XML-ANWENDUNGEN MIT ADOBE FRAMEMAKER IN DER TECHNISCHEN DOKUMENTATION

D I P L O M A R B E I T

IM FACHBEREICH INFORMATIK UND KOMMUNIKATIONSSYSTEME DER HOCHSCHULE MERSEBURG (FH)

vorgelegt von
Matthias Schleiff
KTT'02
aus Halle (Saale)

Erster Betreuer: Herr Dr. Thomas Meinike
Zweiter Betreuer: Herr Dipl. Tech.-Red. (FH) Andreas Lenz

eingereicht am: 04.05.2007

Ich danke allen, die mich bei der Anfertigung dieser Diplomarbeit unterstützt haben, vor allem meinen Betreuern Herrn Dr. Thomas Meinike und Andreas Lenz, sowie Ronald Peltsch, Klaus Müller und Christian Günther

Hinweis

Die in dieser Arbeit genannten Marken-, Firmen- oder Produktnamen sind Marken-, Firmen- oder Produktnamen der jeweiligen Hersteller und/oder Eigentümer.

Inhaltsverzeichnis

Einleitung

In dieser Diplomarbeit wird die Entwicklung einer XML-Anwendung für die Software Adobe FrameMaker 7.2 dokumentiert. Mit dem FrameMaker-eigenen Konzept der sogenannten strukturierten Anwendungen werden die Möglichkeiten des DTP-Programms um ein Vielfaches erweitert. Die Strukturen einer Technischen Dokumentation können so in einer Abfolge hierarchisch geordneter Elemente abgebildet und als programmunabhängige XML- oder SGML-Daten gespeichert werden. Die Layoutfunktionen, die zum Erzeugen der Druckausgabe der Dokumente notwendig sind und für die FrameMaker als ein auf Technische Dokumentation spezialisiertes DTP-Programm alle nötigen Funktionalitäten bereit stellt, werden dabei weitgehend automatisiert. Eine XML-Anwendung vereinfacht zum Einen die Bedienung des Programms, zum Anderen entfällt der von reinen XML-Editoren bekannte Schritt der Erzeugung von PDF-Dokumenten mit XSL-FO, da bei FrameMaker diese Funktion bereits integriert ist. Das Printlayout eines Dokuments wird dabei von FrameMaker in der Seitenvorschau jederzeit am Bildschirm angezeigt, kann also beim Erstellen der Dokumente durch den Technischen Redakteur kontrolliert werden. Mit einer entsprechend entwickelten XML-Anwendung für Adobe FrameMaker lassen sich somit die Möglichkeiten des „klassischen“ Desktop-Publishing mit den Vorteilen von XML verbinden.

Ausgangspunkt der Entwicklung war die Analyse eines bestehenden Redaktionsprozesses in der Technischen Dokumentation. Dieser Prozess wird seit Jahren von zwei selbstständigen Technischen Redakteuren bei Lenz-KD erfolgreich eingesetzt; z. B. für die umfangreiche Dokumentation von Schienenfahrzeugen. Er basiert auf einer klaren Strukturierung der Informationen nach den Methoden von Funktionsdesign und Information Mapping® und wurde im Laufe der Zeit immer weiter optimiert. Die Dokumente wurden in diesem Redaktionsprozess bereits mit Adobe FrameMaker realisiert. Das Programm ist dabei jedoch als DTP-Programm ohne seine SGML-/XML-basierten Strukturierungsmöglichkeiten eingesetzt worden.

Im ersten Teil der Arbeit wird auf die allgemeinen Anforderungen an ein Redaktionssystem in der Technischen Dokumentation eingegangen. Daraus wird für den konkreten Fall die Entscheidung für den strukturierten Einsatz von Adobe FrameMaker begründet. Im zweiten Teil wird die Entwicklung der XML-Anwendung mit dem Programm grundlegend erörtert. Im Anhang der Arbeit wird in Teil A eine Bedienungsanleitung der erstellten Anwendung für Technische Redakteure wiedergegeben, die mit der XML-Anwendung für Adobe FrameMaker arbeiten. In Teil B folgt schließlich die zur entwickelten Anwendung gehörende DTD. Der entwickelte XML-Dokumenttyp trägt wie die XML-Anwendung den Arbeitstitel „Documentation Publisher's Markup Language“ (dpML).

1 Anforderungen an ein Redaktionssystem in der Technischen Dokumentation

An ein Redaktionssystem in der Technischen Dokumentation werden hohe Ansprüche gestellt. Mit dem Begriff „Redaktionssystem“ ist in diesem Zusammenhang nicht nur eine Software gemeint, mit der Anwenderdokumentationen für Technische Produkte verfasst werden, sondern der gesamte Prozess der Planung, Organisation und Umsetzung aller notwendigen Arbeitsschritte beim Erstellen von hochwertigen Technischen Dokumentationen. Die Grundlagen dieses Systems können in einem Redaktionsleitfaden zusammengetragen werden, der systematisch die Arbeitsabläufe, die verwendeten Werkzeuge, die Qualitätsmaßstäbe und die Gestaltungsregeln für die Technische Dokumentation beschreibt [Romberg 2000, S. 43]. Eine Redaktionssystem-Software unterstützt den Redakteur idealerweise beim gesamten Erstellungsprozess.

Im folgenden Abschnitt 1.1 „Aspekte der Qualitätssicherung“ werden die Anforderungen an ein Redaktionssystem zum Erstellen von Anwenderdokumentationen unter Aspekten der Qualitätssicherung aufgezeigt. Danach wird in 1.3 „Softwarelösungen in der Technischen Dokumentation“ ein Überblick über bestehende Softwarelösungen gegeben und erörtert, in wie weit sich mit diesen Werkzeugen die Qualitätsstandards verwirklichen lassen. Im Abschnitt 1.2 „Unterstützung des Redaktionsprozesses durch moderne EDV“ werden Konzepte der modernen Elektronischen Datenverarbeitung vorgestellt, die die Prozesse im Bereich der Technischen Dokumentation unterstützen können. Im letzten Abschnitt 1.4 „Die Entscheidung für Adobe FrameMaker“ wird schließlich die Entscheidung für Adobe FrameMaker begründet und das Programm näher vorgestellt.

1.1 Aspekte der Qualitätssicherung

1.1.1 Rechtliche Anforderungen

Die rechtlichen Anforderungen an die Qualität von Technischen Dokumentationen sind in den letzten Jahren in Deutschland enorm gestiegen. Schon länger gilt, dass alle Beteiligten innerhalb einer Absatzkette eines Produktes vom Hersteller über die verschiedenen Handelsstufen bis hin zum Letztverkäufer durch eine entsprechende Qualität der Technischen Dokumentation sicherstellen müssen, dass der Endverbraucher alle Informationen erhält, um das Produkt zu nutzen und insbesondere gefährliche Situationen, die im Umgang mit dem Produkt entstehen können, zu vermeiden. Wird diese Instruktionspflicht verletzt, drohen Schadensersatzansprüche und behördliche Eingriffe in den Produktvertrieb [Heuer 2001, S. 34]. Grundlage dafür bildete die am 1.1.1993 novellierte Fassung des Gerätesicherheitsgesetzes (GSG) und die EG-Richtlinie über die allgemeine Produktsicherheit vom 29.6.1992 für Produkte, die

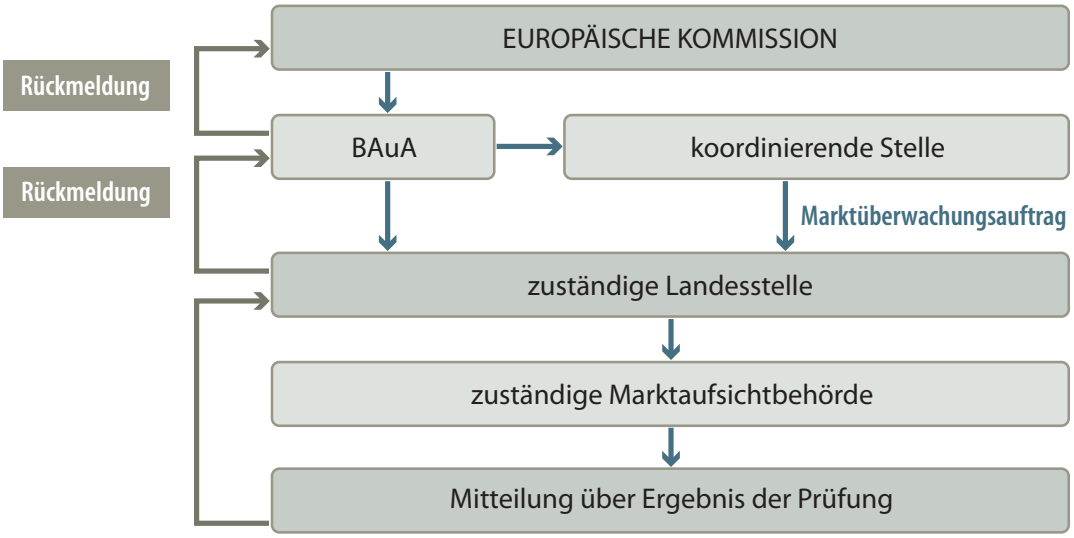


Abbildung 1.1: Ablauf der Marktüberwachung [Quelle: Galbierz 2005, S. 48]

für private Verbraucher bestimmt sind oder von privaten Verbrauchern benutzt werden können. Diese haben rechtliche Mindestanforderungen an den Inhalt von Benutzerinformationen verbindlich vorgegeben, die europaweit gelten [Bauer u. a. 1994, S. 178f].

