen Anwendungsfall ist das Element `replaceable` gedacht. Es kann *innerhalb* von anderen Elementen genutzt werden, um die Teile auszuzeichnen, die es zu ersetzen gilt.

Beispiel 4.43. Das Element `replaceable`

```
<screen>&prompt.user; <userinput>man <replaceable>command</
replaceable></userinput></screen>
```

Darstellung:

```
% man command
```

Dieses Beispiel zeigt, dass nur der Text mit `replaceable` umschlossen werden soll, den der Benutzer einzusetzen hat. Sämtlicher anderer Text sollte wie üblich ausgezeichnet werden.

```
<para>Die Zeile
  <literal>maxusers <replaceable>n</replaceable></literal>
  in der Kernelkonfigurationsdatei bestimmt die Größe vieler ↺
Systemtabellen
  und stellt einen groben Richtwert dafür dar, wie viele ↺
gleichzeitige Anmeldungen
  das System unterstützt.</para>

<para>Für einen Arbeitsplatzrechner stellt <literal>32</
literal> einen guten
  Wert für <replaceable>n</replaceable> dar.</para>
```

Darstellung:

Die Zeile `maxusers n` in der Kernelkonfigurationsdatei bestimmt die Größe vieler Systemtabellen und stellt einen groben Richtwert dafür dar, wie viele gleichzeitige Anmeldungen das System unterstützt.

Für einen Arbeitsplatzrechner stellt `32` einen guten Wert für *n* dar.

4.2.5.13. Fehlermeldungen des Systems darstellen

In manchen Fällen kann es nötig sein, Fehlermeldungen darzustellen, die von FreeBSD erzeugt werden können. Für solche Fälle ist das Element `errorname` vorgesehen.

Beispiel 4.44. Das Element `errorname`

```
 <screen><errorname>Panic: cannot mount root</errorname></
screen>
```

Darstellung:

```
Panic: cannot mount root
```

4.2.6. Bilder und Grafiken

Wichtig

Die Verwendung von Grafiken innerhalb der Dokumentation ist momentan noch in einem experimentellen Stadium. Es ist daher wahrscheinlich, dass sich die hier beschriebenen Mechanismen noch ändern werden.

Für die Verwendung von Grafiken ist es notwendig, den Port graphics/ImageMagick zusätzlich zu installieren, da er *nicht* vom Port textproc/docproj mitinstalliert wird.

Das beste Beispiel für den Einsatz von Grafiken ist der unter `doc/en_US.ISO8859-1/articles/vm-design/` zu findene Artikel „Design elements of the FreeBSD VM system“. Falls beim Lesen der folgenden Kapitel Fragen unbeantwortet oder unklar bleiben, empfiehlt es sich, die unter dem genannten Verzeichnis befindlichen Dateien zu studieren und anhand ihrer zu verstehen, wie alles zusammenhängt. Es empfiehlt sich, den Artikel in verschiedenen Ausgabeformaten zu erzeugen, da man so sehen kann, wie die Grafiken in Abhängigkeit vom Ausgabemedium angeordnet werden.

4.2.6.1. Unterstütze Grafikformate

Zur Zeit werden nur zwei Grafikformate unterstützt. Welches von beiden Formaten zum Einsatz kommen sollte, hängt von der Art der Grafik ab.

Für Bilder, die vorrangig Vektorelemente wie Netzwerkdiagramme, Zeitlinien und ähnliches beinhalten, sollte Encapsulated Postscript als Format gewählt werden. Wichtig ist es in diesem Fall, dass die Grafikdatei die Endung `.eps` hat. Für Bitmapgrafiken, wie zum Beispiel Bildschirmfotos, steht das Portable Network Grafic Format zur Verfügung. In diesem Fall, sollte die Grafikdatei immer die Endung `.png` haben.

In das Subversion-Repository sollten *nur* Grafiken in diesen beiden Formaten übernommen werden.

Es sollte darauf sehr darauf geachtet werden, das richtige Format für das richtige Bild zu wählen. Erwartungsgemäß wird es Dokumente geben, die eine Mischung aus PNG- und

EPS-Grafiken enthalten. In solchen Fällen, stellen die Makedateien die Verwendung des richtigen Formats in Abhängigkeit vom Ausgabeformat sicher. *Deshalb sollte die gleiche Grafik niemals in zwei unterschiedlichen Formaten in das CVS-Archiv übernommen werden.*

Wichtig

Es ist absehbar, dass das Dokumentationsprojekt in Zukunft das Scalable Vector Graphic-Format (SVG) als Standardformat für Vektorgrafiken übernehmen wird. Zum jetzigen Zeitpunkt ist dieser Wechsel noch nicht möglich, da der Stand der jetzigen SVG-Anwendungen noch nicht den dafür notwendigen Erfordernissen entspricht.

4.2.6.2. DocBook-Elemente für den Grafikeinsatz

Das Auszeichnen von Bildern mittels DocBook ist relativ einfach. Zuerst wird ein `mediaobject`-Element eingefügt, das als Container für medienspezifische Elemente fungieren kann. Für die Zwecke des FDPs sind das die Elemente `imageobject` und `textobject`.

In das `mediaobject`-Element sollten ein Element vom Typ `imageobject` und zwei `textobject`-Elemente eingefügt werden. Das `imageobject`-Element verweist auf die eigentliche Grafikdatei. Dabei ist allerdings nur der Dateipfad ohne Erweiterung anzugegeben. Die `textobject`-Elemente werden dafür genutzt, Texte aufzunehmen, die dem Leser anstelle des Bildes oder zusammen mit dem Bild angezeigt werden.

Dies kann unter zwei Umständen geschehen:

- Wenn ein Dokument als HTML-Datei durch einem Browser angezeigt wird. In diesem Falle muss jeder Grafik ein Alternativtext zugeordnet werden, der dem Leser angezeigt werden kann. Meist ist das notwendig, wenn der Browser die Grafik noch nicht geladen hat oder wenn der Benutzer den Mauszeiger über die Grafik führt.

- Wenn das Dokument als Textdatei gelesen wird. Da in einer Textdatei keine Grafiken angezeigt werden können, sollte es für die Grafik eine Textentsprechung geben, die alternativ angezeigt werden kann.

Das folgende Beispiel soll das bisher geschriebene illustrieren. Angenommen es liegt eine einzubindende Grafik in der Datei `bild1.png` vor, die die Darstellung eines As in einem Rechteck enthält. Die ASCII-Alternative könnte so ausgezeichnet werden:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="bild1"> ❶
  </imageobject>
```

```
  <textobject>
    <literallayout class="monospaced">+ - - - - - - - - - - - - - - -
+ ❷
|       A       |
+- - - - - - - - - - - - - -+</literallayout>
  </textobject>
  <textobject>
    <phrase>Ein Bild</phrase> ❸
  </textobject>
</mediaobject>
```

❶ Innerhalb vom Element `imageobject` befindet sich ein Element `imagedata`, welches mit Hilfe des Attributes `fileref` den Namen der Grafikdatei (ohne Erweiterung) angibt. Die Bestimmung der Dateierweiterung wird von den Stylesheets übernommen.

❷ Das erste `textobject`-Element enthält ein `literallayout`-Element, dessen Attribut `class` den Wert `monospaced` zugewiesen bekommt. Der Inhalt dieses Elements wird genutzt, wenn das Dokument in Textform ausgegeben wird. An dieser Stelle hat der Autor die Möglichkeit seine „Textzeichenkünste“ unter Beweis zu stellen.

Wichtig ist, dass die erste und die letzte Zeile sich gleichauf mit dem öffnenden und dem schließenden Tag befindet. Dadurch wird sichergestellt, dass keine unnötigen Leerzeichen in die Ausgabe aufgenommen werden.

❸ Das zweite `textobject`-Element sollte lediglich ein `phrase`-Element enthalten. Wird das Dokument nach HTML konvertiert, wird dessen Inhalt für das Attribut `alt` des `img`-Tags verwendet.

4.2.6.3. Die `Makefile`-Einträge

Alle in einem Dokument verwendeten Grafiken müssen in dem zugehörigen Makefile in der Variable `IMAGES` enthalten sein. `IMAGES` sollte immer die Namen der *Quellgrafiken* enthalten. Werden in einem Dokument beispielsweise die drei Grafiken `bild1.eps` , `bild2.png` und `bild3.png` referenziert, sollte das `Makefile` die folgende Zeile enthalten:

```
…
IMAGES= bild1.eps bild2.png bild3.png
…
```

Eine andere Möglichkeit wäre:

```
…
IMAGES=  bild1.eps
IMAGES+= bild2.png
IMAGES+= bild3.png
…
```

Es kann nicht oft genug betont werden: Welche Grafikdateien für das zu erzeugende Dokument benötigt werden, wird von dem Makefiles bestimmt. `IMAGES` darf nur die Originaldateien enthalten.

4.2.6.4. Grafiken und Kapitel in Unterverzeichnissen

Wenn Sie Ihre Dokumentation in mehrere kleine Dateien aufspalten (siehe Abschnitt 3.7.1, „Dateien mit Allgemeinen Entitäten einbinden“), müssen Sie sorgfältig vorgehen.

Angenommen es handelt sich um ein Buch, dessen drei Kapitel in separaten Verzeichnissen angelegt wurden (`kapitel1/kapitel.xml` , `kapitel2/kapitel.xml` und `kapitel3/kapitel.xml`). Enthalten die Kapitel Grafiken, empfiehlt es sich, diese in den gleichen Verzeichnisses abzulegen, wie die Kapitel selbst. In diesem Falle gilt es jedoch zu beachten, dass die Pfade der Grafikdateien in der Variable `IMAGES` und in den `imagedata`-Elementen immer auch den Verzeichnisnamen enthalten.

Soll beispielsweise die Datei `kapitel1/bild1.png` in das in `kapitel1/kapitel.xml` enthaltene Kapitel eingebunden werden, sollte dies so erfolgen:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="kapitel1/bild1"> ❶
  </imageobject>

  …
</mediaobject>
```

❶ `fileref` muss den Datei- und den Verzeichnisnamen enthalten.

Das `Makefile` muss dementsprechend die Zeile

```
…
IMAGES=  kapitel1/bild1.png
…
```

enthalten.

Wird dies beachtet, sollte es zu keinen Problemen kommen.

4.2.7. Querverweise

Anmerkung

Querverweise sind auch Flußelemente.

4.2.7.1. Querverweise innerhalb eines Dokumentes

Um innerhalb eines Dokumentes Verweise anzulegen, muss angegeben werden, von welcher Textstelle aus wohin verwiesen werden soll.

Jedes DocBook-Element besitzt ein Attribut `id`, über das seinem Element ein eindeutiger Bezeichner zugewiesen werden kann.

In den meisten Fällen werden Querverweise nur zu Kapiteln gesetzt. Die `chaper`- und `sect*`-Elemente sollten aus diesem Grunde ein gesetztes `id`-Attribut besitzen.

