Die Fibel für neue Mitarbeiter des FreeBSD-Dokumentationsprojekts

Die Fibel für neue Mitarbeiter des FreeBSD-Dokumentationsprojekts

Version: 47399
2015-09-09 18:17:42Z von jkois.

Zusammenfassung

Vielen Dank für Ihr Interesse und Ihre Mitarbeit an der FreeBSD-Dokumentation. Wir freuen uns über jeden Beitrag.

Diese Fibel enthält die Informationen, die Sie für die Mitarbeit am FreeBSD-Dokumentationsprojekt (auch als FDP bekannt) benötigen. Diese reichen von verpflichtender und optionaler Software bis hin zur Philosophie des FreeBSD-Dokumentationsprojekts.

Bitte beachten Sie, dass diese Fibel *jederzeit* unter Bearbeitung und noch nicht vollständig ist. Falls Sie einen Fehler finden, würden wir uns freuen, wenn Sie uns darüber informieren.

Inhaltsverzeichnis

Tabellenverzeichnis

Liste der Beispiele

Benutzungshinweise

1. Die Eingabeaufforderungen

Die folgende Tabelle enthält die Eingabeaufforderung eines normalen Benutzers sowie die des Superusers. Die in diesem Buch verwendeten Beispiele benutzen die jeweilige Eingabeaufforderung, um zu zeigen, als welcher Benutzer die Beispiele ausgeführt werden.

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`.
Bildschirmausgaben	`You have mail.`
Bildschirmein- und ausgaben	`% date +"The time is %H:%M"` `The time is 09:18`
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.
Text, der vom Benutzer durch seine Eingaben ersetzt werden muss	Um die Hilfeseiten nach einem bestimmten Begriff zu durchsuchen, geben Sie `man -k` *`Suchbegriff`* ein.
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 (auch nur FDP genannt). Qualitativ hochwertige Dokumentation ist sehr wichtig für den Erfolg von FreeBSD. Jeder Beitrag, der zu diesem Projekt geleistet wird, ist ungemein wertvoll.

Dieses Dokument macht den Leser mit dem FDP vertraut und erklärt, wie man selbst Dokumente erstellt und einreicht und wie die verfügbaren Werkzeuge effektiv beim Schreiben eingesetzt werden können.

Jeder kann zum FDP beitragen. Die einzige Voraussetzung ist die Bereitschaft, helfen zu wollen.

Nach dem Lesen dieses Dokuments werden Sie in der Lage sein,

- die vom FDP betreuten Dokumente zu erkennen,
- die benötigten Dokumentations-Werkzeuge und Dateien zu installieren,
- Änderungen an der Dokumentation vorzunehmen,
- Änderungen zur Begutachtung durch das FDP einreichen können.

1.1. Die FreeBSD-Dokumentationsreihe

Das FDP umfasst vier verschiedene Kategorien:

- *Handbook:* Das Handbuch ist die umfassende Quelle und Referenz für FreeBSD-Benutzer.
- *FAQ:* Eine Sammlung von kurzen Fragen und Antworten zu Themen, die auf den verschiedenen Mailinglisten und in auf FreeBSD spezialisierten Foren regelmäßig diskutiert werden. Lange und komplizierte Antworten werden Sie hier nicht finden.
- *Manualpages:* Die englischen Manualpages werden normalerweise 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.
- *Die Webseite:* Die Hauptpräsenz von FreeBSD im Internet, zu erreichen unter http://www.FreeBSD.org oder einem der zahlreichen Spiegelserver. Für viele Menschen ist sie der erste Kontakt mit FreeBSD.

Übersetzer-Teams sind für die Übersetzung des Handbuchs und der Webseite in verschiedene Sprachen verantwortlich. Manualpages werden derzeit nicht übersetzt.

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

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

Committ-Nachrichten des FDP sind über svn abrufbar und werden zusätzlich unter `http://lists.FreeBSD.org/mailman/listinfo/svn-doc-all` archiviert.

Beide Repositories sind auch über ein Web-Interface erreichbar: https://svnweb.FreeBSD.org/doc/ sowie https://svnweb.FreeBSD.org/base/.

Viele Menschen haben FreeBSD-spezifische 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. Schnellstart

Dieser Abschnitt beschreibt, was Sie tun müssen, bevor Sie Änderungen an der FreeBSD-Dokumentation vornehmen können. Abonnieren Sie zuerst die Mailingliste FreeBSD documentation project. Einige Mitglieder dieser Mailingliste sind auch auf dem IRC-Kanal `#bsddocs` auf EFnet erreichbar. Nehmen Sie mit Ihnen Kontakt auf, wenn Sie Fragen oder Probleme bei der Arbeit an der FreeBSD-Dokumentation haben.

1. Installieren Sie den Metaport textproc/docproj als Port oder Paket, um automatisch alle für die Arbeit an der Dokumentation benötigten Werkzeuge zu installieren.

2. Installieren Sie eine lokale Arbeitskopie der Dokumentation von einem Spiegelserver des FreeBSD-Repository nach `~/doc` (lesen Sie dazu auch Kapitel 3, *Die Arbeitskopie*).

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

3. Ihr Texteditor sollte für die Arbeit an der FreeBSD-Dokumentation wie folgt konfiguriert sein:

   - Zeilenumbruch nach 70 Zeichen.
   - Tabstop auf 2 Zeichen.
   - 8 Leerzeichen sollen durch einen Tabstop ersetzt werden.

   Konfigurationen für einige häufig verwendete Editoren finden Sie im Kapitel 13, *Der Schreibstil*.

4. Aktualisieren Sie Ihre lokale Arbeitskopie:

   ```
   % svn up ~/doc
   ```

5. Bearbeiten Sie die Datei. Bevor Sie umfangreiche Änderungen an einer Datei vornehmen, kündigen Sie die geplanten Änderungen bitte auf der Mailingliste an.

   Eine Auflistung häufig verwendeter Tags und Entities finden Sie in Kapitel 8, *XHMTL Markup (noch nicht übersetzt)* und in Kapitel 9, *DocBook Markup (noch nicht übersetzt)*.