Die rechtlichen Vorgaben haben sich durch die Umsetzung von EU-Richtlinien weiterentwickelt. Die EG-Richtlinie über die allgemeine Produktsicherheit vom 29.6.1992 ist im Produktsicherheitsgesetz (ProdSG) in nationales Recht umgesetzt worden. Dadurch bestanden einige Mehrfachregelungen durch GSG und ProdSG, die kritisiert wurden. Abhilfe schaffte das zum 1.5.2004 in Kraft getretene Geräte- und Produktsicherheitsgesetz (GPSG), das die beiden älteren Gesetze (GSG und ProdSG) zusammenfasst. Mit dem Geräte- und Produktsicherheitsgesetz wurde die EU-Produktsicherheitsrichtlinie von 2001 in nationales Recht umgesetzt. Die Zielsetzung ist, laut Gesetzestext, „ein einziges umfassendes Gesetz zur Gewährleistung von Sicherheit und Gesundheit im Zusammenhang mit der Vermarktung technischer Produkte“ zu verwirklichen. Wirtschaft, Behörden und vor allem der Verbraucher sollen von der Neuordnung profitieren, da ein einzelnes Gesetz an Stelle von zweien klarere Regelungen schaffe und weniger Bürokratie bedeute. Das GPSG beinhaltet viele der alten Regelungen aus den alten Gesetzen, einige zentrale Aspekte sind hinzugekommen [Galbierz 2004, S. 42].

Die wichtigste Neuerung durch das GPSG ist, dass der Verbraucher noch besser und vorbeugend vor unsicheren Produkten geschützt werden soll. Das bedeutet, dass eine stärkere Kontrolle und Vernetzung der Marktaufsichtsbehörden bezweckt wird und dass die Behörden jetzt präventiv tätig werden müssen. In der Praxis heißt das, dass der Hersteller aufgefordert wird, unsichere Produkte zurückzurufen, auch wenn noch gar kein Schaden durch das Produkt ausgelöst worden ist. Ein Produkt kann somit aufgrund fehlerhafter oder fehlender Instruktionen auf dem Produkt oder in der Gebrauchsanweisung als unsicher eingestuft werden und vom Markt genommen werden [Galbierz 2005, S. 47f].

Entsteht ein Schaden durch einen Produktfehler, so haftet der Hersteller verschuldenunabhängig nach dem Produkthaftungsgesetz (ProdHaftG). Wenn er seine Verkehrs-

sicherheitspflichten schuldhaft (vorsätzlich oder fahrlässig) verletzt hat und dadurch dem Endverbraucher Schaden entstanden ist, haftet er jedoch nach § 823 Abs. 1 BGB, was unter Umständen zu höheren Geldbußen und gegebenenfalls zu Freiheitsstrafen führen kann. Wesentlich bei der deliktischen Haftung nach § 823 Abs. 1 BGB ist die „nachteilige Beweislage“ für den Hersteller. Das heißt der Hersteller muss nachweisen, dass er seine Verkehrssicherheitspflichten nicht vorsätzlich oder fahrlässig verletzt hat [Heuer 2001, S. 36–38].

Hinzu kommt, dass der Gesetzgeber durch die Modernisierung des Schuldrechts 2002 missverständliche Montageanleitungen als Mangel definiert hat. Das heißt, dass der Käufer eines Produktes seinen Gewährleistungsanspruch gegenüber dem Verkäufer auch bei einer mangelhaften Technischen Dokumentation geltend machen kann. Dadurch ist der Verbraucherschutz erneut gestärkt worden [Galbierz 2004, S. 42].

Für den Hersteller ist es daher von vitalem Interesse, auch im Bereich der Technischen Dokumentation sicherzustellen, dass seine Produkte den gängigen Sicherheitsstandards entsprechen. Das GPSG fordert für Bedienungsanleitungen noch stärker als vor seinem Inkrafttreten, dass vor möglichen Fehlbedienungen und einem nahe liegenden Missbrauch des Produktes gewarnt wird [Galbierz 2004, S. 47]. Wie in der Konstruktion sollten dafür die entsprechenden Normen des Deutschen Instituts für Normung (DIN) und der europäischen Normungsinstitutionen (CEN, CENELEC, ETSI) berücksichtigt werden. Für den US-amerikanischen Markt sind die Normen des American National Standards Institute (ANSI) zu beachten. In der internationalen Organisation für Normung (ISO) wird daran gearbeitet, die Normung weltweit zu vereinheitlichen. Einen umfassenden Überblick über die verschiedenen Normenarten und ihre Bedeutung gibt Gabriel [2005]. Grundsätzlich gilt, dass die Anwendung von Normen freiwillig ist, weil diese keine unmittelbare Rechtswirkung haben. Ihnen kann jedoch mittelbar ein verpflichtender Charakter zukommen, da durch sie wesentliche Rechtsbegriffe wie der Stand der Technik ausgefüllt werden. Missachten Unternehmen diese Vorgaben, liegt der Vorwurf der Fahrlässigkeit nahe [Heuer 2001, S. 44].

Der vereinigte Wirtschaftsraum der Europäischen Union erfordert einheitliche technische Standards für Produkte in den Mitgliedsstaaten. Dafür werden entsprechende EU-Richtlinien erarbeitet, die dann in nationales Recht umgesetzt werden, wie es z. B. beim Geräte- und Produktsicherheitsgesetz (GPSG) geschehen ist. Besondere Bedeutung für die praktische Umsetzung der Richtlinien bzw. Gesetze haben die sogenannten „harmonisierten Normen“. Das GPSG definiert eine harmonisierte Norm als eine nicht verbindliche technische Spezifikation, die im Amtsblatt der EU veröffentlicht und nach einem vereinheitlichten Verfahren von einer europäischen Normenorganisation angenommen wurde. Die besondere rechtliche Bedeutung der harmonisierten Normen ist, dass sie die sogenannte Konformitätsvermutung auslösen. Das heißt, dass bei Produkten, die entsprechend dieser Normen gestaltet sind, davon ausgegangen werden kann, dass sie den betreffenden grundlegenden Anforderungen der einzelnen EU-Richtlinien entsprechen [Gabriel 2005, S. 33].

Normen mit Vermutungswirkung gibt es nach dem GPSG auch auf nationaler Ebene. Dadurch bleibt die Konformitätsvermutung nicht nur auf Produkte beschränkt, für die EU-Richtlinien bestehen. Die entsprechenden nationalen Normen werden von der Bundesanstalt für Arbeitsschutz und Arbeitsmedizin im Bundesanzeiger bekannt gegeben. Die Rechtssicherheit für Hersteller und Inverkehrbringer von entsprechenden Produkten ist damit erhöht worden. Normen mit Vermutungswirkung bilden die

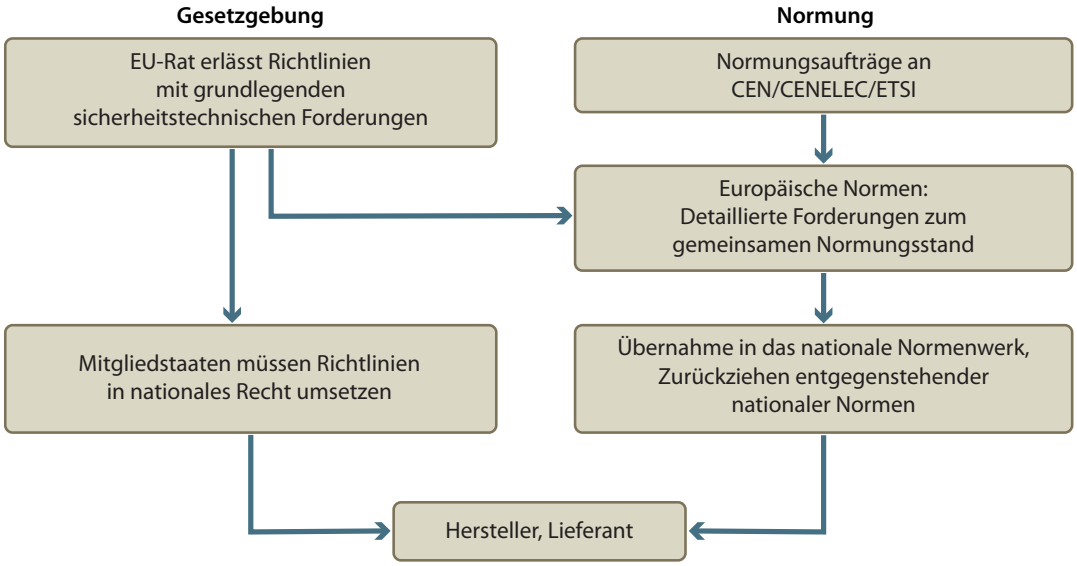


Abbildung 1.2: Zusammenhang zwischen Gesetzgebung und Normung [Quelle: Gabriel 2001, S. 176]

Grundlage für die Überprüfungen durch die Marktüberwachungsbehörden [Gabriel 2005, S, 35].