Beispiel 4.45. `chapter` und `section` mit dem Attribut `id`

```
<chapter id="kapitel1">
  <title>Einführung</title>

  <para>Das ist eine Einführung. Sie enthält ein ↺
Unterkapitel, das
    ebenfalls einen eigenen Bezeichner hat.</para>

  <sect1 id="kapitel1-unterkapitel1">
    <title>Unterkapitel 1</title>

    <para>Das ist ein Unterkapitel.</para>
  </sect1>
</chapter>
```

Als Wert für das Attribut `id` sollte immer ein selbsterklärender Bezeichner gewählt werden. Zudem ist es notwendig, dass dieser Bezeichner innerhalb des Dokumentes eindeutig ist. Im obigen Beispiel wurde der Bezeichner für das Unterkapitel gebildet, indem der Bezeichner des übergeordneten Kapitels um den Titel des Unterkapitels erweitert wurde. Diese Vorgehensweise hilft sicherzustellen, dass Bezeichner eindeutig sind und bleiben.

Manchmal soll jedoch nicht auf den Anfang eines Kapitels verwiesen werden, sondern zum Beispiel auf eine Stelle in einem Absatz oder auf ein bestimmtes Beispiel. In solchen Fällen kann an der Stelle, auf die verwiesen werden soll, das Element `anchor` mit gesetztem Attribut `id` eingefügt werden. `anchor` kann selber keinen weiteren Inhalt aufnehmen.

Beispiel 4.46. Querverweise und das Element `anchor`

```
<para>Dieser Absatz enthält ein
  <anchor id="absatz1"/>Ziel für Querverweise, was jedoch keine
  Auswirkung auf dessen Darstellung hat.</para>
```

Zum Anlegen des eigentlichen Querverweises selbst kann eines der beiden Elemente `xref` oder `link` genutzt werden. Beide besitzen das Attribut `linkend`, dem der `id`-Wert des Verweiszieles zugewiesen wird. Ob sich das Ziel vor oder nach dem Verweis befindet, spielt keine Rolle.

`xref` und `link` unterscheiden sich jedoch hinsichtlich der Art und Weise, auf die der Text erzeugt wird, auf dem der Querverweis liegt. Kommt `xref` zum Einsatz, hat der Autor keine Kontrolle darüber – der Text wird automatisch für ihn erzeugt.

Beispiel 4.47. Einsatz von `xref`

Für dieses Beispiel wird davon ausgegangen, dass sich folgendes Textfragment irgendwo innerhalb eines Dokumentes auftaucht, dass das vorherige `id`-Beispiel enthält.

```
<para>Weitere Informationen gibt
  es im <xref linkend="kapitel1"/>.</para>

<para>Genauere Informationen können im
  <xref linkend="kapitel1-unterkapitel1"/> gefunden werden.</
para>
```

Der Verweistext wird automatisch von den Stylesheets erzeugt und so hervorgehoben, dass ersichtlich ist, dass es sich bei dem Text um einen Verweis handelt.

> Weitere Informationen können in der *Einführung* gefunden werden.
>
> Genauere Informationen können im *Unterkapitel 1* gefunden werden.

Der Text, auf dem der HTML-Link für den Querverweis liegt, wurde von den Kapitelüberschriften übernommen.

Anmerkung

Das bedeutet, dass es *nicht möglich* ist, mit der Hilfe von `xref` einen Querverweis zu einer mit `anchor` gekennzeichneten Stelle anzulegen. Da `anchor` keinen Inhalt aufnehmen kann, können die Stylesheets nicht automatisch einen Text für den Verweis erzeugen.

Möchte man selber den für den Verweis benutzten Text bestimmen können, sollte das Element `link` verwendet werden. Im Gegensatz zu `xref` kann `link` Inhalt aufnehmen, der dann für den Verweis verwendet wird.

Beispiel 4.48. `link` beutzen

Für dieses Beispiel wird davon ausgegangen, dass es sich in dem Dokument befindet, das auch das `id`-Beispiel enthält.

```
<para>Weitere Informationen können
  im <link linkend="kapitel1">ersten Kapitel</link> gefunden
  werden.</para>

<para>Genauere Informationen können in
  <link linkend="kapitel1-unterkapitel1">diesem</link> Kapitel
  gefunden werden.</para>
```

Aus diesem SGML-Fragment würden die Stylesheets folgendes generieren (der hervorgehobene Text deutet den erzeugten Verweis an):

> Weitere Informationen können im *ersten Kapitel* gefunden werden.
>
> Genauere Informationen können in *diesem* Kapitel gefunden werden.

Anmerkung

Das letzte Beispiel ist schlecht. Es sollten niemals Wörter wie „dieses“ oder „hier“ als Linktext benutzt werden. Solche Wörter zwingen den Leser dazu, den Kontext des Verweises zu lesen, um zu verstehen, wohin der Verweis führt.

Anmerkung

Mit dem Element `link` *kann* auf mit `anchor` gekennzeichnete Stellen im Dokument verwiesen werden, da der Inhalt von `link` als Text für den Querverweise genutzt wird.

4.2.7.2. Verweise auf Dokumente im WWW

Das Anlegen von Verweisen auf externe Dokumente ist wesentlich einfacher – solange die URL des zu referenzierenden Dokumentes bekannt ist. Um von einem bestimmten Textabschnitt auf das gewünschte externe Dokument zu verweisen, muss die jeweilige Stelle mit dem Element `ulink` ausgezeichnet werden. Mittels des Attributes `url` kann die Adresse des Zieldokumentes angegeben werden. Bei der Umformung des Quelldokumentes in die verschiedenen Ausgabeformate wird der sich zwischen Start- und Endtag befindliche Text für den Verweis übernommen, den der Leser aufrufen kann.

Beispiel 4.49. Verweise mit `ulink`

```
<para>Natürlich ist es möglich, anstatt diesen Text
  weiterzulesen, sofort die
  <ulink url="&url.base;/de/index.html">FreeBSD-Homepage</
ulink>
  aufzurufen.<para>
```

Darstellung:

Natürlich ist es möglich, anstatt diesen Text weiterzulesen, sofort die FreeBSD-Homepage aufzurufen.

Kapitel 5. Stylesheets

Übersetzt von Johann Kois.

SGML legt nicht fest, wie ein Dokument am Monitor oder auf einem Ausdruck dargestellt werden soll. Für diese Aufgabe wurden spezielle Sprachen entwickelt, die Formatvorlagen (die sogenannten *Stylesheets*) für die Darstellung der Inhalte definieren. Zu diesen Sprachen gehören beispielsweise DynaText, Panorama, SPICE, JSSS, FOSI, CSS, DSSSL und andere mehr.

DocBook verwendet in DSSSL geschriebene Stylesheets. XHTML verwendet hingegen in CSS geschriebene Stylesheets.

5.1. DSSSL

Das Documentation Project verwendet eine anpasste Version der von Norm Walsh entwickelten modularen DocBook-Stylesheets, die über den Port textproc/dsssl-docbook-modular installiert werden können.

Die FreeBSD-Modifikationen sind hingegen nicht in der Ports-Sammlung enthalten, sondern befinden sich im Quellcode-Repository des Documentation Projects in der Datei `doc/share/xml/freebsd.dsl` . Diese Datei ist umfassend kommentiert und mit Beispielen versehen. Dadurch können Sie einfach nachvollziehen, wie die ursprünglichen Stylesheets vom FreeBSD Documentation Project angepasst wurden.

5.2. CSS

Cascading Stylesheets (CSS) erlauben es, Elementen eines XHTML-Dokuments Formatangaben (wie Schriftart, Größe, Schriftfarbe und andere mehr) zuzuweisen, ohne das XHTML-Dokument mit diesen Informationen zu überfrachten.

5.2.1. Die DocBook-Dokumente

The FreeBSD DSSSL-Stylesheets enthalten eine Referenz auf ein Stylesheet namens `docbook.css` , das sich im gleichen Verzeichnis wie die XHTML-Dateien befindet. Diese projektweite CSS-Datei wird automatisch von `doc/share/misc/docbook.css` kopiert und installiert, wenn DocBook-Dokumente nach XHTML konvertiert werden.

Kapitel 6. Verzeichnisstruktur unter doc/

Übersetzt von Johann Kois.

Der doc/-Baum ist auf eine besondere Weise organisiert. Dies gilt analog für die Dokumente, aus denen der FDP besteht. Das Ziel dieser Organisation ist es, das Hinzufügen neuer Dokumente zu erleichtern, sowie

1. die automatische Konvertierung der Dokumente in andere Formate einfach zu gestalten,

2. die Konsistenz zwischen den verschiedenen auf diese Weise organisierten Dokumenten sicherzustellen, was die parallele Bearbeitung verschiedener Dokumente vereinfacht, sowie

3. die Entscheidung, wo neue Dokumente innerhalb des Baumes platziert werden sollen, zu erleichtern.

Zusätzlich wird dadurch dem Umstand Rechnung getragen, dass die Dokumentation in verschiedenen Sprachen und Kodierungen vorhanden sein kann. Es ist von großer Bedeutung, dass die Struktur des Dokumentationsbaumes dabei dennoch einheitlich bleibt.

6.1. doc/ als höchste Ebene

Unterhalb von doc/ existieren zwei Arten von Verzeichnissen, die jeweils über spezifische Dateinamen und eine spezifische Bedeutung verfügen.
Verzeichnis: share/
Bedeutung: Enthält Dateien, die für alle Sprachen und Kodierungen der Dokumentation gültig sind. Es enthält weitere Unterverzeichnisse, um diese Informationen zu kategorisieren. So enthält share/mk beispielsweise die Dateien, die die make(1)-Infrastruktur bilden, während sich die für die SMGL-Unterstützung nötigen Dateien (darunter die FreeBSD DocBook DTD) unter share/xml befinden.
Verzeichnis: Sprache.Kodierung/
Bedeutung: Für jede verfügbare Sprache und Kodierung existiert ein eigenes Unterverzeichnis. Beispiele dafür sind en_US.ISO8859-1/ oder zh_TW.Big5/ . Zwar sind diese Verzeichnisnamen nicht gerade kurz, durch die vollständige Angabe von Sprache und Kodierung werden aber Probleme bei einer eventuellen Erweiterung der Dokumentation (etwa um eine zusätzliche Kodierung für eine bereits vorhandene Sprache) vermieden. Auch eine eventuelle Konvertierung der Dokumentation nach Unicode ist dadurch problemlos möglich.

6.2. Die Verzeichnisse Sprache.Kodierung/

Diese Verzeichnisse enthalten die eigentliche Dokumentation. Auf dieser Ebene erfolgt eine Unterteilung in drei Kategorien, die durch entsprechende Verzeichnisnamen gekennzeichnet werden.
Verzeichnis: articles
Inhalt: DocBook-formatierte Artikel (article) oder ähnliche Dokumente. Meist relativ kurz und in Abschnitte aufgeteilt. Artikel sind in der Regel als ein einziges, großes HTML-Dokument verfügbar.
Verzeichnis: books
Inhalt: DocBook-formatierte Bücher (book) oder ähnliche Dokumente. Umfangreiche Dokumente, die in Kapitel aufgeteilt werden. Sind in der Regel sowohl als eine einzige, große HTML-Datei (für Personen mit einer schnellen Internetanbindung oder für einen einfachen Druck über ein Browser) oder als eine Sammlung von vielen kleinen, miteinander verlinkten Dateien verfügbar.
Verzeichnis: man
Inhalt: Dient für Übersetzungen von Manualpages. Es enthält ein oder mehrere mann-Verzeichnisse, je nachdem, welche Abschnitte der Manualpages bereits übersetzt wurden.