6. Nachdem Sie Ihre Änderungen vorgenommen haben, prüfen Sie diese auf potentielle Probleme:

   ```
   % igor -R filename.xml | less -RS
   ```

   Werden Fehler gemeldet, editieren Sie die Datei erneut. Speichern Sie das Ergebnis und fühen Sie den Test erneut aus. Wiederholen Sie dies solange, bis keine Fehler mehr gemeldet werden.

7. Bauen Sie die Dokumentation *immer*, bevor Sie Änderungen einreichen. Dazu führen Sie **`make`** im Hauptverzeichnis des Dokuments aus, dass Sie gerade bearbeiten. Um beispielsweise die deutsche Version des FreeBSD-Handbuchs als einzelne HTML-Dateien zu bauen, führen Sie **`make`** im Verzeichnis `de_DE.ISO8859-1/books/handbook/` aus. Durch diesen Schritt wird sichergestellt, dass Ihre Änderungen den Bau der Dokumentation nicht wegen eines Fehlers abbrechen.

8. Wenn Ihre Änderungen abgeschlossen und erfolgreich getestet wurden, erzeugen Sie eine „Differenzdatei“ mit Ihren Änderungen:

   ```
   % cd ~/doc
   % svn diff > bsdinstall .diff.txt
   ```

   Geben Sie der Differenzdatei einen aussagekräftigen Namen. Im angegebenen Bespiel wurden Änderungen im Abschnitt `bsdinstall` des Handbuchs vorgenommen.

9. Reichen Sie Ihre Änderungen über das webbasierte Problembericht-Formular ein. Geben Sie eine kurze Beschreibung in der Form *[patch] kurze Beschreibung des Problems* ein. Als Kategorie wählen Sie `docs` und als Klasse `doc-bug`. Danach geben Sie eine Beschreibung Ihrer Änderungen ein sowie eventuelle weitere wichtige Punkte. Verwenden Sie danach den Button [Browse...], um Ihre Differenzdatei hochzuladen.

Kapitel 2. Die Werkzeuge

Um die FreeBSD-Dokumentation zu verwalten und in verschiedene Formate zu konvertieren, werden diverse Werkzeuge verwendet. Einige dieser Werkzeuge sind verpflichtend und müssen auf Ihrem System installiert sein, bevor Sie den Beispielen in den folgenden Kapiteln folgen können. Andere sind hingegen optional und dienen dazu, zusätzliche Funktionalität bereitzustellen oder das Erzeugen der Dokumentation zu vereinfachen.

2.1. Verpflichtende Werkzeuge

Installieren Sie zuerst den Port `textproc/docproj` über die Ports-Sammlung. Dieser *Metaport* installiert alle verpflichtenden Werkzeuge für die Arbeit an der FreeBSD-Dokumentation. Einige dieser Komponenten werden in den folgenden Abschnitten näher beschrieben.

2.1.1. Die DTDs und die Entitäten

Das FDP benutzt verschiedene *Document Type Definitions* (DTDs) und diverse XML-Entitätensätze. Diese werden durch den Port `textproc/docproj` automatisch installiert.

XHTML DTD (`textproc/xhtml`)
: XHTML ist die meistverwendete Auszeichnungssprache des World Wide Web und wird durchgängig für die FreeBSD-Webseite genutzt.

DocBook DTD (`textproc/docbook-xml-450`)
: DocBook ist als Auszeichnungssprache für technische Dokumentationen entwickelt worden. Ein Großteil der FreeBSD-Dokumentation wird mittels DocBook erstellt.

ISO 8879 entities (`textproc/iso8879`)
: Enties des ISO 8879:1986-Standards, 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.

2.2. 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, teTeX und Modular DocBook Stylesheets (`print/jadetex`, `print/teTeX` und `textproc/dsssl-docbook-modular`)
: Jade, teTeX und Modular DocBook Stylesheets werden eingesetzt, um DocBook-Dokumente nach DVI, Postscript und PDF zu konvertieren. Hierfür müssen die JadeTeX-Makros installiert sein.

  Ist nicht geplant, die Dokumente in einem dieser Formate zu erzeugen (wenn also XHTML und Text ausreichend sind), brauchen Sie diese nicht zu installieren. Dazu deaktivieren Sie diese Option im Konfigurationsbildschirm des Ports `textproc/docproj`.

Vim (`editors/vim`)
: Ein beliebter Texteditor zur Bearbeitung von XML und davon abgeleiteten Dokumenten wie DocBook XML.

Emacs oder XEmacs (`editors/emacs` oder `editors/xemacs`)
: Beide Texteditoren haben einen speziellen Modus zur Bearbeitung von Dokumenten entsprechend den Vorgaben einer XML DTD. Zusätzlich bieten sie Funktionen an, mit denen sich der Tippaufwand reduzieren und Fehlerwahrscheinlichkeit senken lässt.

Kapitel 3. Die Arbeitskopie

Die *Arbeitskopie* ist eine Kopie des FreeBSD Dokumentationsrepositories, die Sie auf Ihren lokalen Computer heruntergeladen haben. Änderungen an der Dokumentation werden in der Arbeitskopie durchgeführt und getestet. Patches für Änderungen im Hauptrepository werden aus der Arbeitskopie erzeugt, nachdem Sie Ihre Änderungen durchgeführt haben.

Eine komplette Kopie des Dokumentationsbaums ist etwa 700 Megabyte groß. Damit Sie die Dokumentation auch in verschiedenen Formaten testen und bauen können, sollten Sie für das Repository mindestens 1 Gigabyte an freiem Speicherplatz bereitstellen.

Die Dateien der FreeBSD-Dokumentation werden mit Subversion verwaltet. Falls es auf Ihrem System noch nicht vorhanden ist, wird dieses Werkzeug vom Port textproc/docproj automatisch installiert.