Werden die Produkte international vermarktet, gelten auf anderen Märkten zum Teil andere Normen, z. B. für den US-amerikanischen Markt die Normen des entsprechenden Normungsinstituts ANSI. Wichtig ist, dass die gültigen Normen, egal ob für den nationalen oder den internationalen Markt, von den Verantwortlichen stets geprüft werden müssen, da es hier immer wieder zu Veränderungen kommen kann. Anfang 2006 trat z. B. die für die Technische Dokumentation relevante Norm DIN EN ISO 12100 in Kraft und ersetzte damit den Normenklassiker EN 292 mit den Teilen 1 und 2. Die Norm gehört zu den harmonisierten Normen mit Konformitätsvermutung und gilt damit als einer der wichtigsten Bausteine auf dem Weg zur CE-Kennzeichnung [Bohn 2006a].

Für den US-amerikanischen Markt ist im Oktober 2006 die Norm ANSI Z535.6-2006 in Kraft getreten. Die Norm regelt die Gestaltung von Sicherheitshinweisen in Betriebsanleitungen und ähnlichen Begleitunterlagen. Die Regelungen spezifizieren diesen Bereich der Sicherheitskennzeichnung genauer, weil vor dem Inkrafttreten der Norm lediglich die Norm ANSI Z535.4 zu Rate gezogen werden konnte, deren Gestaltungsvorgaben sich aber auf Sicherheitskennzeichnung auf Produkten beziehen. Die Norm spezifiziert nun genauer, wie Gebrauchs- und Betriebsanleitungen für den amerikanischen Markt umgesetzt werden sollten [Gabriel 2006].

Für einen tieferen Einstieg in das Thema „Normen für die Technische Dokumentation“ wird an dieser Stelle auf Gabriel [2001] verwiesen. Von der Grundlage der Normung bis zur Normenrecherche wird in diesem Aufsatz auf alle wesentlichen Punkte eingegangen.

Die konkrete Umsetzung der Normen erfordert eine enorme geistige Vorarbeit. Anhand praktischer Leitfäden kann der Technische Redakteur schneller normgerechte

Dokumentationen erstellen. Solche Leitfäden sollten auch die Grundlage für die Verwaltung der Informationen in einem Redaktionssystem bilden. Eine Checkliste für die rechtlich verbindlichen Informationsinhalte für die Technische Dokumentation findet sich z. B. bei Heuer [2001, S. 47]. Wie die Dokumentation „haftungssicher“ gemacht werden kann, füllt bei Rögner [2003] ein ganzes Buch. Bei Leitfäden dieser Art ist aber immer zu beachten, dass sich die gesetzlichen Grundlagen und die entsprechenden Normen weiterentwickeln. Sie entbinden den Technischen Redakteur nicht von der Recherche aktueller Gesetze, Normen und Richtlinien. Nicht umsonst wird im Titel von Rögner [2003] das Wort „haftungssicher“ in Anführungszeichen gesetzt und in den sieben Schritten, die im Laufe des Erstellungsprozesses der Dokumentation nach dieser Anleitung durchlaufen werden sollen, wird im Schritt 3 ebenfalls gefordert, Regelwerke (Gesetze, Richtlinien und Normen) zu recherchieren [Rögner 2003, S. 16].

Die rechtlichen Anforderungen bilden die Grundlagen für klar definierte Qualitätsstandards in der Technischen Dokumentation. Die wichtigste Aufgabe des Technischen Redakteurs ist das Warnen vor den Restgefahren, die von einem Produkt ausgehen, also vor den Gefahren, die durch den Hersteller nicht vorbeugend konstruktiv vermieden werden können. Dabei müssen vor allem die folgenden Punkte beachtet werden:

- Definition des bestimmungsgemäßen Gebrauchs des Produktes
- Warnung vor nahe liegendem Missbrauch
- Warnung vor möglichen Fehlbedienungen
- Qualifikation der Anwender
- Einwirkung auf andere Produkte
- Sachgerechte Aufbewahrung und Pflege des Produktes

Weiterführende Informationen finden sich zu dazu u. a. bei Heuer [2001, S. 39–43], Romberg [2000, S. 23f] und Galbierz [2005, S. 48].

Um die Restgefahren eines Produktes erkennen und dokumentieren zu können, muss eine Risikobeurteilung durchgeführt werden, die im Ergebnis die oben genannten Punkte herausarbeiten kann. Dafür gibt es ebenfalls Normen und Richtlinien, z. B. die DIN EN 1050 „Leitsätze zur Risikobeurteilung“ [vgl. Bohn 2006b].

Erst nachdem man die Restgefahren eines Produktes abschätzen kann, sollte man mit dem Schreiben der Technischen Dokumentation beginnen.

1.1.2 Texte und Textstrukturen

Die Textqualität einer Anwenderdokumentation muss sich an den Erwartungen der Anwender orientieren. Romberg [2000, S. 20] betont, dass Technische Dokumentationen in der Regel nicht zum Vergnügen gelesen werden, sondern um Wissen und Fertigkeiten im Umgang mit dem Produkt zu erwerben und stellt folgende Anforderungen zusammen:

- Einfache und logische Struktur, in der die benötigten Informationen schnell und einfach zu finden sind:

  - Navigationshilfen
  - Orientierungshilfen
  - Übersichtliches Layout
- Verständlich dargebotene textliche und bildliche Informationen:
  - Ohne übermäßige Anstrengung und ohne fremde Hilfe verständlich
  - Klare Gliederung
  - Unkomplizierte, nüchterne Sprache
- Einheitliches Erscheinungsbild der inhaltlichen, strukturellen und formalen Gestaltung:
  - Konsistenz bezogen auf die Handhabung, nicht auf die Formulierung
  - Konsistenz von Textstrukturierungsmitteln, Layout, Typografie, Bilder, Piktogramme, Terminologie und Abkürzungen
- Sachliche Richtigkeit aller Informationen über das Produkt und aller Instruktionen zu seiner Verwendung:
  - Keine inhaltlichen Widersprüche
  - Keine Unklarheiten
- Produktindividuelle bzw. produktserienspezifische Ausführung:
  - Inhalt ist aktuell auf das gelieferte Produkt hinsichtlich Funktionalität und Ausstattung abgestimmt
- Vollständigkeit:
  - Alle Informationen für den Umgang mit dem Produkt und zum Nutzen seines benötigten Funktionsumfanges sind zielgruppenspezifisch enthalten

Einen ähnlichen Kriterienkatalog stellt Oehmig [2006] vor, mit dessen Hilfe die Qualität Technischer Dokumentationen überprüft werden kann. Als Voraussetzung der Anwendung wird allerdings eindeutig eine Qualifikation im Bereich der „Technischen Redaktion“ gefordert [Oehmig 2006, S. 43]. Ein gut ausgebildeter, erfahrener Redakteur lässt sich im Bereich der Textqualität nicht durch Software ersetzen. Um die Strukturierung der Texte in konsistenter Weise vornehmen zu können ist es für den Redakteur allerdings von Vorteil, mit Software zu arbeiten, die ihn bei dieser Arbeit unterstützt und die die Strukturierung auf ansprechende Weise visualisiert [Badenbrink und Ried 2001, S. 108–110].

Auf sprachlicher Ebene Konsistenz herzustellen ist eine komplexe Aufgabe, die mit dem Einsatz einer sogenannten „kontrollierten Sprache“ gelöst werden kann. Mithilfe verschiedener Methoden (z. B. Negativlisten, Positivlisten, Terminologiesammlungen) werden dabei Formulierungen innerhalb der Technischen Dokumentation vereinheitlicht. Ziele einer kontrollierten Sprache sind Textverständlichkeit, leichte und wirtschaftliche Übersetzbarkeit, Rechtssicherheit und Umsetzung der Corporate Identity auf sprachlicher Ebene (Corporate Wording) [Reuther 2006]. Für den Anwender hat kontrollierte Sprache einen großen Nutzen, da Anwenderdokumentationen durch sie

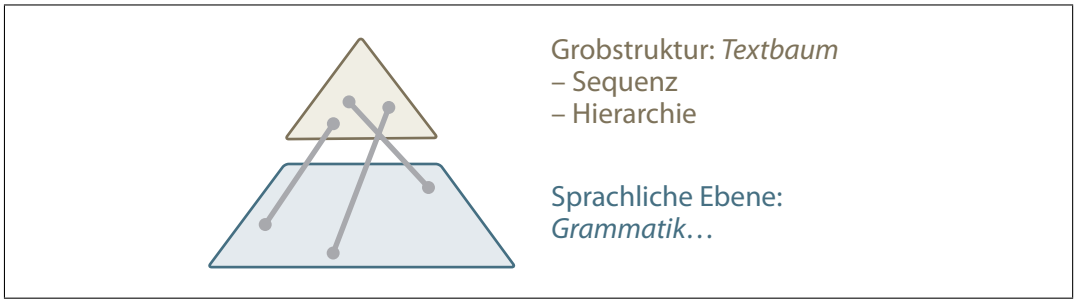


Abbildung 1.3: Texte lassen sich in Grobstruktur und sprachliche Ebene gliedern [Quelle: Badenbrink und Ried 2001, S. 103]

verständlich werden. Die Hersteller global vertriebener Produkte, für die eine Technische Dokumentation in vielen verschieden Sprachen erstellt wird, erreichen außerdem einen wirtschaftlichen Vorteil, wenn beim Übersetzungsprozess moderne Werkzeuge wie Translation-Memory-Systeme zum Einsatz kommen.