Nicht jedes Sprache.Kodierung -Verzeichnis enthält all diese Unterverzeichnisse. Ob ein Verzeichnis vorhanden ist, hängt vielmehr davon ab, ob bereits ein entsprechender Teil der Dokumentation übersetzt wurde.

6.3. Dokumentenspezifische Informationen

Dieser Abschnitt enthält Informationen zu einigen vom FreeBSD Documentation Project (FDP) verwalteten Dokumenten.

6.3.1. Das Handbuch

books/handbook/

Das Handbuch wurde unter Verwendung der vom FreeBSD Project erweiterten DocBook-DTD geschrieben.

Das Handbuch ist als DocBook-book organisiert. Es besteht aus mehreren Teilen (parts), die wiederum mehrere Kapitel (chapter) enthalten können. Kapitel sind zusätzlich in Abschnitte (sect1) und Unterabschnitte (sect2, sect3 und so weiter) unterteilt.

6.3.1.1. Physikalische Organisation

Das Verzeichnis handbook enthält sowohl weitere Verzeichnisse als auch zahlreiche einzelne Dateien.

Anmerkung

Die Organisation des Handbuchs hat sich im Laufe der Zeit geändert, daher könnten die Informationen in diesem Abschnitt eventuell nicht mehr dem akutellen Stand entsprechen. Haben Sie Fragen zur Organisation des Handbuchs, so wenden Sie sich bitte an das FreeBSD documentation project.

6.3.1.1.1. Makefile

Das Makefile definiert verschiedene Variablen zur Konvertierung der XML-Quellen in andere Formate. Außerdem listet es die verschiedenen Dateien auf, aus denen das Handbuch gebaut wird. Zusätzlich wird die Standard-doc.project.mk inkludiert, die den für die Konvertierung in andere Formate notwendigen Code bereitstellt.

6.3.1.1.2. book.xml

Das Hauptdokument innerhalb des Handbuchs. Neben der DOCTYPE-Deklaration des Handbuchs werden hier auch die Elemente aufgelistet, die die Struktur des Handbuchs definieren.

book.xml verwendet Parameterentitäten, um Dateien mit der Endung .ent zu laden. Diese Dateien definieren die allgemeinen Entitäten, die innerhalb des Handbuchs verwendet werden.

6.3.1.1.3. Verzeichnis/chapter.xml

Jedes Kapitel des Handbuchs wird in einer chapter.xml genannten Datei gespeichert. Jedes Verzeichnis erhält den Namen des id-Attributs des chapter-Elements.

Enthält eine Kapiteldatei beispielsweise die Einträge

```
<chapter id="kernelconfig">
...
</chapter>
```

so handelt es sich um die Datei chapter.xml im Verzeichnis kernelconfig. Im Allgemeinen enthält diese Datei das komplette Kapitel.

Wird die XHTML-Version des Handbuchs gebaut, entsteht dadurch kernelconfig.html . Der Grund dafür ist allerdings der Wert des id-Attributs, und nicht der Name des Verzeichnisses.

In früheren Versionen des Handbuchs wurden all diese Dateien im gleichen Verzeichnis wie die Datei `book.xml` gespeichert und nach dem Wert des `id`-Attributs der `chapter`-Elemente benannt. Durch die Verwendung von eigenen Verzeichnissen für die verschiedenen Kapitel wurde das Handbuch für künftige Erweiterungen vorbereitet. Beispielsweise wurde es dadurch möglich, Bilder in die einzelnen Kapitel aufzunehmen. Die Bilder für das Handbuch werden zentral im Verzeichnis `share/images/books/handbook` gespeichert. Existiert eine lokalisierte Version eines Bildes, wird diese hingegen gemeinsam mit dem XML-Quellcode im gleichen Verzeichnis gespeichert. Ein Vorteil dieser Methode ist beispielsweise die Vermeidung von Namenskollisionen. Außerdem ist es übersichtlicher, mit mehreren Verzeichnissen zu arbeiten, die jeweils nur einige Dateien enthalten, als mit einem einzigen Verzeichnis, das eine Vielzahl von Dateien enthält.

Durch dieses Vorgehen entstanden viele Verzeichnisse, die jeweils eine `chapter.xml` enhalten, beispielsweise `basics/chapter.xml` , `introduction/chapter.xml` oder `printing/chapter.xml` .

Wichtig

Im Normalfall sollte ein Umstrukturierung des Handbuchs nicht dazu führen, dass dafür Dateien umbenannt werden müssen (es sei denn, einzelne Kapitel werden neu aufgenommen oder entfernt). Kapitel und Verzeichnisse sollten daher nicht nach ihrer Reihenfolge innerhalb des Handbuchs benannt werden, da sich diese Reihenfolge bei einer Umstrukturierung des Handbuchs ändern könnte.

Die Datei `chapter.xml` ist keine komplette XML-Datei, da unter anderem die Zeilen mit der DOCTYPE-Deklaration am Beginn der Datei nicht vorhanden sind.

Durch diesen Umstand ist es nicht möglich, einzelne Dateien direkt nach HTML, RTF, PS oder ein anderes Format zu konvertieren. Vielmehr muss dazu das *komplette* Handbuch neu gebaut werden.

Kapitel 7. Die Erzeugung der Zieldokumente

Übersetzt von Johann Kois.

Dieses Kapitels erklärt detailliert, *wie der Bau der Dokumentation organisiert ist* und *wie Sie diesen Prozess beeinflussen können*.

Nachdem Sie dieses Kapitel gelesen haben, werden Sie:

- Wissen, wie Sie (unter Verwendung der im Kapitel SGML-Werkzeuge beschriebenen Tools) die FDP-Dokumentation selbst bauen können.
- In der Lage sein, sowohl die make-Anweisungen der für jedes Dokument benötigten `Makefiles` als auch die Anweisungen der projektweiten Vorgaben der Datei `doc.project.mk` zu lesen und zu verstehen.
- Den Bau der Dokumentation über make-Variablen und make-Target anpassen können.

7.1. Für den Bau der FreeBSD-Dokumentation benötigte Werkzeuge

Zusätzlich zu den im Kapitel XML-Werkzeuge beschriebenen Werkzeugen benötigen Sie noch folgende Programme:

- Das wichtigste Werkzeug zum Bau der Dokumentation ist make, genauer Berkeley Make.
- Der Bau von Paketen erfolgt unter FreeBSD mit pkg_create. Wenn Sie ein anderes Betriebssystem als FreeBSD einsetzen, müssen Sie entweder ohne Pakete auskommen oder den Quellcode selbst kompilieren.
- gzip dient zur Erstellung komprimierter Versionen der Dokumentation. Unterstützt werden sowohl bzip2- als auch zip-Archive. Wollen Sie Pakete der Dokumentation erstellen, benötigen Sie auch noch tar.
- Mit install installieren Sie in der Standardeinstellung die Dokumentation auf Ihrem System. Es gibt aber auch alternative Wege, die Dokumentation zu installieren.

7.2. Die `Makefiles` des Dokumentationsbaums verstehen

Innerhalb des FreeBSD Documentation Projects gibt es drei verschiedene Arten von `Makefiles`:

- Ein Makefile in einem Unterverzeichnis gibt Anweisungen an dessen Dateien und Unterverzeichnisse weiter.
- Ein Dokument-Makefile beschreibt das Dokument, das aus dem Inhalt des jeweiligen Verzeichnisses gebaut werden soll.
- Make-Includes sind der "Klebstoff", der für den Bau der Dokumentation erforderlich ist. In der Regel heissen diese Dokumente doc.xxx.mk .

7.2.1. Unterverzeichnis-Makefiles

Derartige Makefiles sind in der Regel wie folgt aufgebaut:

```
SUBDIR =articles
SUBDIR+=books

COMPAT_SYMLINK = en

DOC_PREFIX?= ${.CURDIR}/..
.include "${DOC_PREFIX}/share/mk/doc.project.mk"
```

Die ersten vier nicht-leeren Zeilen definieren die make-Variablen SUBDIR, COMPAT_SYMLINK , und DOC_PREFIX .

Die erste SUBDIR-Anweisung weist (ebenso wie die COMPAT_SYMLINK -Anweisung) einer Variable einen Wert zu und überschreibt dabei deren ursprünglichen Wert.

Die zweite SUBDIR-Anweisung zeigt, wie man den aktuellen Wert einer Variable ergänzen kann. Nach der Ausführung dieser Anweisung hat die Variable SUBDIR den Wert articles books .

Die Anweisung DOC_PREFIX zeigt, wie man einer Variable einen Wert zuweist (vorausgesetzt, die Variable ist nicht bereits definiert). Eine derartige Anweisung ist beispielsweise sinnvoll, wenn sich DOC_PREFIX nicht dort befindet, wo es vom Makefile erwartet wird. Durch das Setzen dieser Variable kann der korrekte Wert an das Makefile übergeben werden.

Was heißt dies nun konkret? Mit den SUBDIR-Anweisungen legen Sie fest, welche Unterverzeichnisse beim Bau der Dokumentation eingeschlossen werden müssen.

COMPAT_SYMLINK wird zur Erstellung von symbolischen Links zwischen den jeweiligen Dokumentsprachen und deren offizieller Kodierung benötigt (so wird beispielsweise doc/en nach en_US.ISO-8859-1 verlinkt).

DOC_PREFIX gibt den Pfad zum Wurzelverzeichnis des Quellcode-Baums des FreeBSD Documentation Projects an. Diese Vorgabe kann jederzeit durch einen eigenen Wert ersetzt werden. Bei .CURDIR handelt es sich um eine in make eingebaute Variable, die den Pfad des aktuellen Verzeichnisses enthält.

Die letzte Zeile bindet `doc.project.mk` , die zentrale, projektweite make-Datei des FreeBSD Documentation Projects, in den Bau ein. Diese Datei enthält den "Klebstoff", der die diversen Variablen in Anweisungen zum Bau der Dokumentation konvertiert.

7.2.2. Dokument-`Makefiles`

Diese `Makefiles` definieren diverse make-Variablen mit Vorgaben zum Bau der im Verzeichnis enthaltenen Dokumentation.

Dazu ein Beispiel:

```
MAINTAINER=nik@FreeBSD.org

DOC?= book

FORMATS?= html-split html

INSTALL_COMPRESSED?= gz
INSTALL_ONLY_COMPRESSED?=

# SGML content
SRCS=  book.xml

DOC_PREFIX?= ${.CURDIR}/../../..

.include "$(DOC_PREFIX)/share/mk/docproj.docbook.mk"
```

Die Variable `MAINTAINER` ist von zentraler Bedeutung. Sie legt fest, wer für ein bestimmtes Dokument des FreeBSD Documentation Projects verantwortlich ist.

`DOC` (ohne die Erweiterung `.xml`) ist der Name des Hauptdokuments des Verzeichnisses, in dem sich das Makefile befindet. Mit `SRCS`-Anweisungen geben Sie alle Dokumente an, aus denen das Dokument besteht. Zusätzlich binden Sie damit wichtige Dateien ein, deren Änderung einen erneuten Bau der Dokumentation erforderlich macht.