3.1. Die Dokumentation und Manualpages

Die FreeBSD-Dokumentation besteht nicht nur aus Büchern und Artikeln. Auch die Manualpages für alle Befehle und Konfigurationsdateien sind Teil des FDP. Die Dokumentation ist dabei auf zwei Repositories verteilt: `doc` für Bücher und Artikel sowie `base` für das Betriebssystem und Manualpages. Um Manualpages zu bearbeiten, muss zusätzlich das Repository `base` ausgecheckt werden.

Ein Repository kann multiple Versionen der Dokumentatation enthalten. Änderungen werden in der Regel aber immer nur an der aktuellen Version durchgeführt, die als `head` bezeichnet wird.

3.2. Einen Spiegelserver wählen

Um die Geschwindigkeit zu erhöhen (und die Downloadzeit zu reduzieren), wählen Sie bitte aus der Liste der Subversion Spiegelserver einen Server in Ihrer Nähe. Ersetzen Sie dazu in den folgenden Beispielen die URL *`https://svn0.us-west.FreeBSD.org/`* durch die des von Ihnen gewählten Spiegelservers.

3.3. Ein Verzeichnis für die Arbeitskopie wählen

Die FreeBSD-Dokumentation wird üblicherweise unter `/usr/doc/` , Quellcode (inklusive Manualpages) unter `/usr/src/` installiert. Es ist sinnvoll, Arbeitskopien in einen anderen Ordner auszuchecken, um potentielle Konflikte mit bereits in diesen Ordnern vorhandenen Dokumenten zu vermeiden. Die folgenden Beispiele verwenden daher die Verzeichnisse `~/doc` sowie `~/src`. Bei beiden Verzeichnissen handelt es sich um Unterverzeichnisse des home-Verzeichnisses des jeweiligen Benutzers.

3.4. Die Arbeitskopie auschecken

Der Download einer Arbeitskopie wird als *checkout* bezeichnet und erfolgt über den Befehl `svn checkout`. Das folgende Beispiel checkt die aktuelle Version der Dokumentatation (`head`) aus dem Hauptdokumentationsbaum aus:

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

Das Auschecken des Quellcodes für die Arbeit an den Manualpages erfolgt analog:

```
% svn checkout https://svn0.us-west.FreeBSD.org/base/head ~/src
```

3.5. Die Arbeitskopie aktualisieren

Die Dokumente und Dateien im FreeBSD-Repository ändern sich beinahe täglich. Änderungen werden durchgeführt und committed. Bereits kurz nach einem Checkout kann es daher Unterschiede zwischen Ihrer Arbeitskopie und dem FreeBSD-Hauptrepository geben. Um eine lokale Arbeitskopie auf den Stand des Hauptrepository zu aktualisieren, wenden Sie den Befehl `svn update` auf das Verzeichnis an, in dem sich Ihre lokale Arbeitskopie befindet:

```
% svn update ~/doc
```

Gewöhnen Sie sich an, `svn update` auszuführen, bevor Sie Dokumente bearbeiten. Sonst kann es passieren, dass das Dokument in der Zwischenzeit bearbeitet wurde, Ihre lokale Kopie diese Änderungen aber noch nicht enthält. Es ist deutlich einfacher, die aktuelle Version zu bearbeiten, als Ihre älteren lokalen Änderungen mit den aktuellen Änderungen des Repositories zu kombininieren.

3.6. Änderungen an der Arbeitskopie zurücknehmen

Manchmal ist es notwendig, durchgeführte Änderungen zurück zu nehmen oder überhaupt von vorne zu beginnen. Änderungen an einer Datei können über den Befehl `svn revert` „zurückgesetzt“ werden (die Datei ist danach wieder in ihrer ursprünglichen Version vorhanden). Wollen Sie beispielsweise Ihre Änderungen an der Datei `chapter.xml` zurücksetzen, um die unbearbeitete Originalversion zu erhalten, geben Sie den folgenden Befehl ein:

```
% svn revert chapter.xml
```

3.7. Eine Differenzdatei erstellen

Nachdem Sie eine oder mehrere Dateien bearbeitet haben, müssen Sie die Unterschiede zwischen Ihrer lokalen Arbeitskopie und dem FreeBSD-Repository in einer Datei sammeln, bevor Sie Ihre Änderungen einreichen können. Diese Dateien werden als *diff*-Dateien bezeichnet und können durch den Befehl `svn diff` erzeugt werden:

```
% cd ~/doc
% svn diff > doc-fix-spelling.diff
```

Geben Sie der Datei einen Namen, die den Inhalt beschreibt. Die Differenzdatei im Beispiel enthält Rechtschreibkorrekturen für den gesamten Dokumentationsbaum.

Wenn Sie Ihre Änderungen über das Webformular „Submit a FreeBSD problem report“ einreichen wollen, fügen Sie bitte die Erweiterung `.txt` an den Dateinamen an, damit das Formular sicher erkennt, dass Sie gewöhnlichen Text hochladen wollen.

Vorsicht: `svn diff` protokolliert ALLE Änderungen im aktuellen Verzeichnis (und dessen Unterverzeichnissen). Wollen Sie einige dieser Änderungen noch nicht einreichen, müssen Sie angeben, für welche Dateien Sie eine Differenzdatei erstellen wollen.

```
% cd ~/doc
% svn diff disks/chapter.xml printers/chapter.xml   > disks-printers.diff
```

3.8. Referenzen zu Subversion

Diese Beispiele haben Ihnen den prinzipiellen Umgang mit Subversion gezeigt. Weitere detaillierte Informationen finden Sie im Subversion Book sowie in der Subversion documentation.

Kapitel 4. Dokumentation-Verzeichnisstruktur

Übersetzt von Johann Kois.

Die Struktur der Dateien und Ordner unterhalb von `doc/` hilft dabei,

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, leichter zu machen.

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.

4.1. doc/ als höchste Ebene

Unterhalb von `doc/` existieren zwei Arten von Verzeichnissen, die jeweils über spezifische Dateinamen und eine spezifische Bedeutung verfügen.