Die Strukturierung der Texte erfordert eine einheitliche Systematik. Bei näherer Untersuchung können Textstrukturen nach verschiedenen Prinzipien in hierarchisch geordnete Ebenen unterteilt werden. Klassisch wird hier in Mikro- und Makrostruktur unterschieden. Die Mikrostruktur ist die sprachliche Ebene, auf der die Informationen abhängig von ihrer Grammatik kodiert sind. Die Makrostruktur beschreibt die Grobgliederung des Textes in übergeordnete Sinnzusammenhänge [Badenbrink und Ried 2001, S. 104].

Ley [2006, S. 52] führt zwei zusätzliche Ebenen ein, zum einen die Mesostruktur oberhalb der Mikrostruktur und zum anderen die Metastruktur oberhalb der Makrostruktur. Die zusätzlichen Ebenen werden aufgrund der Prinzipien benötigt, nach denen die Texte strukturiert werden. Ley [2006, S. 51] nennt Funktionsprinzip und Inhaltsprinzip und erläutert, dass einem Informationsträger des Textes, wie z. B. einem Satz oder Absatz nach dem Funktionsprinzip eine bestimmte Funktion zugeordnet werden kann. Diese Funktion ist auch abhängig von dem Kontext, in dem Informationseinheiten auftauchen. Jeder Teil des Informationsproduktes kann ebenso inhaltlich beschrieben werden, was dann das Inhaltsprinzip darstellt. Die Mesostruktur ist die Ebene, auf der das Funktionsprinzip am stärksten zum Tragen kommt. Die Elemente auf dieser Ebene werden als „funktionale“ oder „kommunikative“ Einheiten bezeichnet. Ihnen kommt eine zentrale Rolle bei der Erstellung von Informationsprodukten zu. Die Metastruktur ist die höchste Ebene eines Informationsproduktes [Ley 2006, S. 52].

Wendet man die klassische Unterscheidung zwischen Mikro- und Makrostruktur an, so gehören die Ebenen Meso- und Metastruktur noch zur Makrostruktur, da diese übergeordnete Sinnzusammenhänge darstellen [vgl. Badenbrink und Ried 2001, S. 104].

Das Konzept der Unterscheidung von Informationseinheiten nach ihrer kommunikativen Funktion ist in der Strukturierungsmethode Funktionsdesign konsequent entwickelt worden [Ley 2006, S. 51]. Ähnlich ist auch das durch Dentz [2001] beschriebene Konzept des Informationsdesigns aufgebaut. Beim Informationsdesign werden die Informationseinheiten nach Informationsklassen klassifiziert und bestimmten Informationsklassen Schlüsselabschnitte zugeordnet, für die genaue Designrichtlinien

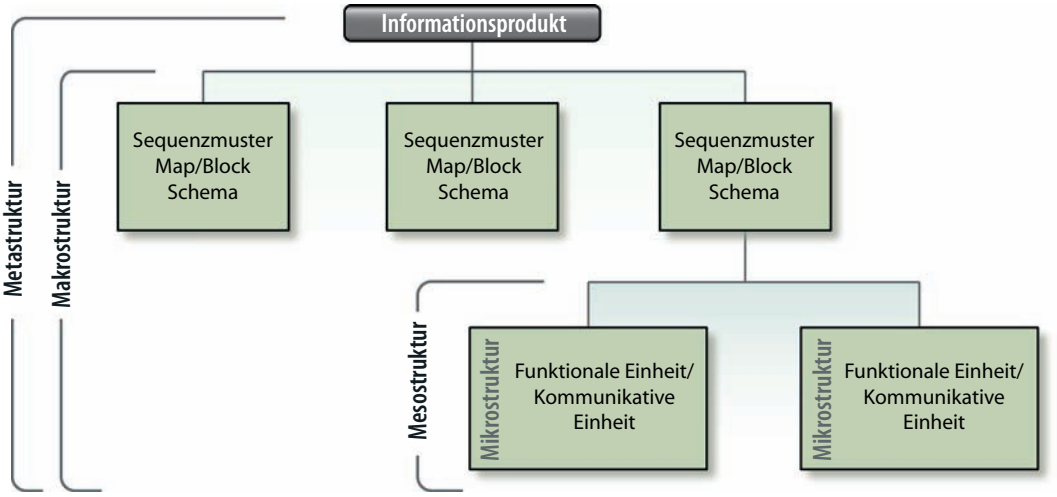


Abbildung 1.4: Ebenen der Textstruktur nach Ley [Quelle: Ley 2006, S. 52]

spezifiziert werden. Eine Anwendung dieser wissenschaftlichen Prinzipien ist die Information Mapping Methode. Information Mapping® stellt ein konkretes und am Rezeptionsverhalten des Anwenders orientiertes Regelwerk zum Strukturieren Technischer Dokumentationen zur Verfügung [vgl. Böhler 2001].

Grundsätzlich sollten bei der Konzipierung der Struktur einer Anwenderdokumentation die Grenzen menschlicher Fähigkeiten der Informationsverarbeitung mitbedacht werden [vgl. Romberg 2000, S. 11–19]. Das ist beispielsweise bei der Konzeption der Information Mapping Methode berücksichtigt worden [Böhler 2001, S. 130–132].

Für welche Art der Textstrukturierung man sich auch entscheidet, die verschiedenen Ebenen der Hierarchien von Texten lassen sich in jedem Fall sehr weit aufgliedern. Die Makrostruktur eines Textes kann z. B. Kapitel enthalten, die aus Abschnitten bestehen, die wiederum aus Unterabschnitten bestehen, die aus funktionalen Einheiten bestehen und so weiter. Badenbrink und Ried [2001, S. 103] schlussfolgern daher: „Texte sind Bäume“.

Die Sequenz der Kapitel bzw. Hauptäste in der Makrostruktur beschreibt u. a. Juhl [2002, S. 22f]. Er unterscheidet die folgenden fünf Inhalte als Kern einer Anleitung, denen er die verschiedenen Kapitel zuweist:

- Leistungsbeschreibung
  - Bestimmungsgemäßer Gebrauch
  - Technische Daten
- Gerätebeschreibung
  - Lieferumfang
- Tätigkeitsbeschreibung
  - Sicherheitshinweise
  - Wartungsplan
  - Fehlersuchtabelle

- ggf. Funktionsweise (nur notwendig, wenn der Benutzer die Funktionsweise verstehen muss)
- ggf. Technische Unterlagen
  - Konstruktionszeichnungen
  - Schaltplan
  - Ersatzteilliste

Die Elemente „Impressum“, „Sicherheitshinweise“ und „Vorwort“ liegen dabei noch vor dem Kernbereich der Anleitung zwischen den Ordnungselementen „Titel“ und „Inhaltsverzeichnis“ [vgl. Juhl 2002, S. 22].

Juhl [2002] gibt Hinweise für den praktischen Einsatz von Ordnungselementen zum Strukturieren von Technischen Dokumentationen. Ordnungselemente sind hilfreich für die Orientierung des Anwenders, der unter Umständen spezifische Informationen sucht und die Anleitung nicht von vorne nach hinten durchliest [Juhl 2002, S. 97]. Mit den Ordnungselementen werden praktische Hinweise zur Konzipierung eines einheitlichen Seiten- und Textlayouts gegeben. Damit kann eine einheitliche Gestaltung z. B. von Seitenzahlen, lebendem Kolumnentitel oder der Zusammenfassung am Anfang des Kapitels erreicht werden [vgl. Juhl 2002, S. 97–119].

1.1.3 Dokumentationsprozesse

Aus den bisherigen Ausführungen über die vielfältigen Anforderungen an Technische Dokumentationen lässt sich schließen, dass ein enormer Organisationsaufwand besteht, um innerhalb einer Technischen Redaktion diese Qualitätsstandards umzusetzen.