Mit `FORMATS` geben Sie an, in welchen Formaten die Dokumentation gebaut werden soll. `INSTALL_COMPRESSED` enthält die Standardvorgaben, die beim Bau komprimierter Pakte der Dokumentation verwendet werden sollen. Der Variable `INSTALL_ONLY_COMPRESS` (die in der Voreinstellung leer ist) wird nur dann ein Wert zugewiesen, wenn ausschließlich komprimierte Pakete der Dokumentation erstellt werden sollen.

Anmerkung

Die Zuweisung von Werten an verschiedene Variablen wurde bereits im Abschnitt Unterverzeichnis-Makefiles behandelt.

Die Variable DOC_PREFIX und die verschiedenen Include-Anweisungen sollten Ihnen ebenfalls bereits vertraut sein.

7.3. Make-Includes des FreeBSD Documentation Projects

Diese Dateien lassen sich am besten verstehen, indem man sich deren Inhalt näher ansieht. Konkret handelt es sich dabei um folgende Dateien:

- doc.project.mk ist die Haupt-Include-Datei, die bei Bedarf alle folgenden Include-Dateien enthält.
- doc.subdir.mk sorgt dafür, dass alle benötigten Verzeichnisse (und Unterverzeichnisse) beim Bau der Dokumentation durchlaufen werden.
- doc.install.mk definiert Variablen, die die Installation der Dokumentation beeinflussen.
- doc.docbook.mk wird verwendet, wenn die Variable DOCFORMAT den Wert docbook hat und die Variable DOC gesetzt ist.

7.3.1. doc.project.mk

Diese Datei hat folgenden Aufbau:

```
DOCFORMAT?= docbook
MAINTAINER?= doc@FreeBSD.org

PREFIX?= /usr/local
PRI_LANG?= en_US.ISO8859-1

.if defined(DOC)
.if ${DOCFORMAT} == "docbook"
.include "doc.docbook.mk"
.endif
.endif

.include "doc.subdir.mk"
.include "doc.install.mk"
```

7.3.1.1. Variablen

DOCFORMAT und MAINTAINER enthalten Standardwerte, falls ihnen über das Dokument-Makefile keine anderen Werte zugewiesen werden.

Bei PREFIX handelt es sich um das Präfix, unter dem die zum Bau der Dokumentation erforderlichen SGML-Werkzeuge installiert sind. In der Regel handelt es sich dabei um /usr/local .

`PRI_LANG` sollte auf die Sprache und Kodierung eingestellt werden, die unter den Leser der Dokumentation am häufigsten verwendet wird. Diese Variable hat den Standardwert "US English".

Anmerkung

`PRI_LANG` beeinflusst in keinster Weise, welche Dokumente gebaut werden können oder sollen. Diese Variable wird lediglich dazu verwendet, häufig verwendete Dokumente in das Wurzelverzeichnis der installierten Dokumentation zu verlinken.

7.3.1.2. Bedingungen

Die Zeile `.if defined(DOC)` ist ein Beispiel für eine make-Bedingung, die (analog zum Einsatz in anderen Programmen) festlegt, was geschehen soll, wenn eine Bedingung "wahr" oder "falsch" ist. `defined` ist eine Funktion, die zurückgibt, ob die angegebene Variable existiert oder nicht.

`.if ${DOCFORMAT} == "docbook"` testet, ob die Variable `DOCFORMAT` den Wert `"docbook"` hat. Ist dies der Fall, wird `doc.docbook.mk` mit in den Bau aufgenommen.

Die zwei `.endif`s schließen die zwei weiter oben definierten Bedingungen.

7.3.2. `doc.subdir.mk`

Den Inhalt dieser Datei hier zu beschreiben, würde zu weit führen. Sie sollten aber nach dem Lesen der vorangegangenen Abschnitte und der folgenden Ausführungen in der Lage sein, Inhalt und Aufgabe dieser Datei zu verstehen.

7.3.2.1. Variablen

- `SUBDIR` legt die Unterverzeichnisse fest, deren Inhalt beim Bau der Dokumentation inkludiert werden muss.
- Mit `ROOT_SYMLINKS` wird der Name der Verzeichnisse angegeben, die von ihrer tatsächlichen Position aus in das Wurzelverzeichnis, unter dem die Dokumentation installiert wird, verlinkt werden sollen. Vorausgesetzt, bei der verwendeten Sprache handelt es sich um die primäre Sprache (die über `PRI_LANG` festgelegt wird).
- `COMPAT_SYMLINK` wird im Abschnitt Unterverzeichnis-Makefiles beschrieben.

7.3.2.2. Targets und Makros

Abhängigkeiten (*Dependencies*) werden folgendermaßen definiert: `target abhaengigkeit1 abhaengigkeit2 ...`. Um `target` zu bauen, müssen Sie zuvor die angegebenen Abhängigkeiten bauen.

Daran anschließend können Anweisungen zum Bau des angegebenen Targets folgen, falls der Konvertierungsprozess zwischen dem Target und seinen Abhängigkeiten nicht bereits früher definiert wurde oder falls die Konvertierung nicht der Standardkonvertierungsmethode entspricht.

Die spezielle Abhängigkeit `.USE` definiert das Äquivalent eines Makros.

```
_SUBDIRUSE: .USE
.for entry in ${SUBDIR}
 @${ECHO} "===> ${DIRPRFX}${entry}"
 @(cd ${.CURDIR}/${entry} && \
 ${MAKE} ${.TARGET:S/realpackage/package/:S/realinstall/install/} ↺
DIRPRFX=${DIRPRFX}${entry}/ )
.endfor
```

In diesem Beispiel kann `_SUBDIRUSE` nun als Makro, welches die angegebenen Befehle ausführt, verwendet werden, indem es im Makefile als Abhängigkeit angegeben wird.

Was unterscheidet dieses Makro nun von beliebigen anderen Targets? Der Hauptunterschied ist, dass es *nach* den Anweisungen der Bauprozedur, in der es als Abhängigkeit angegeben ist, ausgeführt wird. Außerdem ändert es die Variable `.TARGET` (die den Namen des aktuell gebauten Targets enthält) nicht.

```
clean: _SUBDIRUSE
 rm -f ${CLEANFILES}
```

In diesem Beispiel führt `clean` das Makro `_SUBDIRUSE` aus, nachdem es den Befehl `rm -f ${CLEANFILES}` erfolgreich ausgeführt hat. Dadurch löscht `clean` zwar beim Wechsel in ein neues *Unterverzeichnis* beim Bau erstellte Dateien, aber nicht beim Wechsel aus einem Unterverzeichnis in ein übergeordnetes Verzeichnis.

7.3.2.2.1. Vorhandene Targets

- `install` und `package` arbeiten nacheinander alle Unterverzeichnisse ab und rufen dabei jeweils ihre realen Versionen (`realinstall` beziehungsweise `realpackage`) auf.
- `clean` entfernt alle Dateien, die beim Bau der Dokumentation erzeugt wurden (dies sowohl im aktuellen Verzeichnis als auch in allen Unterverzeichnissen). `cleandir` hat die gleiche Aufgabe, würde aber zusätzlich die Objekt-Verzeichnisse löschen (falls diese existieren).

7.3.2.3. Weitere Bedingungen

- `exists` gibt "wahr" zurück, wenn die angegebene Datei bereits existiert.

- `empty` gibt "wahr" zurück, wenn die angegebene Variable leer ist.
- `target` gibt "wahr" zurück, wenn das angegebene Target noch nicht existiert.

7.3.2.4. Schleifenkonstrukte in `make (.for)`

`.for` erlaubt es, bestimmte Anweisungen für jedes Element einer Variable zu wiederholen, indem dieser Variable in jedem Durchlauf der Schleife das jeweilige Element der untersuchten Liste zugewiesen wird.

```
_SUBDIRUSE: .USE
.for entry in ${SUBDIR}
 @${ECHO} "===> ${DIRPRFX}${entry}"
 @(cd ${.CURDIR}/${entry} && \
 ${MAKE} ${.TARGET:S/realpackage/package/:S/realinstall/install/} ↺
DIRPRFX=${DIRPRFX}${entry}/ )
.endfor
```

Falls das Verzeichnis `SUBDIR` leer ist, würde in unserem Beispiel keine Aktion erfolgen. Enthält das Verzeichnis hingegen ein oder mehrere Elemente, werden die Anweisungen zwischen `.for` und `.endfor` für jedes Element ausgeführt, wobei `entry` durch das jeweilige Element ersetzt werden würde.

Kapitel 8. Die Webseite

Übersetzt von Johann Kois.

8.1. Vorbereitung

Sorgen Sie für genügend Plattenplatz (zwischen 200 und 500 MB). Der genaue Wert hängt davon ab, welche Methode Sie zum Bau der Webseiten verwenden. Dieser Platz wird von den SGML-Werkzeugen, den benötigten Teilen des Subversion-Baums, für temporären Speicher zum Bau der Seiten sowie für die Installation der Webseiten benötigt.

Anmerkung

Stellen Sie sicher, dass Ihre Dokumentationsports aktuell sind. Wenn Sie sich nicht sicher sind, entfernen Sie die alten Ports mit pkg_delete(1), bevor Sie die neue Version installieren. Derzeit wird unter anderem jade-1.2 vorausgesetzt. Haben Sie beispielsweise jade-1.1 installiert, deinstallieren Sie es mit:

```
# pkg_delete jade-1.1
```

8.1.1. svn verwenden

Um Dateien aus dem doc/ Subversion-Repository „auszuchecken“, muss svn installiert sein. Ist dies bei Ihnen noch nicht der Fall, können Sie dies entweder mit pkg_add(1) oder über die FreeBSD Ports-Sammlung nachholen:

```
# cd /usr/ports/devel/subversion
# make install clean
```

Um alle zum Bau der Webseite benötigten Quellen auszuchecken, führen Sie den folgenden Befehl aus:

```
# svn checkout https://svn0.us-
east.FreeBSD.org/doc/head/ /usr/build
```

svn0.us-east.FreeBSD.org ist ein öffentlicher Server. Wählen Sie einen Mirror in Ihrer Nähe und überprüfen Sie das Serverzertifikat aus der Liste Subversion mirror sites.

Tipp

Falls Sie `svn` nicht als Benutzer `root` ausführen, stellen Sie bitte zuvor sicher, dass Sie in das Verzeichnis `/usr/build` schreiben dürfen. Ist dies nicht möglich, so müssen Sie hier ein anderes Zielverzeichnis angeben, in das die Quellen der Webseite gespeichert werden sollen.

Nachdem `svn` seine Arbeit beendet hat, befindet sich die komplette FreeBSD-Webseite im Verzeichnis `/usr/build` (oder in dem von Ihnen angegebenen Verzeichnis). Haben Sie ein alternatives Verzeichis angegeben, müssen Sie `/usr/build` in den folgenden Ausführung durch Ihr gewähltes Zielverzeichnis ersetzen.

Das ist alles. Sie können nun mit dem Bau der Webseiten beginnen.

8.2. Die Webseiten bauen

Nachdem Sie die Quellen der Webseite erfolgreich heruntergeladen haben, können Sie mit dem Bau der Webseite beginnen. In unserem Beispiel erfolgt der Bau im Verzeichnis `/usr/build` , in dem sich bereits alle benötigten Dateien befinden.