Verzeichnis	Bedeutung
`share`	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 XML-Unterstützung nötigen Dateien (darunter die FreeBSD DocBook DTD) unter `share/xml` befinden.
`lang.encoding`	Für jede verfügbare Sprache und Kodierung existiert ein eigenes Unterverzeichnis. Beispiele dafür sind `en_US.ISO8859-1/` oder `zh_TW.UTF-8/`. 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.

4.2. 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.

Verzeichis	Bedeutung
`articles`	DocBook-formatierte Artikel (`article`) oder ähnliche Dokumente. Meist relativ kurz und in Abschnitte aufgeteilt. Artikel sind in der Regel als ein einziges, großes XHTML-Dokument verfügbar.

Verzeichis	Bedeutung
`books`	DocBook-formatierte Bücher (`book`) oder ähnliche Dokumente. Umfangreiche Dokumente, die in Kapitel aufgeteilt werden. Sind in der Regel sowohl als eine einzige, große XHTML-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.
`man`	Dient für Übersetzungen von Manualpages. Es enthält ein oder mehrere `man`*n*-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.

4.3. Dokumentenspezifische Informationen

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

4.3.1. Das Handbuch

`books/handbook/`

Das Handbuch wurde unter Verwendung von DocBook XML (und der vom FreeBSD Project erweiterten XML 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.

4.3.1.1. 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.

4.3.1.1.1. `Makefile`

Das `Makefile` definiert verschiedene Variablen zur Konvertierung derXML-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.

4.3.1.1.2. `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.

4.3.1.1.3. *`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

Benennen Sie Kapitel und Verzeichnisse nicht nach Ihrer Reihenfolge innerhalb des Handbuchs. Dann führt eine Umstrukturierung des Handbuchs im Normalfall nicht dazu, dass dafür Dateien umbenannt werden müssen (es sei denn, einzelne Kapitel werden neu aufgenommen oder entfernt).

Die Datei `chapter.xml` ist keine komplette XML-Datei. Dies bedeutet, dass sie nicht alleine gebaut werden kann, sondern nur als Teil des Handbuchs.

Kapitel 5. Die Erzeugung der Zieldokumente

Übersetzt von Johann Kois.

Dieses Kapitels erklärt detailliert, *wie der Bau der Dokumentation organisiert ist* und wie Sie diesen Prozess mit make(1) beeinflussen können.

5.1. DocBook in verschiedene Ausgabeformate konvertieren

Aus einer einzigen DocBook-Quellcodedatei können verschiedene Ausgabeformate erstellt werden. Welches Dateiformat erstellt wird, wird über die Variable `FORMATS` festgelegt. Eine Liste aller verfügbaren Formate ist in `KNOWN_FORMATS` gespeichert:

```
% cd ~/doc/en_US.ISO8859-1/books/handbook
% make -V KNOWN_FORMATS
```

Tabelle 5.1. Häufige Ausgabeformate

FORMATS	Dateityp	Beschreibung
`html`	HTML, Einzeldatei	Eine einzelne `book.html` oder `article.html`.
`html-split`	HTML, multiple Dateien	Multiple HTML-Dateien, eine für jedes Kapitel oder für jeden Abschnitt. Dieser Typ wird in der Regel für die Nutzung des Dokuments auf einer Internetseite verwendet.
`pdf`	PDF	Portable Document Format

Welches Format verwendet wird, hängt vom jeweiligen Dokument ab, in der Regel handelt es sich aber um `html-split`. Weitere Formate werden über die Variable `FORMATS` angegeben. Dabei können Sie ein einzelnes Format, aber auch mehrere Formate gleichzeitig definieren.

Beispiel 5.1. Das Dokument als eine einzelne HTML-Seite bauen

```
% cd ~/doc/en_US.ISO8859-1/books/handbook
% make FORMATS=html
```

Beispiel 5.2. Das Dokument in den Formaten HTML-Split sowie PDF bauen

```
% cd ~/doc/en_US.ISO8859-1/books/handbook
% make FORMATS="html-split pdf"
```

5.2. Für den Bau der FreeBSD-Dokumentation benötigte Werkzeuge

Die folgende Werkzeuge werden benötigt, um die FDP-Dokumente zu bauen und zu installieren.

- Das wichtigste Werkzeug ist make(1), genauer Berkeley Make.
- Der Bau von Paketen erfolgt unter FreeBSD mit pkg_create(1).
- gzip(1) dient zur Erstellung komprimierter Versionen der Dokumentation. bzip2(1) wird ebenfalls unterstützt. Wollen Sie Pakete der Dokumentation erstellen, benötigen Sie auch tar(1).
- Mit install(1) installieren Sie die Dokumentation auf Ihrem System.

5.3. 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`.

5.3.1. 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(1)-Variablen `SUBDIR`, `COMPAT_SYMLINK`, und `DOC_PREFIX`.

Die `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(1) eingebaute Variable, die den Pfad des aktuellen Verzeichnisses enthält.

Die letzte Zeile bindet `doc.project.mk`, die zentrale, projektweite make(1)-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.

5.3.2. Dokument-`Makefile`s

Diese `Makefile`s 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.

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

5.4. make(1)-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.

5.4.1. `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"
```

5.4.1.1. 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 nicht, welche Dokumente gebaut werden können oder sollen. Diese Variable wird lediglich dazu verwendet, häufig verwendete Dokumente in das Wurzelverzeichnis der installierten Dokumentation zu verlinken.

5.4.1.2. Bedingungen

Die Zeile `.if defined(DOC)` ist ein Beispiel für eine make(1)-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.

5.4.2. `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.

5.4.2.1. 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.

5.4.2.2. 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.

5.4.2.2.1. 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).

5.4.2.3. 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.

5.4.2.4. 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 6. Die Webseite

Übersetzt von Johann Kois.

6.1. 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 *~/doc*, in dem sich bereits alle benötigten Dateien befinden.

Sie starten den Bau der Webseiten, indem Sie in das Unterverzeichnis `en_US.ISO8859-1/htdocs` des Dokumentationsbaums (in unserem Beispiel also unter `~/doc`) wechseln und dort den Befehl `make all` ausführen.