Ein wichtiges Konzept für Qualitätsmanagement ist der Redaktionsleitfaden. Er gewährleistet eine einheitliche Struktur und Gestaltung für die verschiedenen Ausprägungsformen der Technischen Dokumentation eines Unternehmens oder eines Dienstleisters in diesem Bereich. Im Redaktionsleitfaden werden die verschiedenen Vorgaben, Festlegungen und Entscheidungen entsprechend einem Style-Guide für die Anfertigung von Anwenderdokumentationen gesammelt festgehalten. Die Mitarbeiter der Dokumentationsabteilung unterstützt er so beim Erstellen qualitativ hochwertiger Dokumentationen [Romberg 2000, S. 43]. Die folgenden Aufgaben des Redaktionsleitfadens stellt Romberg [2000] zusammen:

- Beschreibung der Arbeitsabläufe und Verantwortlichkeiten bei den Prozessen innerhalb der Dokumentationsabteilung
- Beschreibung der verwendeten Werkzeuge und ihrer Bedienung, besonders wenn spezifische Anpassungen durchgeführt wurden
- Beschreibung der messbaren Qualitätsmaßstäbe für die Arbeitsergebnisse
- Beschreibung der verbindlichen Gestaltungsregeln in Bezug auf Sprache, Inhalte und Struktur

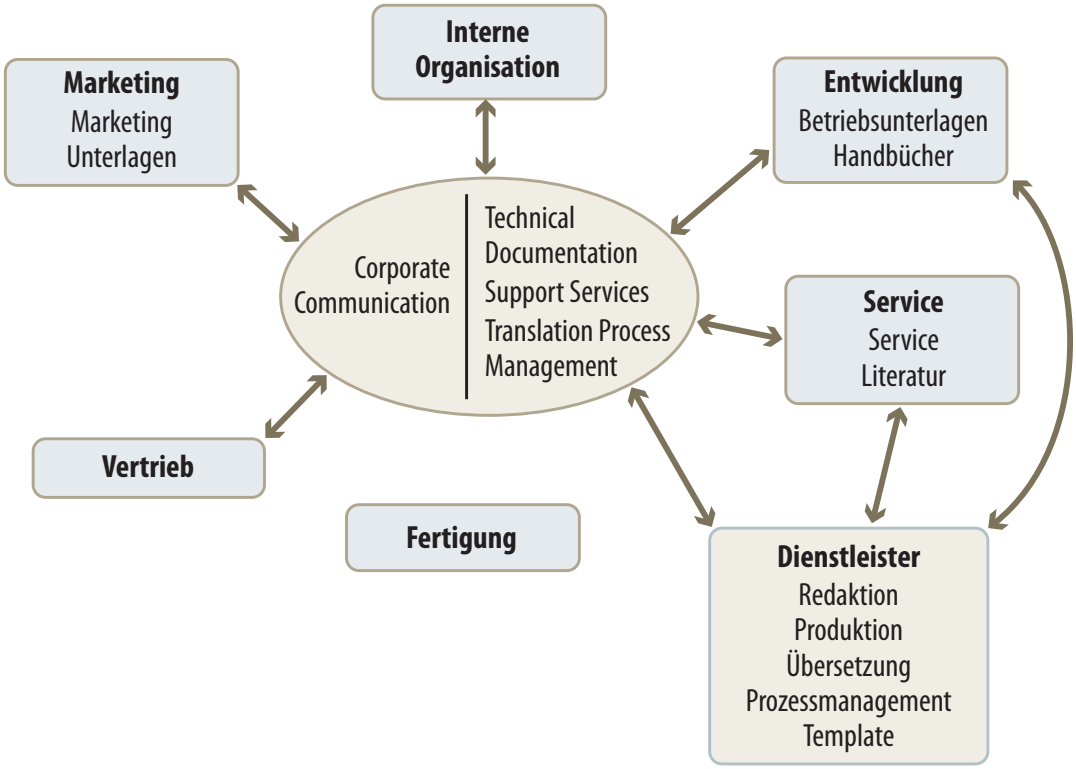


Abbildung 1.5: Durch eine abteilungsübergreifende Unterstützung lassen sich Corporate Identity und Corporate Design so positionieren, dass eine kontinuierliche Kommunikationsstruktur entsteht
[Quelle: Gust und Plattner 2003, S. 33]

Innerhalb der Dokumentationsabteilung sollen die Arbeitsabläufe verbessert und nach verschiedenen Kompetenzen Verantwortlichkeiten zugewiesen werden. Dies spiegelt sich auch in einer steigenden Qualifikation der Technischen Redakteure wieder [vgl. Straub 2006]. In Bezug auf die Dokumentationsabteilung gibt es aber in vielen Unternehmen erhebliche Defizite. Produktinformationen müssen aus fast allen Unternehmensbereichen zusammengetragen werden, diese Informationsprozesse sind jedoch oft nicht optimiert [Ried 2003]. Gust und Plattner [2003] führen aus, dass sich die Bedeutung der Technischen Dokumentation meist nicht in der Unternehmensstruktur widerspiegelt. Abgekoppelt vom Produktentwicklungsumfeld und der Marketingabteilung ungeordnet kann die Dokumentationsabteilung ihr volles Potenzial nicht entfalten.

Edgar Glaser führt in Hentschel [2006, S. 19] auf, dass ein vom Käufer konfigurierter Audi etwa alle 10.000 Modelle ein zweites Mal vorkommt. Das bedeutet für die Technische Dokumentation niedrige Auflagen, hohen Termindruck und hoch granulare Inhalte. Insgesamt lässt sich sagen, dass die Anzahl der Funktionen von Produkten immer weiter zunimmt. Die Produkte werden außerdem hinsichtlich Mechanik und Funktionalität modular aufgebaut. Dadurch entsteht eine große Variantenvielfalt in Verbindung mit immer kürzeren Innovationszyklen. Gleichzeitig ist der Anwender oft vom Funktionsumfang überfordert, weil er die technologischen Zusammenhänge der benutzten Produkte nicht erkennen kann [Romberg 2000, S. 1f].

Für die Technische Dokumentation eines Unternehmens bedeutet das eine Heraus-

forderung, die nur durch eine vollständige Integration der Dokumentationsabteilung in die Geschäftsprozesse zu bewerkstelligen ist [Hentschel 2006, S. 19]. In vielen Unternehmen herrscht noch immer die Sichtweise vor, dass nach der Entwicklung eines Produktes die eigentliche Arbeit getan ist und die Technische Dokumentation danach ein notwendiges, ungeliebtes Anhängsel darstellt [Romberg 2000, S. 7]. Das lässt sich mit den neuen Marktbegebenheiten – gerade im Hightech-Sektor – jedoch nicht vereinbaren. Dort muss in kürzesten Zyklen von der Entwicklung bis zur Marktreife eines Produktes kalkuliert werden. Nach wenigen Wochen kann die neueste Innovation schon überholt sein. Ein zusätzlicher Monat zum Fertigstellen einer Anleitung ist im Hinblick auf den extremen Kostendruck deshalb schlichtweg zu teuer [Gust und Plattner 2003, S. 31].

Als die drei größten Herausforderungen geben die in einer Umfrage zu Tools und Prozessen in Technischen Redaktionen befragten Redakteure an [Hurst 2006]:

1. Inhalte wiederverwerten (63,4 %)
2. Konsistenz beim Schreiben wahren (53,7 %)
3. Lokalisierungskosten minimieren (46,3 %)

1.2 Unterstützung des Redaktionsprozesses durch moderne EDV

Die moderne elektronische Datenverarbeitung (EDV) bietet passende Konzepte, mit denen sich die bisher aufgezeigten Qualitätsstandards verwirklichen lassen. Weil die Entwicklung der Informationsgesellschaft mit ihrem noch jungen Teilgebiet der Technischen Dokumentation nicht losgelöst von der Entwicklung von Hard- und Software im EDV-Bereich betrachtet werden kann, sind diese Qualitätsstandards untrennbar mit den Konzepten in der EDV verbunden. Elektronische Datenverarbeitung ist die Grundlage für die computerunterstützte Arbeit des Technischen Redakteurs. Möglichkeiten zur Effizienzsteigerung hängen von deren Entwicklungsstand ab und sind zum Teil erst durch verbesserte EDV erkannt worden. Softwareinnovationen haben viele Aspekte der Qualität von Anwenderdokumentationen überhaupt erst messbar gemacht.

1.2.1 Objektorientierung

Eine zentrale Weiterentwicklung in der modernen Informationstechnik ist die Effizienzsteigerung der Softwareentwicklung durch sogenanntes „Software Engineering“. Programme werden entgegen dem Bild aus der Gründergeneration der Programmierer nicht länger von einzelnen verrückten Genies geschrieben. Statt dessen werden Grundstrukturen der Software sorgfältig geplant und überwacht. Dadurch werden die Programmabläufe modularisiert. Die verschiedenen Programmmodule funktionieren unabhängig voneinander und sind mehrfach in verschiedenen Anwendungen nutzbar. Das vereinfacht die Anpassung, Erweiterung, Wartung und Wiederverwendung von Softwaremodulen. Durch die sogenannte Kapselung der Daten wird dabei eine bessere Trennung von Anwendungen und Daten erreicht [Bienert 1998, S. 201–203].