1. Wechseln Sie in das Bau-Verzeichis.

```
# cd /usr/build
```

2. Sie starten den Bau der Webseiten, indem Sie in das Unterverzeichnis `en_US.ISO8859-1/htdocs` wechseln und dort den Befehl make(1) `all` ausführen.

```
# cd en_US.ISO8859-1/htdocs
# make all
```

8.3. Installieren der Webseiten auf Ihrem Server

1. Wechseln Sie wieder in das Verzeichnis `en_US.ISO8859-1/htdocs` , falls Sie dieses inzwischen verlassen haben.

```
# cd /usr/build/en_US.ISO8859-1/htdocs
```

2. Führen Sie make(1) `install` aus und setzen Sie die Variable `DESTDIR` auf das Verzeichnis, in das Sie die Webseiten installieren wollen. Die daraus resultierenden

Dateien werden unter `$DESTDIR/data` installiert, was als die document root ihres Webservers konfiguriert sein sollte.

```
# env DESTDIR=/usr/local/www make install
```

3. Wenn Sie die Webseiten bereits früher in dieses Verzeichnis installiert haben, wurden während der Installation keine veralteten Seiten entfernt. Wenn Sie die Webseiten beispielsweise täglich neu bauen und installieren, findet und entfernt der folgende Befehl alle Dateien, die in den letzten drei Tagen nicht aktualisiert wurden:

```
# find /usr/local/www -ctime 3 -print0 | xargs -0 rm
```

8.4. Umgebungsvariablen

`ENGLISH_ONLY`
: Ist diese Variable gesetzt und nicht leer, bauen und installieren die Makefiles ausschließlich die englischen Dokumente. Sämtliche Übersetzungen werden dabei ignoriert. Dazu ein Beispiel:

```
# make ENGLISH_ONLY=YES all install
```

Wenn Sie die Variable `ENGLISH_ONLY` deaktivieren und alle Webseiten inklusive aller Übersetzungen bauen wollen, setzen Sie die Variable `ENGLISH_ONLY` auf einen leeren Wert:

```
# make ENGLISH_ONLY="" all install clean
```

`WEB_ONLY`
: Ist diese Variable gesetzt und nicht leer, bauen und installieren die Makefiles nur die HTML-Seiten des Verzeichnisses `en_US.ISO8859-1/htdocs`. Alle Dokumente des Verzeichnisses `en_US.ISO8859-1` (Handbuch, FAQ, Artikel) werden dabei ignoriert:

```
# make WEB_ONLY=YES all install
```

`WEB_LANG`
: Ist diese Variable gesetzt, wird die Dokumentation nur für die durch diese Variable festgelegten Sprachen gebaut und im Verzeichnis `/usr/build` installiert. Alle weiteren Sprachen (ausgenommen Englisch) werden ignoriert. Dazu ein Beispiel:

```
# make WEB_LANG="el_GR.ISO8859-7 es_ES.ISO8859-1 hu_HU.ISO8859-2 ↺
nl_NL.ISO8859-1" all install
```

`NOPORTSCVS`
: Ist diese Variable gesetzt, checken die Makefiles keine Dateien aus dem Ports-CVS-Repository aus. Stattdessen werden die Dateien aus dem Verzeichnis `/usr/ports` (oder aus dem Verzeichnis, auf das die Variable `PORTSBASE` zeigt) verwendet.

`WEB_ONLY`, `ENGLISH_ONLY`, `WEB_LANG` und `NOPORTSCVS` sind Variablen für Makefiles. Diese werden entweder in `/etc/make.conf`, in `Makefile.inc` oder als Umgebungsvariablen auf der Kommandozeile oder in Ihrer Konfigurationsdatei gesetzt.

Kapitel 9. Übersetzungen

Übersetzt von Johann Kois.

Dieses Kapitel enthält die FAQ für die Übersetzung der FreeBSD Dokumentation (FAQ, Handbuch, Artikel, Manualpages und sonstige Dokumente) in andere Sprachen.

Es beruht *sehr* stark auf den Übersetzungs-FAQ des FreeBSD German Documentation Projects, die ursprünglich von Frank Gründer <elwood@mc5sys.in-berlin.de > geschrieben und danach von Bernd Warken <bwarken@mayn.de > ins Englische übersetzt wurden.

Diese FAQ wird vom Documentation Engineering Team <doceng@FreeBSD.org > gepflegt.

F: Warum eine FAQ?

A: Es melden sich immer mehr Leute auf der Mailingliste freebsd-doc, die Teile der FreeBSD Dokumentation in andere Sprachen übersetzen wollen. Diese FAQ soll die am häufigsten gestellten Fragen beantworten, damit möglichst rasch mit der Übersetzung begonnen werden kann.

F: Was bedeuten die Abkürzungen i18n und l10n?

A: i18n steht für *internationalization* (Internationalisierung), l10n für *localization* (Lokalisierung). Es handelt sich dabei um besser handhabbare Abkürzungen dieser Begriffe.

i18n kann als „i“, gefolgt von 18 Buchstaben, gefolgt von einem „n“, gelesen werden. Analog steht l10n für „l“, gefolgt von 10 Buchstaben, gefolgt von einem „n“.

F: Gibt es eigene Mailinglisten für Übersetzer?

A: Ja. Die verschiedenen Übersetzergruppen haben jeweils eigene Mailinglisten. Genauere Informationen finden Sie in der Liste der Übersetzungsprojekte. Diese Liste enthält die Mailinglisten und Internetseiten, die von den einzelnen Übersetzungsprojekten betrieben werden.

F: Werden noch Übersetzer benötigt?

A: Ja. Je mehr Leute an der Übersetzung arbeiten, desto schneller wird diese fertig, und umso schneller sind Änderungen im englischen Original auch in den übersetzten Dokumenten vorhanden.

Sie müsssen kein professioneller Dolmetscher sein, um dabei zu helfen.

F: Welche Sprachen muss ich dafür kennen/können?

A: Idealerweise haben Sie gute Kenntnisse in geschriebenem Englisch, außerdem sollten Sie natürlich fit in der Sprache sein, in die Sie übersetzen wollen.

Englisch ist allerdings nicht unbedingt nötig. Sie könnten beispielsweise auch die FAQ vom Spanischen ins Ungarische übersetzen.

F: Welche Software wird benötigt?

A: Es ist sehr empfehlenswert, eine lokale Kopie des FreeBSD Subversion-Repository (als Minimum den Dokumentationsteil) anzulegen. Dazu geben Sie den folgenden Befehl ein:

```
% svn checkout https://svn0.us-east.FreeBSD.org/doc/head/ head
```

Anmerkung

svn0.us-east.FreeBSD.org ist ein öffentlicher Server. Wählen Sie einen Mirror in Ihrer Nähe und überprüfen Sie das Serverzertifikat auf der Seite Subversion mirror sites.

Anmerkung

Damit dieser Befehl funktioniert, muss der Port devel/subversion installiert sein.

Sie sollten außerdem mit svn vertraut sein. Damit ist es möglich, festzustellen, was sich zwischen einzelnen Versionen eines Dokuments geändert hat.

Wenn Sie beispielsweise wissen wollen, was sich zwischen den Revisionen `r33733` und `r33734` der Datei `en_US.ISO8859-1/books/fdp-primer/book.xml` geändert hat, geben Sie den folgenden Befehl ein:

```
% svn diff -r33733:33734 en_US.ISO8859-1/books/fdp-primer/book.↺
xml
```

F: Wie finde ich heraus, ob noch jemand Teile der Dokumentation in die gleiche Sprache übersetzt?

A: Die Übersetzungsseite des Documentation Projects listet alle Übersetzungs-Teams auf, die derzeit aktiv sind. Arbeitet bereits jemand an der Übersetzung in Ihre Sprache, so kontaktieren Sie dieses Team, damit Dokumente nicht unnötigerweise mehrfach übersetzt werden.

Wenn Ihre Sprache nicht aufgeführt ist, senden Sie bitte eine E-Mail an das FreeBSD documentation project. Vielleicht denkt ja jemand über eine Übersetzung nach, hat sich aber noch nicht dafür entschieden.

F: Niemand übersetzt in meine Sprache. Was soll ich machen?

A: Gratulation, Sie haben gerade das „FreeBSD *Ihre-Sprache* Documentation Translation Project“ gestartet. Willkommen.

Entscheiden Sie zuerst, ob Sie die dafür nötige Zeit zur Verfügung haben. Da Sie als Einziger an der Übersetzung in Ihre Sprache arbeiten, sind Sie dafür verantwortlich, Ihre Arbeit zu veröffentlichen und die Arbeit von Freiwilligen, die Ihnen dabei helfen wollen, zu koordinieren.

Senden Sie eine E-Mail an die Mailingliste des Documentation Projects, in der Sie bekanntgeben, dass Sie an der Übersetzung der Dokumentation arbeiten, damit die Internetseiten aktualisiert werden können.

Gibt es in Ihrem Land einen FreeBSD-Spiegelserver, so sollten Sie den dafür Zuständigen kontaktieren und nachfragen, ob er Ihnen Speicherplatz oder E-Mailadressen für Ihr Projekt zur Verfügung stellen würde.

Danach wählen Sie ein Dokument aus und beginnen mit der Übersetzung. Am besten beginnen Sie mit kleineren Dateien, beispielsweise den FAQ oder einem der Artikel.

F: Ich habe ein Dokument übersetzt. Wo soll ich es hinschicken?

A: Das kommt darauf an. Wenn Sie bereits in einem Übersetzer-Team arbeiten (etwa dem japanischen oder dem deutschen Team), dann sollten Sie deren Richtlinien zum Umgang mit neuer Dokumentation folgen, die auf deren Internetseiten beschrieben werden.

Wenn Sie die einzige Person sind, die an der Übersetzung in eine Sprache arbeitet, oder wenn Sie für ein Übersetzungsprojekt verantwortlich sind, und Ihre Aktualisierungen an das FreeBSD Project übermitteln wollen, sollten Sie Ihre Übersetzungen dorthin senden (lesen Sie dazu auch die nächste Frage).

F: Ich arbeite als einziger an der Übersetzung in diese Sprache, wie versende ich meine Übersetzungen?

oder

Wir sind ein Übersetzer-Team, und wollen Dokumente versenden, die unsere Mitglieder übersetzt haben.

A: Stellen Sie zuerst sicher, dass Ihre Übersetzungen korrekt organisiert sind. Sie sollte sich also im existierenden Dokumentationsbaum befinden, und ohne Fehler bauen lassen.

Zurzeit wird die FreeBSD Dokumentation unterhalb des Verzeichnisses `head/` gespeichert. Die direkten Unterverzeichnisse werden entsprechend der Sprachkodierung benannt, in der sie geschrieben sind. Diese Kodierung nach ISO639 finden Sie auf einem FreeBSD-System unter `/usr/share/misc/iso639` , vorausgesetzt, das System wurde nach dem 20. Januar 1999 gebaut.

Wenn in Ihrer Sprache mehrere Kodierungen (wie dies etwa für Chinesisch der Fall ist) vorhanden sind, existiert für jede Kodierung ein eigenes Unterverzeichnis.

Zuletzt existieren auch noch Verzeichnisse für die einzelnen Dokumente.