```
% cd ~/doc/en_US.ISO8859-1/htdocs
% make all
```

Tipp

Der Bau der Webseiten erfordert die Datei `INDEX` der Ports-Sammlung und schlägt fehl, wenn `/usr/ports` nicht existiert. Der einfachste Weg, dies zu vermeiden, ist die Installatin der Ports-Sammlung.

6.2. Installieren der Webseiten auf Ihrem Server

Führen Sie `make 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.

Die Installation der Webseiten wird als `root` ausgeführt, weil die Berechtigungen des Webserver-Verzeichnisses den Schreibzugriff für normale Benutzer verhindern. Im folgenden Beispiel wurden die Webseiten durch den Benutzer `jru` in dessen Heimatverzeichnis, also unter `/usr/home/jru/doc` gebaut.

```
# cd /home/jru/doc/en_US.ISO8859-1/htdocs
# env DESTDIR=/usr/local/www  make install
```

Veraltete (und nicht mehr verwendete) Dateien werden während der Installation nicht automatisch entfernt. Der folgende Befehl findet (und löscht) alle Dateien im Installationsverzeichnis, die in den letzten drei Tagen nicht aktualisiert wurden:

```
# find /usr/local/www  -ctime 3 -delete
```

6.3. 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 anderen Dokumente des Verzeichnisses `en_US.ISO8859-1` (wie 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 *~/doc* gebaut und danach 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
```

`WEB_ONLY`, `ENGLISH_ONLY`, `WEB_LANG` sind Variablen für make(1). Diese werden entweder in `/etc/make.conf` , in `Makefile.inc` oder als Umgebungsvariablen auf der Kommandozeile oder in Ihrer Konfigurationsdatei gesetzt.

Kapitel 7. 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“.

7.1. 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 hinzugefügt, 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.

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

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 seine 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.

7.2. 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.

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

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 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 7.1. 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 7.2. 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>
```

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

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 7.3. 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 7.4. 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 7.5. 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.

7.2.1. 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 7.6. **`.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
```

Beispiel 7.7. **`.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
```

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

```
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 - ein validierender Parser [20]. 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.

Spalte	Bedeutung
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.

7.3. 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.

  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.

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

`>`
: Schließt den mit `<!` begonnenen Ausdruck ab.

7.3.1. 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.

7.3.1.1. 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.

7.3.1.2. 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.

7.3.2. 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.

7.4. 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...

7.5. Kommentare

Kommentare sind SGML-Konstrukte, die normalerweise nur in DTDs gültig sind. Dennoch ist es, wie in Abschnitt 7.4, „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 7.8. 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 7.9. 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.

7.5.1. 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.

7.6. 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*.

7.6.1. 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 betreffenden 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 7.10. 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.

7.6.2. 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 7.11. 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">
]>
```

7.6.3. 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
   ```

7.7. Dateien mit Entitäten einbinden

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

7.7.1. 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.

Beispiel 7.12. 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;
```

[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.

```
</html>
```

Warnung

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

7.7.2. 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 7.13. 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>
```

7.7.3. Fingerübungen...

7.7.3.1. 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.

7.7.3.2. 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.

7.8. Markierte Bereiche

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

Beispiel 7.14. 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.

7.8.1. Schlüsselworte für markierte Bereiche

7.8.1.1. `CDATA` und `RCDATA`

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

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.

[8] auf Englisch *marked sections*

[9] auf Englisch *character data*

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

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 7.15. CDATA als Inhaltsmodell für markierte Bereiche

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

7.8.1.2. 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 7.16. 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 7.17. 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.

7.8.2. 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.

7.9. 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 8. XHMTL Markup (noch nicht übersetzt)

Dieses Kapitel ist noch nicht übersetzt. Lesen Sie daher bitte das Original in englischer Sprache. Wenn Sie bei der Übersetzung mithelfen wollen, schicken Sie bitte eine E-Mail an 'FreeBSD German Documentation Project' <de-bsd-translators@de.FreeBSD.org>.

Kapitel 9. DocBook Markup (noch nicht übersetzt)

Dieses Kapitel ist noch nicht übersetzt. Lesen Sie daher bitte das Original in englischer Sprache. Wenn Sie bei der Übersetzung mithelfen wollen, schicken Sie bitte eine E-Mail an 'FreeBSD German Documentation Project' <de-bsd-translators@de.FreeBSD.org>.

Kapitel 10. 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.

10.1. 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.

10.2. 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.

10.2.1. 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 11. 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 12. PO-Übersetzungen

12.1. Einführung

Das GNU gettext-System bietet Übersetzern eine einfache Möglichkeit, Übersetzungen von Dokumenten zu erstellen und zu verwalten. Übersetzbare Strings werden dabei aus dem Originaldokument extrahiert und in eine PO-Datei (Portable Object-Datei) gespeichert. Übersetzte Versionen der Strings werden über einen zusätzlichen Editor eingefügt. Diese Strings können danach direkt verwendet werden oder zu einer kompletten Übersetzung des Originaldokuments zusammengefügt werden.

12.2. Schnellstart

Die Anleitung geht davon aus, das sie die Anweisungen in Abschnitt 1.2, „Schnellstart“ bereits ausgeführt haben, allerdings müssen Sie noch zusätzlich die Option `TRANSLATOR` im Port textproc/docproj aktivieren. War diese Option bisher nicht aktiviert, öffnen Sie jetzt die Konfiguration, aktivieren die Option und bauen den Port anschließend neu.

```
# cd /usr/ports/textproc/docproj
# make config
# make clean deinstall install clean
```

Das folgende Beispiel zeigt, wie Sie den kurzen Artikel Leap Seconds auf Spanisch übersetzen.