Ein Problem des wachsenden Funktionsumfangs von Software ist, dass es zu Überschneidungen ähnlicher oder identischer Teilfunktionen verschiedener Anwendungsprogramme kommt. Diese würden bei fester Implementierung redundant vorliegen, die Programmgröße würde zunehmen und die verschiedenen Anwendungen bräuchten längere Zeit zum Laden. Damit nicht alle Programmkomponenten, die in einer Software benötigt werden, fest implementiert werden müssen, gibt es ein Konzept zur Einbindung gerade benötigter Komponenten zur Laufzeit. Dieses „dynamic linking“ muss vom Betriebssystem bereitgestellt werden und wird über Programmkomponenten, die Dynamic Link Libraries (DLLs) genannt werden, eingebettet [Bienert 1998, S. 204f].

Eine modular aufgebaute Anwendung erzeugt auf der Basis der ihr verliehenen Fähigkeiten einen ganz speziellen Typus von Daten oder Dokumenten. Dieser ist nicht mit dem Dateiformat anderer, aus anderen Komponenten aufgebauten Anwendungen, kompatibel und kann daher z. B. nicht einfach in ein übergeordnetes Element eingebettet werden. Eine Lösung dieses Problems ist, dass an die Daten aus einer Anwendung, die zur Verarbeitung nötigen Komponenten „angehängt“ werden. Die Trennung von Programmen (ausführbare Dateien) und Daten (statische Dateien) wird dadurch aufgehoben [Bienert 1998, S. 205].

Ein weiterer Schritt in diese Richtung ist die Objektorientierung. Objektorientierung heißt, dass die Daten und die zu ihrer Bearbeitung nötigen Programme in logische Einheiten, sogenannte Objekte integriert werden. Dadurch können diese Objekte miteinander kommunizieren, indem die zu einem Objekt gehörenden Anwendungen aufgerufen werden [Bienert 1998, S. 206].

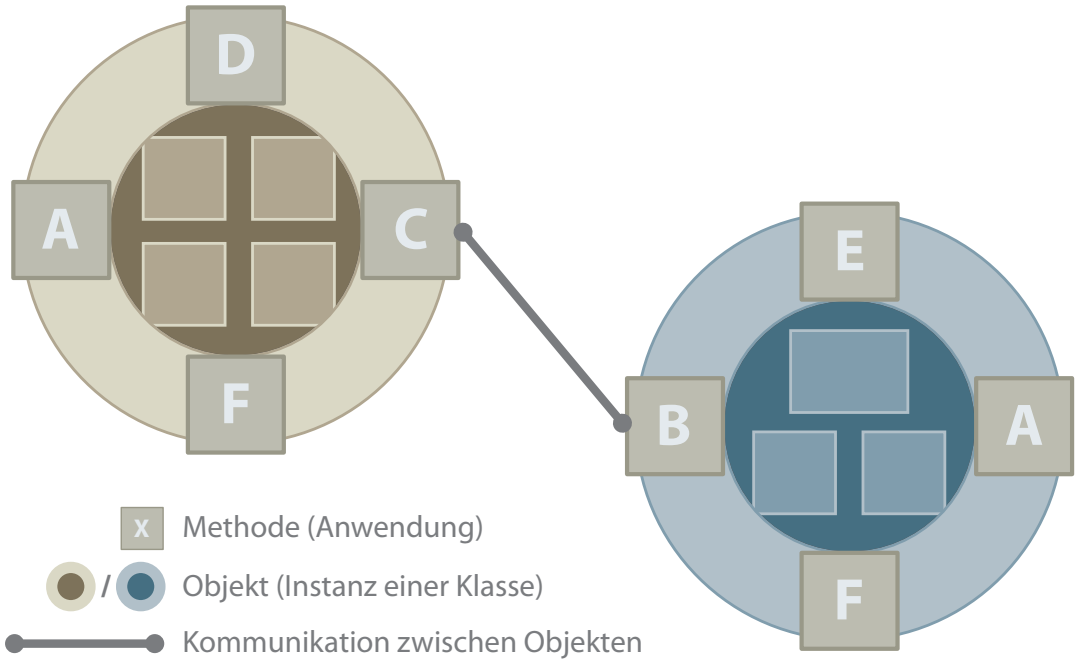


Abbildung 1.6: Objektorientierung [Quelle: Bienert 1998, S. 207]

Objekte sind geschlossene Einheiten logisch verknüpfter Daten und Programme. Die Programme nennt man „Methoden“, die Daten „Variablen“ eines Objekts. Objektmodelle sind zur besseren Simulation der Realität entwickelt worden. Objekte sind in Klassen organisiert, deren Eigenschaften (Variablen und Methoden) hierarchisch an

Unterklassen vererbt werden können. Die Variablen der Objekte einer Klasse sind definiert und ein konkreter Wertesatz für diese Variablen stellt ein Objekt-Exemplar, eine sogenannte Instanz dieser Klasse dar [Bienert 1998, S. 208f].

Objekte können andere Objekte als Bestandteil haben (composite objects). Komplexe Anordnungen von Teilkomponenten, z. B. von Maschinen, können so simuliert werden [Bienert 1998, S. 210].

1.2.2 Universelle Dokumentenformate

Die Objektphilosophie hat einen sehr großen Einfluss auf die aktuellen Trends im Bereich der Informationstechnik. Das Objektmodell eignet sich nicht nur als Realitätsmodell, sondern auch als Informationsmodell. In diesem Sinne wurde die Standard Generalized Markup Language (SGML) entwickelt. SGML ist eine universelle Dokumentensprache, die Layoutbeschreibung und Ordnungsfunktionen verbindet. Dokumente werden als Sammlung komplexer Objekte verstanden, die im Sinne einer objektorientierten Datenbank verwaltet werden [Bienert 1998, S. 218f].

Ein universelles Dokumentenformat ist nötig, weil die Fähigkeiten von Textverarbeitungs- und Desktop-Publishing-Programmen, Daten zu organisieren oder gar Prozesse abzubilden nicht oder nur rudimentär vorhanden sind [Bienert 1998, S. 217]. Der Austausch von Dokumenten zwischen verschiedenen Anwendungen mit sogenannten proprietären Fileformaten gestaltet sich schwierig, da hier kein Standard für die Darstellung logischer Ordnungsprinzipien existiert [Bienert 1998, S. 155]. Das im vorigen Abschnitt 1.2.1 erwähnte „Anhängen“ der für die Verarbeitung benötigten Programmkomponenten an die statischen Daten ist keine Ideallösung, da sie dazu führt, dass diese Programmkomponenten ständig redundant gespeichert werden müssen und deren Ausführbarkeit außerdem abhängig vom Betriebssystem ist. Bienert [1998, S. 155] fordert darum:

> Moderne digitale Dokumentenformate sind portabel, also unabhängig von Betriebssystemen und Anwendungsprogrammen und integrieren Text, Grafik, Bild und Ton als mögliche Erscheinungsformen für Inhalte.

Durch die Entwicklung des Internets hat sich eine einfache, auf SGML basierende Auszeichnungssprache für Dokumente, die Hypertext Markup Language (HTML), weltweit durchgesetzt [Behme und Mintert 2000, S. 41]. Während HTML von jedem Internetbrowser mehr oder weniger gut angezeigt wird, hat sich SGML im Bereich der Printdokumente nicht als Standard etabliert. Zwar gab es hier z. B. die Text Encoding Initiative (TEI) und mit DocBook eine umfangreiche Beschreibungssprache, SGML ist seit 1986 sogar ISO-Standard, die Unterstützung der Softwareindustrie blieb aber weitgehend aus. Die Komplexität von SGML schreckte wohl viele Entwickler ab, so dass das Akronym SGML oft scherzhaft als „sounds great maybe later“ aufgelöst wird [Tidwell und Lichtenberg 2003, S. 4].

Die Fähigkeiten von SGML sollten Einzug in das Internet halten, wo HTML mit seinen wenigen Elementen zwar auf SGML basiert, aber wenig Flexibilität bietet. Deshalb wurde eine abgespeckte Variante von SGML durch das World Wide Web Consortium (W3C) für den Einsatz im Web entwickelt. Diese als „Web-SGML“ in den ISO-Standard integrierte und 1998 vom W3C verabschiedete Beschreibungssprache für Dokumentenauszeichnungssprachen wurde „Extensible Markup Language“ (XML) getauft. Der

ISO-Standard SGML wurde 1997 so angepasst, dass XML eine Untermenge von SGML darstellt [Rath 2006; Behme und Mintert 2000, S. 47].

Mit XML ist das universelle Dokumentenformat de facto Realität geworden. Es gibt immer mehr XML-basierte Software und Austauschformate für verschiedenste Daten. Für XML gibt es zahlreiche begleitende Standards, die ebenfalls vom W3C entwickelt wurden. Dazu gehören z. B. die „Extensible Stylesheet Language“ (XSL) und „XSL-Transformations“ (XSLT), die für die entwickelte Anwendung von Bedeutung sind. Für die Technische Dokumentation ist vor allem interessant, dass XML Unicode-basiert ist und damit alle gängigen Schriftsprachen der Welt dargestellt werden können. Das ermöglicht einen einfachen Austausch von Daten für Übersetzungen [Rath 2006].