Die Verzeichnishierarchie für eine hypothetische schwedische Übersetzung könnte etwa so aussehen:

```
head/
    sv_SE.ISO8859-1/
                      Makefile
                      htdocs/
                            docproj/
                      books/
                            faq/
                                Makefile
                                book.xml
```

Bei `sv_SE.ISO8859-1` handelt es sich um den Namen der Übersetzung in der `lang.encoding` Form. Beachten Sie auch, dass zum Bauen der Dokumentation zwei Makefiles notwendig sind.

Komprimieren Sie Ihre Übersetzungen mit tar(1) und gzip(1) und senden Sie sie an das FreeBSD Project.

```
% cd doc
% tar cf swedish-docs.tar sv_SE.ISO8859-1
% gzip -9 swedish-docs.tar
```

Legen Sie das Archiv `swedish-docs.tar.gz` irgendwo ab. Wenn Sie keinen eigenen Webspace haben (etwa weil Ihr Internetprovider Ihnen keinen zur Verfügung stellt), können Sie auch eine E-Mail an das Documentation Engineering Team <doceng@FreeBSD.org> schicken, um abzuklären, ob Sie die Datei auch als E-Mail schicken können.

In beiden Fällen sollten Sie mit send-pr(1) einen Bericht über den Versand der Dokumentation erstellen. Es ist sehr hilfreich, wenn Sie Ihre Übersetzung vorher korrekturlesen lassen und überprüfen, da es unwahrscheinlich ist, dass der Committer Ihre Sprache sehr gut beherrscht.

Danach wird jemand (meistens der Documentation Project Manager, derzeit ist dies das Documentation Engineering Team <doceng@FreeBSD.org>) überprüfen, ob sich Ihre Übersetzungen problemlos bauen lassen. Dabei wird besonders auf folgende Punkte geachtet:

1. Verwenden alle Dateien RCS-Strings (wie "ID")?

2. Arbeitet `make all` im Verzeichnis `sv_SE.ISO8859-1` korrekt?

3. Funktioniert `make install` ohne Probleme?

Gibt es dabei Probleme, so wird die Person, die Ihren Beitrag durchsieht, sich wieder an Sie wenden, damit Sie das Problem beheben.

Treten keine Probleme auf, wird Ihre Übersetzung so rasch als möglich committed.

F: Kann ich landes- oder sprachspezifische Informationen in meine Übersetzung aufnehmen?

A: Wir bitten Sie, dies nicht zu tun.

Nehmen wir an, dass Sie das Handbuch ins Koreanische übersetzen und einen Abschnitt mit Händlerinformationen in das Handbuch aufnehmen wollen.

Es gibt keinen Grund, warum diese Information nicht auch in der englischen (oder der deutschen, oder der spanischen, oder der japanischen oder der ...) Version vorhanden sein sollte. Es ist etwa denkbar, dass sich jemand mit englischer Muttersprache während eines Aufenthalts in Korea eine FreeBSD-Kopie kaufen möchte. Außerdem wird dadurch die weltweite Präsenz von FreeBSD verdeutlicht, was natürlich ebenfalls von Vorteil ist.

Wenn Sie also länderspezifische Informationen ergänzen wollen, sollten Sie dies zuerst in der englischen Version (mittels send-pr(1)) tun, und die Änderung anschließend in Ihre Sprache übersetzen.

Vielen Dank.

F: Wie lassen sich sprachspezifische Zeichen darstellen?

A: Nicht-ASCII-Zeichen innerhalb der Dokumentation werden durch SGML-Entities dargestellt.

Diese bestehen aus: Kaufmännischem Und (&), den Namen der Entity, und einem Strichpunkt (;).

Die Namen der Entities sind in ISO8879 definiert, die als Port textproc/iso8879 installiert werden kann.

Dazu einige Beispiele:
Entity: é
Darstellung: é
Beschreibung: Kleines „e“ mit (akutem) Akzent
Entity: É
Darstellung: É
Beschreibung: Großes „E“ mit (akutem) Akzent
Entity: ü
Darstellung: ü

Beschreibung: Kleines Umlaut-„u“

Nachdem Sie den iso8879-Port installiert haben, ist die vollständige Liste unter `/usr/local/share/xml/iso8879` vorhanden.

F: Wie spricht man den Leser an?

A: In englischen Dokumenten wird der Leser mit „you“ angesprochen, es wird nicht zwischen formeller/informeller Anrede unterschieden, wie dies in manchen anderen Sprachen der Fall ist.

Wenn Sie in eine Sprache übersetzen, die diese Unterscheidung trifft, verwenden Sie die Form, die auch in den anderen technischen Dokumentationen dieser Sprache verwendet wird. Für deutsche Versionen ist dies die dritte Person Plural („Sie“).

F: Muss ich zusätzliche Informationen in meine Übersetzungen einbauen?

A: Ja.

Der Header der englischen Version jedes Textes sieht in etwa so aus:

```
<!--
     The FreeBSD Documentation Project

     $FreeBSD: head/en_US.ISO8859-1/books/faq/book.xml 38674 ↺
2012-04-14 13:52:52Z $
```

Das exakte Aussehen kann unterschiedlich sein, die Zeile mit $FreeBSD$ sowie der Ausdruck `The FreeBSD Documentation Project` sind allerdings immer enthalten. Beachten Sie, dass die Zeile mit $FreeBSD von Subversion automatisch expandiert wird, daher sollte an dieser Stelle in Ihren neuen Dokumenten nur `$FreeBSD$` stehen.

Ihre übersetzten Dokumente sollten eine eigene $FreeBSD$-Zeile enthalten. Zusätzlich sollten Sie die Zeile mit `The FreeBSD Documentation Project` in `The FreeBSD Ihre-Sprache Documentation Project` ändern.

Außerdem sollten Sie eine weitere Zeile einfügen, die festlegt, auf welcher Version des englischen Originals Ihre Übersetzung basiert.

Die spanische Version dieser Datei könnte etwa so beginnen:

```
<!--
     The FreeBSD Spanish Documentation Project

     $FreeBSD: head/es_ES.ISO8859-1/books/faq/book.xml 38826 ↺
2012-05-17 19:12:14Z hrs $
     Original revision: r38674
-->
```

Kapitel 10. Der Schreibstil

Übersetzt von Johann Kois.

Damit von verschiedenen Autoren geschriebene Dokumente zueinander konsistent bleiben, gibt es einige Richtlinien, denen Autoren bei der Erstellung ihrer Dokumente folgen müssen.

Verwendung von amerikanischem Englisch
: Es gibt mehrere englische Varianten und damit verbunden verschiedene Schreibweisen für das gleiche Wort. Wo dies der Fall ist, ist die amerikanische Schreibweise zu verwenden. Man schreibt daher „color“ statt „colour“, „rationalize“ statt „rationalise“, und so weiter.

Anmerkung

Die Verwendung von Britischem Englisch ist akzeptabel, wenn es sich um einen neuen Beitrag handelt, solange die gesamte Schreibweise eines Dokuments einheitlich bleibt. Alle anderen Dokumente wie Bücher, Internetseiten, Manualpages und andere müssen allerdings amerikanisches Englisch verwenden.

Vermeiden von Zusammenziehungen
: Verwenden Sie keine Zusammenziehungen, sondern schreiben Sie die Phrase immer aus. Die Schreibweise „Don't use contractions.“ wäre also nicht korrekt.

  Die Vermeidung von Zusammenziehungen sorgt für einen etwas formelleren Ton, ist präziser und erleichtert die Arbeit der Übersetzer.

Nutzung von Kommas bei Aufzählungen
: Bei einer Aufzählung innerhalb eines Absatzes sollten Sie zwischen den einzelnen Begriffen Kommas setzen. Zwischen dem letzten und vorletzten Begriff setzen Sie ein Komma und das Wort „und“.

  Dazu ein Beispiel:

  > Das ist eine Liste von ein, zwei und drei Dingen.

  Handelt es sich dabei um eine Liste von drei Begriffen, „ein“, „zwei“, und „drei“, oder um eine Liste von zwei Begriffen, „ein“ und „zwei und drei“?

  Es ist daher besser, explizit ein serielles Komma zu setzen:

  > Das ist eine Liste von ein, zwei, und drei Dingen.

Vermeidung von redundanten Begriffen
: Versuchen Sie, keine redundanten Phrasen zu verwenden. Dies gilt insbesondere für die Ausdrücke „der Befehl“, „die Datei“, und „man command“.

  Die folgenden zwei Beispiele veranschaulichen dies für Befehle. Bevorzugt wird die Schreibweise des zweiten Beispiels.

  Verwenden Sie den Befehl `svn`, um Ihre Quellen zu aktualisieren.

  Verwenden Sie `svn`, um Ihre Quellen zu aktualisieren.

  Analoges gilt für Dateinamen, wobei wiederum die zweite Schreibweise bevorzugt wird.

  … in der Datei `/etc/rc.local` …

  … in `/etc/rc.local` …

  Auch für Manualpages gibt es zwei Schreibweisen. Auch hier wird die zweite Schreibweise bevorzugt (das zweite Beispiel nutzt das Tag `citerefentry`).

  Weitere Informationen finden Sie in `man csh`.

  Weitere Informationen finden Sie in csh(1).

Zwei Leerzeichen am Satzende
: Verwenden Sie immer zwei Leerzeichen am Ende eines Satzes. Dadurch erhöht sich die Lesbarkeit des Textes und die Nutzung von Werkzeugen wie Emacs wird vereinfacht.

  Nun könnte man behaupten, dass ein Punkt vor einem Großbuchstaben das Satzende markiert. Vor allem bei Namen, beispielsweise bei „Jordan K. Hubbard“, ist dies allerdings nicht der Fall. Wir haben hier ein großes `K`, gefolgt von einem Punkt und einem Leerzeichen. Dennoch handelt es sich nicht um den Anfang eines neuen Satzes.

Eine ausführliche Beschreibung des korrekten Schreibstils finden Sie im Buch Elements of Style von William Strunk.

10.1. Anleitungen für einen korrekten Schreibstil

Damit die Quellen der Dokumentation selbst dann konsistent bleiben, wenn viele Leute daran arbeiten, folgen Sie bitte den folgenden Konventionen.

10.1.1. Groß- und Kleinschreibung

Tags werden in Kleinbuchstaben geschrieben, Sie schreiben also `para`, *nicht* `PARA`.

Text im SGML-Kontext wird hingegen in Großbuchstaben geschrieben. Man schreibt also `<!ENTITY…>` und `<!DOCTYPE…>`, *nicht* `<!entity…>` und `<!doctype…>`.

10.1.2. Abkürzungen (Akronyme)

Abkürzungen sollten bei ihrer ersten Verwendung immer ausgeschrieben werden. Man schreibt also beispielsweise „Network Time Protocol (NTP)“. Nachdem die Abkürzung definiert wurde, sollte hingegen nur noch die Abkürzung verwendet werden, es sei denn, die Verwendung des gesamten Begriffes ergibt im jeweiligen Kontext mehr Sinn. Im Normalfall werden Akronyme in jedem Dokument nur einmal definiert. Es ist allerdings auch möglich, sie für jedes Kapitel erneut zu definieren.

Die drei ersten Vorkommen der Abkürzung sollten in `acronym`-Tags eingeschlossen werden. Zusätzlich wird ein `role`-Attribut mit dem vollständigen Begriff definiert. Dadurch kann ein Link zu einem Glossar erzeugt werden. Außerdem wird der komplette Begriff angezeigt, wenn man den Mauscursor über die Abkürzung bewegt.