Prozedur 12.1. Installieren Sie einen PO-Editor

- Sie benötigen einen PO-Editor, um die Übersetzungsdateien zu bearbeiten. Dieses Beispiel verwendet dafür editors/poedit.

  ```
  # cd /usr/ports/editors/poedit
  # make install clean
  ```

Prozedur 12.2. Grundkonfiguration

Wenn Sie mit der Arbeit an einer bisher nicht existierenden Sprache oder einem nicht existierenden Dokument beginnen wollen, müssen Sie zuerst die Verzeichnisstruktur und ein `Makefile` anlegen oder von der englischen Originalversion kopieren:

1. Legen Sie ein Verzeichnis für die neue Übersetzung an. Der Quellcode des englischen Artikel befindet sich unter `~/doc/en_US.ISO8859-1/articles/leap-seconds/` . Die spanische Übersetzung muss daher im Verzeichnis `~/doc/es_ES.ISO8859-1/articles/leap-seconds/` erstellt werden. Der Pfad ist also (abgesehen vom sprachspezifischen Verzeichnis) identisch.

   ```
   % svn mkdir --parents ~/doc/es_ES.ISO8859-1/articles/leap-seconds/
   ```

2. Kopieren Sie das `Makefile` des Originaldokuments in das Übersetzungsverzeichnis:

   ```
   % svn cp ~/doc/en_US.ISO8859-1/articles/leap-seconds/Makefile \
       ~/doc/es_ES.ISO8859-1/articles/leap-seconds/
   ```

Prozedur 12.3. Übersetzung

Das Übersetzen eines Dokuments erfolgt in zwei Schritten, zuerst werden die zu übersetzenden Strings aus dem Originaldokument extrahiert, um sie anschließend zu übersetzen. Diese beiden Schritte werden wiederholt, bis ein brauchbares übersetztes Dokument entstanden ist.

1. Extrahieren Sie die zu übersetzenden Strings aus der originalen englischen Version und speichern Sie diese in einer PO-Datei:

```
% cd ~/doc/es_ES.ISO8859-1/articles/leap-seconds/
% make po
```

2. Verwenden Sie einen PO-Editor, um Ihre Übersetzungen in der PO-Datei zu speichern. Dafür gibt es diverse Editoren, dieses Beispiel verwendet `poedit` (das Sie über den Port editors/poedit installieren können).

   Der Name der PO-Datei setzt sich aus einem zwei Zeichen langen Sprachcode, gefolgt von einem Unterstrich sowie einem ebenfalls zwei Zeichen langen Regionencode zusammen. Für eine Übersetzung ins Spanische heißt die Datei daher `es_ES.po` .

```
% poedit es_ES.po
```

Prozedur 12.4. Ein übersetztes Dokument erzeugen

1. Generieren Sie das übersetzte Dokument:

```
% cd ~/doc/es_ES.ISO8859-1/articles/leap-seconds/
% make tran
```

   Der Name der erzeugten Datei entspricht dem Namen der englischen Originaldatei, normalerweise also `article.xml` für Artikel oder `book.xml` für Bücher.

2. Überprüfen Sie die erzeugte Datei, indem Sie sie in HTML bauen und in einem Internetbrowser öffnen:

```
% make FORMATS=html
% firefox article.html
```

12.3. Bisher nicht exisitierende Dokumente übersetzen

Der erste Schritt, um ein neues Dokument zu übersetzen, besteht darin, ein Verzeichnis zu suchen oder zu erstellen, in dem die Übersetzung gespeichert werden soll. FreeBSD legt alle übersetzten Dokumente in einem Unterverzeichnis ab, dessen Name sich aus der Sprache und aus der Region in der Form *`lang_REGION`* zusammensetzt. *`lang`* ist ein Code, der immer aus zwei Zeichen in Kleinbuchstaben besteht. Auf ihn folgt ein Unterstrich und danach der Code für die *`REGION`* (der aus zwei Zeichen in Großbuchstaben besteht).

Tabelle 12.1. Existierende Sprachen

Sprache	Region	Übersetzungsverzeichnis	PODateiname	Zeichensatz
Englisch	Vereinigte Staaten von Amerika	`en_US.ISO8859-1`	`en_US.po`	ISO 8859-1
Bengalisch	Bangladesch	`bn_BD.UTF-8`	`bn_BD.po`	UTF-8
Dänisch	Dänemark	`da_DK.ISO8859-1`	`da_DK.po`	ISO 8859-1
Deutsch	Deutschland	`de_DE.ISO8859-1`	`de_DE.po`	ISO 8859-1
Griechisch	Griechenland	`el_GR.ISO8859-7`	`el_GR.po`	ISO 8859-7
Spanisch	Spanien	`es_ES.ISO8859-1`	`es_ES.po`	ISO 8859-1
French	France	`fr_FR.ISO8859-1`	`fr_FR.po`	ISO 8859-1
Ungarisch	Ungarn	`hu_HU.ISO8859-2`	`hu_HU.po`	ISO 8859-2
Italienisch	Italien	`it_IT.ISO8859-15`	`it_IT.po`	ISO 8859-15
Japanisch	Japan	`ja_JP.eucJP`	`ja_JP.po`	EUC JP
Koreanisch	Korea	`ko_KR.UTF-8`	`ko_KR.po`	UTF-8

Sprache	Region	Übersetzungsverzeichnis	PODateiname	Zeichensatz
Mongolisch	Mongolien	mn_MN.UTF-8	mn_MN.po	UTF-8
Holländisch	Niederlande	nl_NL.ISO8859-1	nl_NL.po	ISO 8859-1
Norwegisch	Norwegen	no_NO.ISO8859-1	no_NO.po	ISO 8859-1
Polnisch	Polen	pl_PL.ISO8859-2	pl_PL.po	ISO 8859-2
Portugisisch	Brasilien	pt_BR.ISO8859-1	pt_BR.po	ISO 8859-1
Russisch	Russland	ru_RU.KOI8-R	ru_RU.po	KOI8-R
Serbisch	Serbien	sr_YU.ISO8859-2	sr_YU.po	ISO 8859-2
Türkisch	Türkei	tr_TR.ISO8859-9	tr_TR.po	ISO 8859-9
Chinesisch	China	zh_CN.UTF-8	zh_CN.po	UTF-8
Chinesisch	Taiwan	zh_TW.UTF-8	zh_TW.po	UTF-8

Die Übersetzungen befinden sich in Unterverzeichnissen des Hauptverzeichnisses der Dokumentation (in den Beispielen von Abschnitt 1.2, „Schnellstart“ ist dies `~/doc/`). Die deutschen Übersetzungen befinden sich daher beispielsweise unter `~/doc/de_DE.ISO8859-1/` , französische Übersetzungen hingegen unter `~/doc/fr_FR.ISO8859-1/` .