Da XML, wie SGML, eine universelle Sprache zur Beschreibung von Dokumentenformaten ist, muss geprüft werden, welche bisher entwickelten XML-Formate für die Technische Dokumentation einsetzbar sind. Als die wichtigsten Standards in diesem Bereich führt Rath [2006] DocBook, DITA und mumasy auf. Die genannten Sprachen lassen sich aber kaum nutzen, ohne dass eigene Anpassungen vorgenommen werden. Durch Änderungen der Document Type Definition (DTD) oder des XML-Schemas lässt sich das aber erreichen.

Die DTD ist die aus SGML übernommene Beschreibung des jeweiligen Dateiformates. Die erlaubte Reihenfolge und Hierarchie der Elemente wird in der DTD-Datei festgelegt. Die Syntax ist relativ einfach, so dass es möglich ist, eine eigene XML-Struktur mit einer DTD zu beschreiben. XML-Schema ist der vom W3C entwickelte Nachfolger der Dokumenttyp-Definition mithilfe einer DTD. Im Gegensatz zur DTD ist die Struktur einer XML-Schema-Datei selbst XML-basiert, kann so also leichter durch XML-Parser verarbeitet werden. Mit XML-Schema kann man dem Inhalt von Elementen außerdem Datentypen zuweisen, was bei DTDs sehr eingeschränkt und nur für Attribute möglich ist. Da eine XML-Schema-Datei eine hohe Verschachtelungstiefe aufweist, lässt sie sich nicht so leicht mit einem beliebigen Texteditor entwickeln, wie eine DTD-Datei. Es gibt aber spezielle Editoren für XML-Schemas, die das Erstellen vereinfachen.

Das Entwickeln einer eigenen Struktur, die den eigenen Anforderungen angepasst wird, ist mit XML möglich. Die Ergebnisse des Autors zeigen, dass mithilfe von XSLT XML-Dateien relativ unkompliziert in andere (Standard-)XML-Formate umgewandelt werden können. Die Standard-Formate sind nicht genau auf die Bedürfnisse der Technischen Dokumentation zugeschnitten. Ihnen fehlt z. B. ein Modell für die Beschreibung der Informationssequenz in Sicherheitshinweisen (Gefahr – Folgen – Abwenden). Statt dessen sind Elemente oft nach ihrer Formatierung benannt (Überschrift, Absatz, etc.). Das kann im Zusammenhang mit der Dokumentbeschreibung sinnvoll sein, hier werden aber wertvolle Potentiale von XML verschenkt, denn sprechende Bezeichnungen der Element-Tags (z. B. Sicherheitshinweis, Gefahr, Folge) erleichtern dem Redakteur die Arbeit.

Ein kritischer Punkt bei der Beschreibung der Struktur ist, ein funktionierendes Tabellenmodell zu definieren. Hierbei sollte man sich an existierenden Standards, z. B. dem von DocBook verwendeten CALS-Tabellenmodell [vgl. OASIS 1995], orientieren und diese entsprechend den eigenen Vorstellungen erweitern. Die Ergebnisse des Autors zeigen, dass Tabellenmodelle die am komplexesten verschachtelten Elementstrukturen besitzen. Tabellen lassen sich mithilfe von XSLT nicht so leicht in andere Formen

umwandeln wie andere XML-Strukturen. Das gilt insbesondere dann, wenn eine spaltenbasierte Tabellenstruktur in eine reihenbasierte umgewandelt werden soll.

1.2.3 WYSIWYG und Single-Source-Publishing

Bei der Umstellung auf einen rein XML-basierten Redaktionsprozess kann man normalerweise nicht mehr gleich am Bildschirm erkennen, wie das geschriebene Wort auf der Seite platziert wird, das gedruckte Ergebnis also aussehen wird. Damit arbeitet man aber nicht mehr nach dem Prinzip, das die Bearbeitung von Printdokumenten am Bildschirm, das sogenannte Desktop-Publishing (DTP), überhaupt erst populär gemacht hat. Dieses Prinzip wird WYSIWYG genannt. Das Akronym bezeichnet dabei „What you see is what you get“ („Was du siehst, ist [das] was du bekommst“) [Wikipedia 2007c]. Desktop-Publishing wurde mit diesem Prinzip von den Firmen Apple (Macintosh), Adobe (PostScript), Aldus (PageMaker) und Linotype (erste PostScript-Schriften und erster PostScript-fähiger Belichter) im Jahr 1985 eingeführt und revolutionierte damit alle Arbeitsprozesse im Bereich Satz und Druck. Seit etwa 1992 werden Printprodukte fast ausschließlich im Rahmen von DTP produziert [Wikipedia 2007a].

Im Rahmen des Single-Source-Publishing mit Redaktionssystemen kann nun nicht mehr unbedingt nach dem WYSIWYG-Prinzip gearbeitet werden. Single-Source-Publishing steht für die Mehrfachverwendung von Quellinformationen. Oft wird diese Methode auch mit dem Cross-Media-Publishing in Verbindung gebracht, also dem Publizieren der Informationen der Ausgangsdokumente in verschiedenen Medien [Ziegler 2004, S. 20]. Diese Methoden werden, gerade bei einer Modularisierung der Quellinformationen (siehe 1.2.5 auf der nächsten Seite), immer wieder indirekt für den Mangel an WYSIWYG verantwortlich gebracht. Es wird argumentiert, dass die XML-Bausteine, die der Redakteur erstellt, nun besser strukturiert seien und durch die automatische Konvertierung in das Printformat – z. B. mithilfe eines XSL-FO-Prozesses – der Arbeitsschritt des „Layoutens“ wegfalle, das Publizieren also effizienter werde. Unschöne Seitenumbrüche können nicht vermieden werden, eine Korrektur am Endlayout sei nicht direkt möglich, ohne dass der XSL-FO-Prozess nochmals durchlaufen und das Ergebnis am Bildschirm mithilfe des PDFs kontrolliert werde [Lüthy und Wetzl 2001, S. 73f]. Der Redakteur kümmere sich nur noch um die Inhalte und dadurch steige deren Qualität. WYSIWYG sei für die Qualität der Publikation eher kontraproduktiv, da so auch keine Konsistenz der Formate gewährleistet sei, der Redakteur müsse sich an den neuen Workflow anpassen [vgl. Lüthy und Wetzl 2001, S. 75f].

Dieser Argumentation kann der Autor nicht zustimmen. Weder der Export von Daten in verschiedene Zielformate, noch die konsistente Verwendung von Formatvorlagen ist durch die Einführung von XML erfunden worden. XML als Austausch- und Speicherformat für die Technische Dokumentation hat erhebliche Vorteile und ist an keine bestimmte Software gebunden. Von einem XML-Editor innerhalb eines Redaktionssystems sollte man aber erwarten können, dass dieser eine Druckvorschau anzeigt, mit der man das Papierlayout kontrollieren kann, schließlich handelt es sich immer noch um die wichtigste mediale Ausprägungsform der Technischen Dokumentation. Die reinen XML-Editoren können gerade das jedoch nicht [vgl. Bergerhoff 2006]. Statt dessen zeigen sie maximal ein Weblayout an, das für die Publikation im Netz nicht verbindlich ist, da jede Browser-Software eine Webseite anders anzeigt. Für die Printproduktion macht diese Weblayout-Anzeige gar keinen Sinn, da ein Anwender auf dem

Papier nicht scrollen kann, sondern umblättern muss. Es ist durchaus möglich, dass ein ungünstiger Seitenumbruch z. B. bei Handlungssequenzen dabei zu Irritationen führt. Bei solchen Problemen kann der Redakteur im WYSIWYG-Modus einer DTP-Anwendung gegensteuern, indem er z. B. Abbildungen kleiner oder größer skaliert und Texte so umformuliert, dass sie auf der Seite günstig angezeigt werden. Diese Möglichkeiten müssen auch in XML-Anwendungen gegeben sein, damit Technische Redakteure XML auch wirklich als vorteilhaft akzeptieren können.

1.2.4 Translation-Memory-Systeme, Terminologiemanagement

Die Arbeitsbedingungen für Übersetzer haben sich in den letzten Jahren durch die flächendeckende Einführung von Translation-Memory-Systemen (TM-Systemen) entscheidend verändert. Im Bereich der Computer Aided Translation (CAT) stellen TM-Systeme eine intelligente Hilfe dar. Alle Sätze, die bereits übersetzt wurden, werden von ihnen gespeichert und wieder vorgeschlagen, wenn der zu übersetzende Text sich wiederholt. Das passiert auch, wenn der Text nicht ganz identisch ist („Fuzzy-Matches“). Eine Terminologiefunktion zeigt außerdem bei festgelegten Begriffen die vorgegebene Übersetzung an.
Anders als bei maschinellem Übersetzen (Machine Translation), bei dem Software ganze Texte automatisch und meistens sehr eigenwillig übersetzt, trifft die letztendlichen Entscheidungen beim Übersetzen dabei immer noch ein Mensch. Bei Übersetzungen mit Unterstützung von TM-Systemen ergeben sich Chancen für eine verbesserte Qualität und Konsistenz bei gleichzeitiger Verringerung der Übersetzungszeit und der Kosten [Massion 2006, S. 30].