10.1.3. Einrückung

Die erste Zeile jeder Datei hat die Einrückung `0`, und zwar *unabhängig* von der Einrückung der Datei, in der sie enthalten ist.

Öffnende Tags erhöhen die Einrückung um zwei Leerzeichen. Schließende Tags verringern sie hingegen um zwei Leerzeichen. Ein Block von acht Leerzeichen wird durch einen Tabulator ersetzt. Ein solcher Block beginnt immer am Anfang einer Zeile, führende Leerzeichen sind hier also nicht erlaubt. Vermeiden Sie außerdem Leerzeichen am Zeilenende. Der Inhalt von Elementen wird um zwei Leerzeichen eingerückt, wenn er sich über mehr als eine Zeile erstreckt.

Der Quellcode dieses Abschnitts sieht daher in etwa so aus:

```
+--- Einrückung (Spalte) 0
V
<chapter>
  <title>...</title>

  <sect1>
    <title>...</title>

    <sect2>
      <title>Einrückung</title>

      <para>Die erste Zeile jeder Datei hat die Einrückung 0, und
        zwar <emphasis>unabhängig</emphasis> von der Einrückung
        der Datei, in der sie enthalten ist.</para>

      ...
    </sect2>
  </sect1>
```

```
</chapter>
```

Wenn Sie Emacs oder XEmacs verwenden, um Ihre Dateien zu bearbeiten, sollte der `sgml-mode` automatisch geladen werden, und die lokalen Emacs-Variablen am Ende einer Datei sollten diesen Stil erzwingen.

Verwenden Sie Vim, sollten Sie Ihren Editor so konfigurieren:

```
augroup sgmledit
  autocmd FileType sgml set formatoptions=cq2l " Special formatting ↺
options
  autocmd FileType sgml set textwidth=70       " Wrap lines at 70 ↺
columns
  autocmd FileType sgml set shiftwidth=2       " Automatically indent
  autocmd FileType sgml set softtabstop=2      " Tab key indents 2 ↺
spaces
  autocmd FileType sgml set tabstop=8          " Replace 8 spaces ↺
with a tab
  autocmd FileType sgml set autoindent         " Automatic ↺
indentation
augroup END
```

10.1.4. Die korrekte Schreibweise von Tags

10.1.4.1. Einrücken von Tags

Tags, die die gleiche Einrückung aufweisen wie das vorangegangene Tag, sollten durch eine Leerzeile getrennt werden, Tags mit unterschiedlicher Einrückung hingegen nicht:

```
<article lang='de'>
  <articleinfo>
    <title>NIS</title>

    <pubdate>October 1999</pubdate>

    <abstract>
      <para>...
 ...
 ...</para>
    </abstract>
  </articleinfo>

  <sect1>
    <title>...</title>

    <para>...</para>
  </sect1>

  <sect1>
    <title>...</title>

    <para>...</para>
```

```
  </sect1>
</article>
```

10.1.4.2. Gliederung von Tags

Tags wie zum Beispiel `itemizedlist`, die immer weitere Tags einschließen und selbst keine Zeichen enthalten, befinden sich immer in einer eigenen Zeile.

Tags wie `para` und `term` können selbst Text enthalten, und ihr Inhalt beginnt direkt nach dem Tag, und zwar *in der gleichen Zeile*.

Dies gilt analog, wenn diese zwei Tag-Arten wieder geschlossen werden.

Eine Vermischung dieser Tags kann daher zu Problemen führen.

Wenn auf ein Start-Tag, das keine Zeichen enthalten kann, unmittelbar ein Tag folgt, das andere Tags einschließen muss, um Zeichen darzustellen, befinden sich diese Tags auf verschiedenen Zeilen. Das zweite Tag wird dabei entsprechend eingerückt.

Wenn ein Tag, das Zeichen enthalten kann, direkt nach einem Tag, das keine Zeichen enthalten kann, geschlossen wird, befinden sich beide Tags in der gleichen Zeile.

10.1.5. Markup-Änderungen (white space changes)

Wenn Sie Änderungen committen, *committen Sie niemals Inhalts- und Formatierungsänderungen zur gleichen Zeit*.

Nur auf diese Weise können die Übersetzungs-Teams sofort erkennen, welche Änderungen durch Ihren Commit verursacht wurden, ohne darüber nachdenken zu müssen, ob sich der Inhalt einer Zeile oder nur deren Formatierung geändert hat.

Nehmen wir an, Sie haben zwei Sätze in einen Absatz eingefügt. Daher sind zwei Zeilen nun länger als 80 Zeichen. Zuerst committen Sie Ihre inhaltliche Änderung inklusive der zu langen Zeilen. Im nächsten Commit korrigieren Sie den Zeilenumbruch und geben in der Commit-Mitteilung an, dass es sich nur um Änderung am Markup handelt (*whitespace-only change*), und Übersetzer den Commit daher ignorieren können.

10.1.6. Vermeiden von fehlerhaften Zeilenumbrüchen (Nutzung von non-breaking space)

Vermeiden Sie Zeilenumbrüche an Stellen, an denen diese hässlich aussehen oder es erschweren, einem Satz zu folgen. Zeilenumbrüche hängen von der Breite des gewählten Ausgabemedium ab. Insbesondere bei der Verwendung von Textbrowsern können schlecht formatierte Absätze wie der folgende entstehen:

```
Data capacity ranges from 40 MB to 15
GB.  Hardware compression …
```

Die Nutzung der Entity ` ` verhindert Zeilenumbrüche zwischen zusammengehörenden Teilen. Verwenden Sie *non-breaking spaces* daher in den folgenden Fällen:

- Zwischen Zahlen und Einheiten:

```
57600 bps
```

- Zwischen Programmnamen und Versionsnummern:

```
FreeBSD 4.7
```

- Zwischen mehreren zusammengehörenden Wörtern (Vorsicht bei Namen, die aus mehr als 3-4 Wörtern bestehen, wie „The FreeBSD Brazilian Portuguese Documentation Project“):

```
Sun Microsystems
```

10.2. Häufig verwendete Wörter

Die folgende Liste enthält einige Beispiele, wie bestimmte Wörter innerhalb des FreeBSD Documentation Projects geschrieben werden. Finden Sie ein gesuchtes Wort hier nicht, sehen Sie bitte in der Liste häufig verwendeter Wörter von O'Reilly nach.

- 2.2.X
- 4.X-STABLE
- CD-ROM
- DoS *(Denial of Service)*
- Ports Collection
- IPsec
- Internet
- MHz
- Soft Updates
- Unix
- disk label
- email
- file system

- manual page
- mail server
- name server
- null-modem
- web server

Kapitel 11. `sgml-mode` und Emacs

Neuere Emacs- und XEmacs-Versionen verfügen über ein nützliches Lisp-Paket namens PSGML. PSGML (das über den Port editors/psgml installiert werden kann) ist ein so genannter *Majormode*, der Funktionen speziell für den Umgang mit SGML-Dateien, -Elementen und deren Attributen bereit stellt. Emacs aktiviert PSGML automatisch, wenn eine Datei mit der Endung `.xml` geladen oder der Befehl `M-X sgml-mode` eingegeben wird.

Die Arbeit an SGML-Dokumenten wie dem FreeBSD-Handbuch kann sich wesentlich einfacher gestalten, wenn einige der Funktionen von PSGML gekannt sind:

`C-c C-e`
: Ruft die Funktion `sgml-insert-element` auf, die nach dem Namen des einzufügenden Elements fragt. Ist dieser eingegeben worden und wurde die Eingabetaste gedrückt, fügt die Funktion Start- und Endtag des neuen Elements ein. Sofern das eingefügte Element laut DTD andere Elemente enthalten muß, werden diese ebenfalls eingefügt.

  Falls Sie unsicher sind, wie der Name des gewünschten Elements lautet oder welche Elemente an der aktuellen Position erlaubt sind, können mittels der Taste Tab alle *an dieser Stelle* möglichen Elemente angezeigt werden. Ebenso ermöglicht Tab die Vervollständigung eines bereits eingegebenen Elementnamens.

`C-c =`
: Ruft die Funktion `sgml-change-element-name` auf, mit der das aktuelle Element – das Element zwischen dessen Start- und Endtag sich der Cursor befindet – ausgewechselt werden kann. Die Funktion fragt nach dem Namen des neuen Elements und ersetzt anschließend Start- und Endtag des alten Elements durch die des neuen Elements.

`C-c C-r`
: Ruft die Funktion `sgml-tag-region` auf, die einen markierten Textabschnitt mit einem Element umschließt. Dazu markieren Sie zuerst den Textabschnitt (gehen Sie zum Anfang des Abschnitts und führen Sie `C-space` aus, dann gehen Sie zum Ende des Abschnitts und führen erneut `C-space` aus), danach führen Sie diese Funktion aus. Sie werden nach dem Namen des einzufügenden Elements gefragt. Dessen Start-Tag wird dann am Anfang des markierten Textes eingefügt, dessen End-Tag am Ende des markierten Texts.

`C-c -`
: Ruft die Funktion `sgml-untag-element` auf, die Start- und Endtag des Elements entfernt, innerhalb dessen sich der Cursor befindet.

`C-c C-q`
: Ruft die Funktion `sgml-fill-element` auf. Diese Funktion formatiert[1] den Inhalt des aktuellen Elements neu. Dieser Vorgang betrifft auch Elemente wie `programlisting`, in denen Leerzeichen und ähnliches Teil der Formatierung sind. Aus diesem Grund ist mit `sgml-fill-element` bedächtig umzugehen.

`C-c C-a`
: Ruft die Funktion `sgml-edit-attributes` auf. Diese öffnet einen zweiten Puffer mit allen Attributen des Elements, innerhalb dessen sich der Cursor befindet. Über Tab kann von einem Attribut zum nächsten gewechselt werden. Ein existierender Attributwert kann mit `C-k` gelöscht werden. Die Tastenfolge `C-c C-c` schließt den Puffer und setzt die Attribute des Elements entsprechend den Puffervorgaben.

`C-c C-v`
: Ruft die Funktion `sgml-validate` auf, die zuerst fragt, ob das aktuelle Dokument gespeichert werden soll und anschließend einen SGML-Validator aufruft. Die Ausgaben des Validators werden in einem neuen Puffer angezeigt. Dadurch hat der Benutzer die Möglichkeit, eventuell vom Validator gefundene Fehler zu korrigieren.

`C-c /`
: Startet die Funktion `sgml-insert-end-tag` , die automatisch das passende End-Tag für das gerade offene Element einfügt.

Zweifellos hat PSGML noch weitere nützliche Funktionen, doch die hier genannten sind die, die der Autor dieser Fibel am meisten benutzt.

Um den richtigen Einzug, die Umwandlung von Tabulatoren in Leerzeichen und die maximale Zeilenlänge für Dokumente des FDPs sicherzustellen, kann folgender Eintrag in `.emacs` vorgenommen werden:

```
(defun local-sgml-mode-hook
  (setq fill-column 70
        indent-tabs-mode nil
        next-line-add-newlines nil
        standard-indent 4
        sgml-indent-data t)
  (auto-fill-mode t)
  (setq sgml-catalog-files '("/usr/local/share/xml/catalog")))
(add-hook 'psgml-mode-hook
  '(lambda () (local-psgml-mode-hook)))
```

[1] Formatieren bedeutet in diesem Zusammenhang, dass die Funktion versucht, soviel Zeichen wie möglich in einer Zeile unterzubringen. Die Stelle, bis zu der gefüllt und dann der Zeilemumbruch erfolgt, ist konfigurierbar.