Jede Sprache enthält Unterverzeichnisse für die verschiedenen Dokumenttypen, also üblicherweise `articles/` und `books/`.

Kombiniert man diese Verzeichnisnamen, erhält man den kompletten Pfad zu einem Artikel oder zu einem Buch. Die französische Übersetzung des NanoBSD-Artikels ist daher etwa unter `~/doc/fr_FR.ISO8859-1/articles/nanobsd/` gespeichert, die mongolische Übersetzung des Handbuchs hingegen unter `~/doc/mn_MN.UTF-8/books/handbook/` .

Soll ein Dokument in eine bisher nicht existierende Sprache übersetzt werden, muss zuerst ein sprachspezifisches Verzeichnis erstellt werden. Existiert die Sprache hingegen schon, muss nur ein Unterverzeichnis unterhalb von `articles/` oder `books/` angelegt werden (falls noch nicht vorhanden).

Der Bau der FreeBSD-Dokumentation wird durch ein `Makefile` gesteuert, das sich im gleichen Verzeichnis wie die Übersetzung befindet. Für einfache Artikel kann dieses `Makefile` oft unverändert aus der englischen Originalversion übernommen werden. Der Übersetzungsprozess für Bücher ist hingegen komplizierter, da dabei verschiedene getrennte `book.xml` und `chapter.xml`-Dateien in ein gemeinsames Dokument integriert werden, das `Makefile` für die Übersetzung eines Buches muss daher in der Regel immer kopiert und angepasst werden.

Beispiel 12.1. Die spanische Übersetzung des Porter's Handbook erstellen

Starten Sie die bisher nicht vorhandene spanische Übersetzung des Porter's Handbook. Das originale Dokument ist ein Buch im Verzeichnis `~/doc/en_US.ISO8859-1/books/porters-handbook/` .

1. Das Verzeichnis für die spanische Übersetzung (`~/doc/es_ES.ISO8859-1/books/`) existiert bereits, daher müssen Sie nur ein Unterverzeichnis für das Porter's Handbook anlegen:

```
% cd ~/doc/es_ES.ISO8859-1/books/
% svn mkdir porters-handbook
A         porters-handbook
```

2. Kopieren Sie das `Makefile` des Originalen Buchs in den neuen Ordner:

```
% cd ~/doc/es_ES.ISO8859-1/books/porters-handbook
% svn cp ~/doc/en_US.ISO8859-1/books/porters-handbook/Makefile .
```

```
A         Makefile
```

Passen Sie das `Makefile` an, damit es nur eine einzige `book.xml` als Eingabe erwartet:

```
#
# $FreeBSD: head/de_DE.ISO8859-1/books/fdp-primer/po-translations/chapter.xml ↺
47399 2015-09-09 18:17:42Z jkois $
#
# Build the FreeBSD Porter's Handbook.
#

MAINTAINER=doc@FreeBSD.org

DOC?= book

FORMATS?= html-split

INSTALL_COMPRESSED?= gz
INSTALL_ONLY_COMPRESSED?=

# XML content
SRCS=  book.xml

# Images from the cross-document image library
IMAGES_LIB+=    callouts/1.png
IMAGES_LIB+=    callouts/2.png
IMAGES_LIB+=    callouts/3.png
IMAGES_LIB+=    callouts/4.png
IMAGES_LIB+=    callouts/5.png
IMAGES_LIB+=    callouts/6.png
IMAGES_LIB+=    callouts/7.png
IMAGES_LIB+=    callouts/8.png
IMAGES_LIB+=    callouts/9.png
IMAGES_LIB+=    callouts/10.png
IMAGES_LIB+=    callouts/11.png
IMAGES_LIB+=    callouts/12.png
IMAGES_LIB+=    callouts/13.png
IMAGES_LIB+=    callouts/14.png
IMAGES_LIB+=    callouts/15.png
IMAGES_LIB+=    callouts/16.png
IMAGES_LIB+=    callouts/17.png
IMAGES_LIB+=    callouts/18.png
IMAGES_LIB+=    callouts/19.png
IMAGES_LIB+=    callouts/20.png
IMAGES_LIB+=    callouts/21.png

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

SYMLINKS=       ${DESTDIR} index.html handbook.html

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

Damit steht die Dokumentstruktur bereit und Sie können die PO-Datei mit `make po` erstellen.

Beispiel 12.2. Die französische Übersetzung des PGP Keys-Artikels erstellen

Starten Sie die bisher nicht vorhandene Übersetzung des PGP Keys-Artikels. Das englische Original ist ein Artikel, der sich im Verzeichnis `~/doc/en_US.ISO8859-1/articles/pgpkeys/` befindet.

1. Das Verzeichnis für französische Artikel (`~/doc/fr_FR.ISO8859-1/articles/`) existiert bereits, Sie müssen daher nur ein neues Unterverzeichnis für den PGP Keys-Artikel anlegen:

```
% cd ~/doc/fr_FR.ISO8859-1/articles/
% svn mkdir pgpkeys
A         pgpkeys
```

2. Kopieren Sie das `Makefile` des originalen Artikels:

```
% cd ~/doc/fr_FR.ISO8859-1/articles/pgpkeys
% svn cp ~/doc/en_US.ISO8859-1/articles/pgpkeys/Makefile .
A         Makefile
```

Überprüfen Sie das `Makefile`. Da es sich um einen einfachen Artikel handelt, sind vermutlich keine Änderungen am `Makefile` nötig. Der Inhalt der dritten Zeile (`$FreeBSD...$`) wird durch das Versionskontrollsystem ersetzt werden, wenn diese Datei committet wird.