Kapitel 12. Weiterführende Quellen

In dieser Fibel werden absichtlich nicht alle Aspekte von XML, der erwähnten DTDs und des FreeBSD-Dokumentationsprojekts behandelt. Interessierten werden daher die nachfolgenden Quellen empfohlen.

12.1. Das FreeBSD-Dokumentationsprojekt

- Die Webseiten des FreeBSD-Dokumentationsprojektes
- Das FreeBSD-Handbuch

12.2. XML

- W3C's XML-Webseite, eine umfangreiche Quelle zu XML.

12.3. HTML

- Das World-Wide-Web-Konsortium
- Die HTML 4.0-Spezifikation

12.4. DocBook

- The DocBook Technical Committee, die Betreuer der DocBook-DTD.
- DocBook: The Definitive Guide, die Online-Dokumentation zur DocBook-DTD.
- The DocBook Open Repository bietet DSSSL-Stilvorlagen und andere Ressourcen für DocBook-Benutzer an.

12.5. Das Linux-Dokumentationsprojekt

- Die Webseiten des Linux-Dokumenationsprojektes.

Anhang A. Beispiele

In diesem Anhang sind XML-Beispieldokumente und Befehle enthalten, die zeigen, wie man aus DocBook-Dokumenten verschiedene Ausgabeformate erzeugen kann. Sofern alle Werkzeuge für das Dokumentationsprojekt ordnungsgemäß installiert wurden, können die angebotenen Beispiele direkt übernommen werden.

Die Beispiele dieses Abschnitts sind bewusst einfach aufgebaut. Daher fehlen in den Beispielen einige Elemente, insbesondere Elemente für die Titelei. Weitere DocBook-Beispiele können in den DocBook-Quellen dieses und anderer Dokumente des FDPs gefunden werden. Die Quellen des FDPs sind im svn doc-Repository und online unter `http://svnweb.FreeBSD.org/doc/` verfügbar.

Um Irritationen zu vermeiden, bauen die XML-Beispiele auf der 4.1er Standard-DocBook DTD anstatt auf der erweiterten FreeBSD-Variante auf. Ebenso werden die Standardstylesheets von Norman Welsh, anstatt der angepassten Stylesheets des FreeBSD-Dokumentationsprojektes benutzt. Dadurch eignen sich die Beispiele auch als generische DocBook-Vorlagen.

A.1. DocBook-Buch (book)

Beispiel A.1. Ein DocBook-Buch (book)

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V4.1//EN">

<book lang='de'>
  <bookinfo>
    <title>Ein Buchbeispiel</title>

    <author>
      <firstname>Vorname</firstname>
      <surname>Nachname</surname>
      <affiliation>
        <address><email>vorname.nachname@domain.de</email></
address>
      </affiliation>
    </author>

    <copyright>
      <year>2000</year>
      <holder>Urheberhinweis</holder>
    </copyright>

    <abstract>
      <para>Falls das Buch eine Zusammenfassung hat, sollte sie
```

```
        hier stehen.</para>
    </abstract>
  </bookinfo>

  <preface>
    <title>Einleitung</title>

    <para>Falls das Buch eine Einleitung hat, sollte diese hier
      stehen.</para>
  </preface>

  <chapter>
    <title>Das erste Kapitel</title>

    <para>Das ist das erste Kapitel des Buches.</para>

    <sect1>
      <title>Der erste Abschnitt</title>

      <para>Das ist der erste Abschnitte des Buches.</para>
    </sect1>
  </chapter>
</book>
```

A.2. DocBook-Artikel (`article`)

Beispiel A.2. Ein DocBook-Artikel (`article`)

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V4.1//EN">

<article lang='de'>
  <articleinfo>
    <title>Ein Beispielartikel</title>

    <author>
      <firstname>Vorname</firstname>
      <surname>Nachname</surname>
      <affiliation>
        <address><email>vorname.nachname@domain.de</email></
address>
      </affiliation>
    </author>

    <copyright>
      <year>2000</year>
      <holder>Urheberhinweis</holder>
```

```
    </copyright>

    <abstract>
      <para>Falls der Artikel eine Zusammenfassung hat, ↺
sollte sie
        hier stehen.</programlisting>
    </abstract>
  </articleinfo>

  <sect1>
    <title>Der erste Abschnitt</title>

    <para>Das ist der erste Abschnitt des Artikels.</para>

    <sect2>
      <title>Der erste Unterabschnitt</title>

      <para>Das ist der erste Unterabschnitt des Artikels.</
para>
    </sect2>
  </sect1>
</article>
```

A.3. Ausgabeformate erzeugen

Für diesen Abschnitt wird vorausgesetzt, dass die im Port textproc/docproj enthaltene Software manuell oder über das Portssystem installiert wurde. Weiter wird vorausgesetzt, dass alle Programme unterhalb des Verzeichnisses `/usr/local` installiert worden sind und die Verzeichnisse, die die ausführbaren Programme enthalten, in der Variable `PATH` enthalten sind.

A.3.1. Benutzung von Jade

Beispiel A.3. Ein DocBook-Dokument in eine einzelne HTML-Datei umwandeln

```
% jade -V nochunks \   ❶
-c /usr/local/share/xml/docbook/dsssl/modular/catalog \    ❷
-c /usr/local/share/xml/docbook/catalog \
-c /usr/local/share/xml/jade/catalog \
-d /usr/local/share/xml/docbook/dsssl/modular/html/docbook.↺
dsl \❸
-t sgml ❹ datei.xml > datei.html  ❺
```

❶ Übergibt den Parameter `nochunks` an die Stylesheets. Dadurch wird die gesamte Ausgabe in die Standardausgabe geschrieben (bei der Benutzung von Norm Walshs Stylesheets).
❷ Legt die von Jade zur Verarbeitung benötigten drei Kataloge fest. Der erste Katalog enthält Informationen zu den DSSSL-Stylesheets, der zweite zur DocBook DTD und der dritte Jade-spezifische Informationen.
❸ Übergibt den vollen Pfad zum DSSSL-Stylesheet, das von Jade zur Verarbeitung des Dokuments benutzt wird.
❹ Weist Jade an, eine *Transformation* von einer DTD zu einer anderen DTD vorzunehmen. In diesem Falle, von der DocBook DTD zur HTML DTD.
❺ Legt fest, welche Datei Jade verarbeiten soll und leitet die Ausgabe in die Datei `datei.html` um.

Beispiel A.4. Ein DocBook-Dokument in mehrere kleine HTML-Dateien umwandeln

```
% jade \
-c /usr/local/share/xml/docbook/dsssl/modular/catalog \    ❶
-c /usr/local/share/xml/docbook/catalog \
-c /usr/local/share/xml/jade/catalog \
-d /usr/local/share/xml/docbook/dsssl/modular/html/docbook.↺
dsl \❷
-t sgml ❸ datei.xml ❹
```

❶ Legt die von Jade zur Verarbeitung benötigten drei Kataloge fest. Der erste Katalog enthält Informationen zu den DSSSL-Stylesheets, der zweite zur DocBook DTD und der dritte Jade-spezifische Informationen.
❷ Übergibt den vollen Pfad zum DSSSL-Stylesheet, das von Jade zur Verarbeitung des Dokuments benutzt wird.
❸ Weist Jade an, eine *Transformation* von einer DTD zu einer anderen DTD vorzunehmen. In diesem Falle, von der DocBook DTD zur HTML DTD.
❹ Legt die zu verarbeitende Datei fest. Die Stylesheets ermitteln eigenständig die Namen aller HTML-Ausgabedateien.

Abhängig von der Struktur des zu verarbeitenden Dokumentes und den Stylesheetregeln zur Aufteilung des Dokumentes, kann dieser Befehl auch nur eine einzelne HTML-Datei erzeugen.

Beispiel A.5. Ein DocBook-Dokument nach Postscript umwandeln

Um eine Postscript-Ausgabe zu erzeugen, muss zuerst die XML-Quelle in eine TeX-Datei umgewandelt werden.

```
% jade -V tex-backend \ ❶
    -c /usr/local/share/xml/docbook/dsssl/modular/catalog \    ❷
    -c /usr/local/share/xml/docbook/catalog \
    -c /usr/local/share/xml/jade/catalog \
    -d /usr/local/share/xml/docbook/dsssl/modular/print/
docbook.dsl \❸
    -t tex ❹ datei.xml
```

❶ Weist die Stylesheets an, verschiedene TeX-spezifische Optionen zu benutzen.

❷ Legt die von Jade zur Verarbeitung benötigten drei Kataloge fest. Der erste Katalog enthält Informationen zu den DSSSL-Stylesheets, der zweite zur DocBook DTD und der dritte Jade-spezifische Informationen.

❸ Übergibt den vollen Pfad zum DSSSL-Stylesheet, das von Jade zur Verarbeitung des Dokuments benutzt wird.

❹ Weist Jade an, die Ausgabe in eine TeX-Datei umzuwandeln.

Die so erzeugte `.tex`-Datei muss anschließend mittels `tex`, unter Angabe des Makropakets `&jadetex` weiterverarbeitet werden.

```
% tex "&jadetex" datei.tex
```

`tex` muss *mindestens* dreimal ausgeführt werden. Der erste Lauf ermittelt die die Querverweise innerhalb des Dokumentes, die für Stichwortverzeichnisse und ähnliches benötigt werden.

Warnungen, wie LaTeX Warning: Reference `136' on page 5 undefined on input line 728., die zu diesem Zeitpunkt ausgegeben werden, können ohne weiteres ignoriert werden.

Der zweite Lauf kann jetzt, da mehr Informationen, wie zum Beispiel die Seitenlängen, zur Verfügung stehen, Einträge im Stichwortverzeichnis und Querverweise genauer bestimmen.

Der dritte Lauf ist für abschließende Aufräumarbeiten notwendig. Die so von `tex` erzeugte Ausgabe befindet sich anschließend in der Datei `datei.div` .

Zum Schluss muss noch `dvips` aufgerufen werden, um die `.div`-Datei in ein Postscript-Dokument umzuwandeln.

```
% dvips -o datei.ps datei.dvi
```

Beispiel A.6. Eine PDF-Datei aus einem DocBook-Dokument erzeugen

Die ersten Schritte, um ein DocBook-Dokument in ein PDF umzuwandeln, stimmen mit denen überein, die notwendig sind, um eine Postscript-Ausgabe zu erzeugen (Beispiel A.5, „Ein DocBook-Dokument nach Postscript umwandeln“).

Nachdem die `.tex`-Datei durch Jade erzeugt wurde, muss pdfTeX stattdessen mit dem Makropaket `&pdfjadetex` aufgerufen werden.

```
% pdftex "&pdfjadetex" datei.tex
```

Dieser Befehl muss ebenfalls dreimal ausgeführt werden. Am Ende liegt mit `datei.pdf` das fertige PDF-Dokument vor. Weitere Schritte sind jetzt nicht mehr notwendig.

Stichwortverzeichnis

M

Mitgliedschaft, 1