```
#
# $FreeBSD: head/de_DE.ISO8859-1/books/fdp-primer/po-translations/chapter.xml ↺
47399 2015-09-09 18:17:42Z jkois $
#
# Article: PGP Keys

DOC?= article

FORMATS?= html
WITH_ARTICLE_TOC?= YES

INSTALL_COMPRESSED?= gz
INSTALL_ONLY_COMPRESSED?=

SRCS=   article.xml

# To build with just key fingerprints, set FINGERPRINTS_ONLY.

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

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

Damit steht die Dokumentstruktur bereit und Sie können die PO-Datei mit `make po` erstellen.

12.4. Übersetzen

Das gettext-System macht die Übersetzung von Dokumenten einfacher, weil sich der Übersetzer dadurch um weniger Dinge kümmern muss. Die zu übersetzenden Strings werden aus dem Originaldokument in eine PO-Datei extrahiert. Danach wird ein PO-Editor verwendet, um die übersetzten Strings einzugeben.

Das von FreeBSD verwendete PO-Übersetzungssystem überschreibt PO-Dateien nicht, daher können die Strings jederzeit extrahiert werden, um die PO-Datei zu aktualisieren.

Ein PO-Editor wird zum Bearbeiten der Datei benötigt. Die Beispiele in diesem Abschnitt verwenden editors/poedit, da es sich dabei um einen einfach zu bedienenden Editor mit nur wenigen Abhängigkeiten handelt. Es gibt aber auch diverse andere PO-Editoren, die zusätzliche Funktionen haben, die bei der Übersetzung von Dokumenten helfen. Die Ports-Sammlung enthält verschiedene dieser Editoren, beispielsweise devel/gtranslator.

Löschen Sie die PO-Datei auf keinen Fall, da Sie alle Änderungen enthält, die Übersetzer vorgenommen haben.

Beispiel 12.3. Die spanische Version des Porter's Handbook übersetzen

Beginnen Sie mit der Übersetzung des spanischen Porter's Handbook.

1. Wechseln Sie in das Verzeichnis des spanischen Porter's Handbook und aktualisieren Sie die PO-Datei (wie bereits in Tabelle 12.1, „Existierende Sprachen“ erwähnt, heißt diese Datei `es_ES.po` .

```
% cd ~/doc/es_ES.ISO8859-1/books/porters-handbook
% make po
```

2. Laden Sie die Datei in Ihren PO-Editor und beginnen Sie mit der Übersetzung:

```
% poedit es_ES.po
```

12.5. Tips für Übersetzer

12.5.1. XML-Tags beibehalten

Achten Sie darauf, dass Sie keine XML-Tags des englischen Originals verändern.

Beispiel 12.4. XML-Tags beibehalten

Englisches Original:

```
If <acronym> NTP</acronym>  is not being used
```

Spanische Übersetzung:

```
Si <acronym> NTP</acronym>  no se utiliza
```

12.5.2. Leerzeichen beibehalten

Achten Sie darauf, dass Sie Leerzeichen am Beginn und am Ende der zu übersetzenden Strings beibehalten. Ihre Übersetzung muss diese Strings ebenfalls enthalten.

12.5.3. Nicht zu übersetzende Tags

Die folgenden Tags dürfen nicht übersetzt werden:

- `<citerefentry>`
- `<command>`
- `<filename>`
- `<literal>`
- `<manvolnum>`
- `<orgname>`

- `<package>`
- `<programlisting>`
- `<prompt>`
- `<refentrytitle>`
- `<screen>`
- `<userinput>`
- `<varname>`

12.6. Ein übersetztes Dokument bauen

Eine übersetzte Version eines Originaldokuments kann jederzeit erzeugt werden. Noch nicht übersetzte Teile des Dokuments werden dabei in Englisch verbleiben. Die meisten PO-Editoren zeigen Ihnen an, welcher Anteil des Dokuments bereits übersetzt ist. Dies erleichtert es dem Übersetzer zu beurteilen, ob sich der Bau des finalen Dokuments bereits lohnt oder nicht.

Beispiel 12.5. Die spanische Version des Porter's Handbook bauen

Bauen Sie die spanische Version des Porter's Handbooks, das in einem früheren Beispiel erzeugt wurde und überprüfen Sie das Ergebnis.

1. Bauen Sie das Dokument. Da das Original vom Typ Buch (*book*) ist, heißt das erzeugte Dokument `book.xml`.

```
% cd ~/doc/es_ES.ISO8859-1/books/porters-handbook
% make tran
```

2. Erzeugen Sie aus der `book.xml` eine HTML-Version des Dokuments und lassen Sie sich das Ergebnis in Firefox anzeigen. Für die englische Dokumentation gehen Sie analog vor. Eine Liste aller verfügbaren Werte für die Variable `FORMATS` finden Sie in Tabelle 5.1, „Häufige Ausgabeformate“.

```
% make FORMATS=html
% firefox book.html
```

Kapitel 13. 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.

13.1. 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.

13.1.1. 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…>`.

13.1.2. 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.

13.1.3. 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
```

13.1.4. Die korrekte Schreibweise von Tags

13.1.4.1. 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>
```

13.1.4.2. 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.

13.1.5. 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.

13.1.6. 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
  ```

13.2. 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 14. Editor Configuration (noch nicht übersetzt)

Dieses Kapitel ist noch nicht übersetzt. Lesen Sie daher bitte das Original in englischer Sprache. Wenn Sie bei der Übersetzung mithelfen wollen, schicken Sie bitte eine E-Mail an 'FreeBSD German Documentation Project' <de-bsd-translators@de.FreeBSD.org>.

Kapitel 15. 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.

15.1. Das FreeBSD-Dokumentationsprojekt

- Die Webseiten des FreeBSD-Dokumentationsprojektes
- Das FreeBSD-Handbuch

15.2. XML

- W3C's XML-Webseite, eine umfangreiche Quelle zu XML.

15.3. HTML

- Das World-Wide-Web-Konsortium
- Die HTML 4.0-Spezifikation

15.4. 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.

15.5. 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