Gerade für ein XML-basiertes Redaktionssystem sind Übersetzungen mithilfe von TM-Systemen interessant, da die wichtigsten Systeme XML-Dateien problemlos verarbeiten können [vgl. Massion 2006, S. 32]. Die Einsparungseffekte lassen sich allerdings nur dann voll ausschöpfen, wenn die zu übersetzenden Texte über eine einheitliche Terminologie verfügen, die am besten mit dem verwendeten TM-System abgeglichen wird [vgl. Schmidt 2007a]. Terminologieverwaltung ist ein komplexes Thema, das z. B. von Schmitz [2001] näher erörtert wird.

In der entwickelten XML-Anwendung findet keine Prüfung der Terminologie oder Kontrolle der Sprache statt. Man könnte dieses Feature zumindest rudimentär mit dem Programm „Duden Korrektor“ in FrameMaker implementieren [vgl. Gust 2006]. Eine weitere Möglichkeit ist die Nutzung eines zusätzlichen, leistungsfähigeren Sprachprüfprogramms [vgl. Reuther 2006, S. 55].

1.2.5 Content-Management-Systeme

Content-Management-Systeme (CMS) in der Technischen Dokumentation dienen dazu, die Erstellungsprozesse der Dokumentationen zu optimieren und zu automatisieren [Lüthy und Wetzl 2001, S. 67]. Der Begriff Dokumenten-Management-System (DMS) wird oft synonym verwendet. Im Web bezeichnet ein CMS ein System zur Realisierung und Pflege umfangreicher Webauftritte. Außerdem gibt es noch aus dem Bereich der Betriebswirtschaft das Produktdaten-Management (PDM) und das Enterprise-Content-Management (ECM), wo ähnliche Systeme zum Einsatz kommen können. All

diese Systeme verbindet die interne, zentrale Speicherung der Daten in einer auf einem Server liegenden Datenbank.

Für die Technische Dokumentation unterscheidet Holzmann [2004] die folgenden Systemansätze:

- Einfache Dokumenten-Management-Systeme
  - Verwaltung und Versionierung beliebiger Dateien
- Erweiterte Dokumenten-Management-Systeme
  - Verwaltung und Versionierung beliebiger Dateien
  - Erweiternde Komponenten, z. B. für Prozesssteuerung, Web-Publishing, Verwaltung von XML-Daten
  - Keine Erzeugung von Dokumenten aus Modulen
- Web-Content-Management-Systeme
  - Erstellung von Websites
  - Meist Verwaltung von HTML-Dateien und Grafiken
  - Meist Schnittstellen zu Enterprise-Resource-Planning (ERP) oder zu Shop-Systemen
  - Datenhaltung in XML
  - Keine Versionierung
- XML-Component-Management-Systeme
  - Verwaltung strukturierter, modularisierter XML-Dokumente
  - Speziell auf die Technische Dokumentation zugeschnitten
- Allgemeine SGML/XML-Systeme
  - Überwiegend Verwaltung von SGML/XML-Daten
  - Verwaltung auf der Ebene von SGML bzw. XML
  - Maximale Granularität der verwalteten Informationen
  - Basiert häufig auf objektorientierter Datenbank

Zusätzlich identifiziert Holzmann [2004, S. 27] zwei Sonderfälle; zum Einen „Wordzentrierte-Management-Systeme“, die auf Word-Dokumenten aufbauen und zum Anderen „Vollständig als relationale Datenbank aufgebautes Systeme“ mit einem sehr variablen und anpassbaren Publikationsprozess in verschiedene Zielformate (auch XML). Als Sonderfälle werden diese Systemansätze deshalb bezeichnet, weil es nur wenige oder einen Anbieter dieser Vertreter auf dem Markt gibt. Holzmann [2004, S. 28–30] charakterisiert weiterhin einige angebotene Vertreter von CMS/DMS und analysiert deren Tauglichkeit für die Technische Dokumentation.

Schmidt [2006] zeigt anhand von Expertenmeinungen, wie sich mithilfe von CMS die Kommunikationsaufgaben eines Unternehmens besser bewältigen lassen. Deutlich wird, dass Art und Implementierung des CMS dabei jedoch immer konkret vom Einzelfall abhängen und der Aufwand dabei nicht unterschätzt werden darf. Bei Ried

[2003] wird deutlich, wie vor dem Einsatz des CMS die Informationsprozesse eines Unternehmens analysiert und optimiert werden müssen, damit die Einführung von Content-Management-Systemen auch zu Effizienzsteigerungen führt. Der Zeitraum für die Evaluation und Einführung eines Systems wird von Ambrus [2004, S. 32] auf etwa ein Jahr bemessen. Die Migration von Daten und Prozessen in die neue CMS-basierte Arbeitsweise muss ebenfalls bewerkstelligt werden.

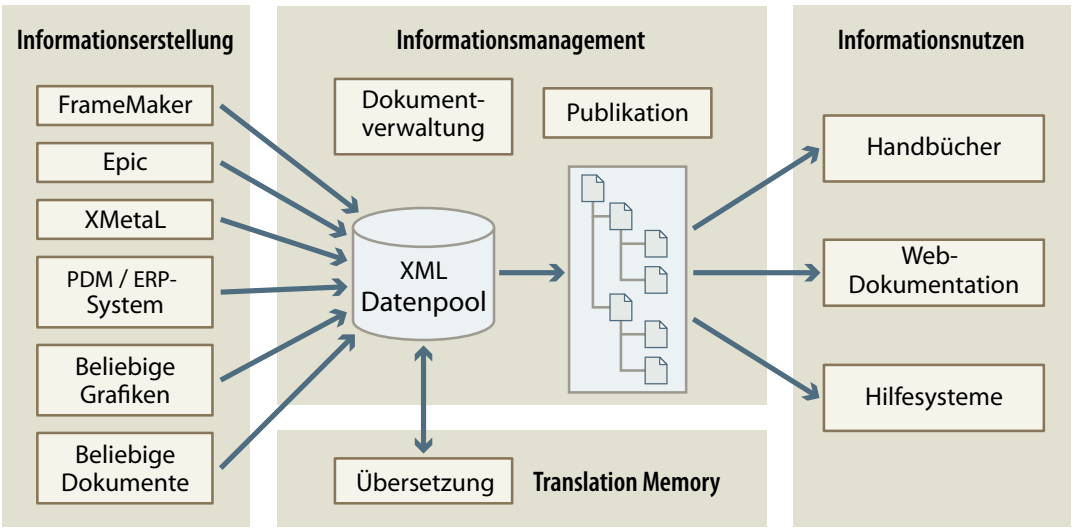


Abbildung 1.7: XML-Component-Management-System
[Quelle: Holzmann 2004, S. 26]

Grundlegend soll ein CMS die Dokumentationsabteilung bei der Erfassung, Verwaltung und Publikation von Informationen unterstützen. Dabei werden die Informationen zentral in einer Datenbank angelegt. Beherrscht das System eine Versionierung der Daten, bietet das den Vorteil, dass in der Datenbank alle gespeicherten Versionsstände der Dokumente abgelegt werden. Das heißt, dass ein gerade wiederholt gespeichertes Dokument auch auf den Zustand bzw. die Version vor dem Speichern zurückgesetzt werden kann. Im Idealfall gilt dies nicht nur für den letzten Speichervorgang, sondern für alle Speichervorgänge. Das Zurücksetzen der Version lässt sich so ebenfalls zurücksetzen. Möglich wird dies dadurch, dass die Datenbank anstelle aller Daten eines Dokuments nur noch die Veränderungen der Daten eines Dokumentes speichert.

Ein weiterer Mehrwert eines CMS soll darin bestehen, dass Informationen aus einem Dokument leicht in ein anderes Dokument übernommen werden. Dazu werden die Informationen für Technische Dokumentationen in Modulen (meist XML-Komponenten) verwaltet, aus denen dann neue Dokumente zusammengestellt werden können. Der kritische Punkt beim Einrichten eines CMS besteht allerdings in der Wahl einer geeigneten Modulgröße. Schlenker [2006] macht eine Hochrechnung auf der Basis einer Modulgröße auf Absatz- bzw. Listenpunktebene, mit der sich Redundanzen wirkungsvoll vermeiden lassen und kommt zu dem Schluss, dass die Wiederverwendung und Variantensteuerung dieser Module zu einer durch den Redakteur kaum zu bewältigenden Modulflut führt. Dadurch wird dieser scheinbare Vorteil eines CMS wieder aufgehoben, da die Modulverwaltung nicht weniger aufwendig ist als das klassische „Copy and Paste“-Verfahren. Bei größerer Granularität der Module kann es durchaus zu Vorteilen, z. B. bei der Verwaltung der Dokumentationen verschiedener Produktvarianten,

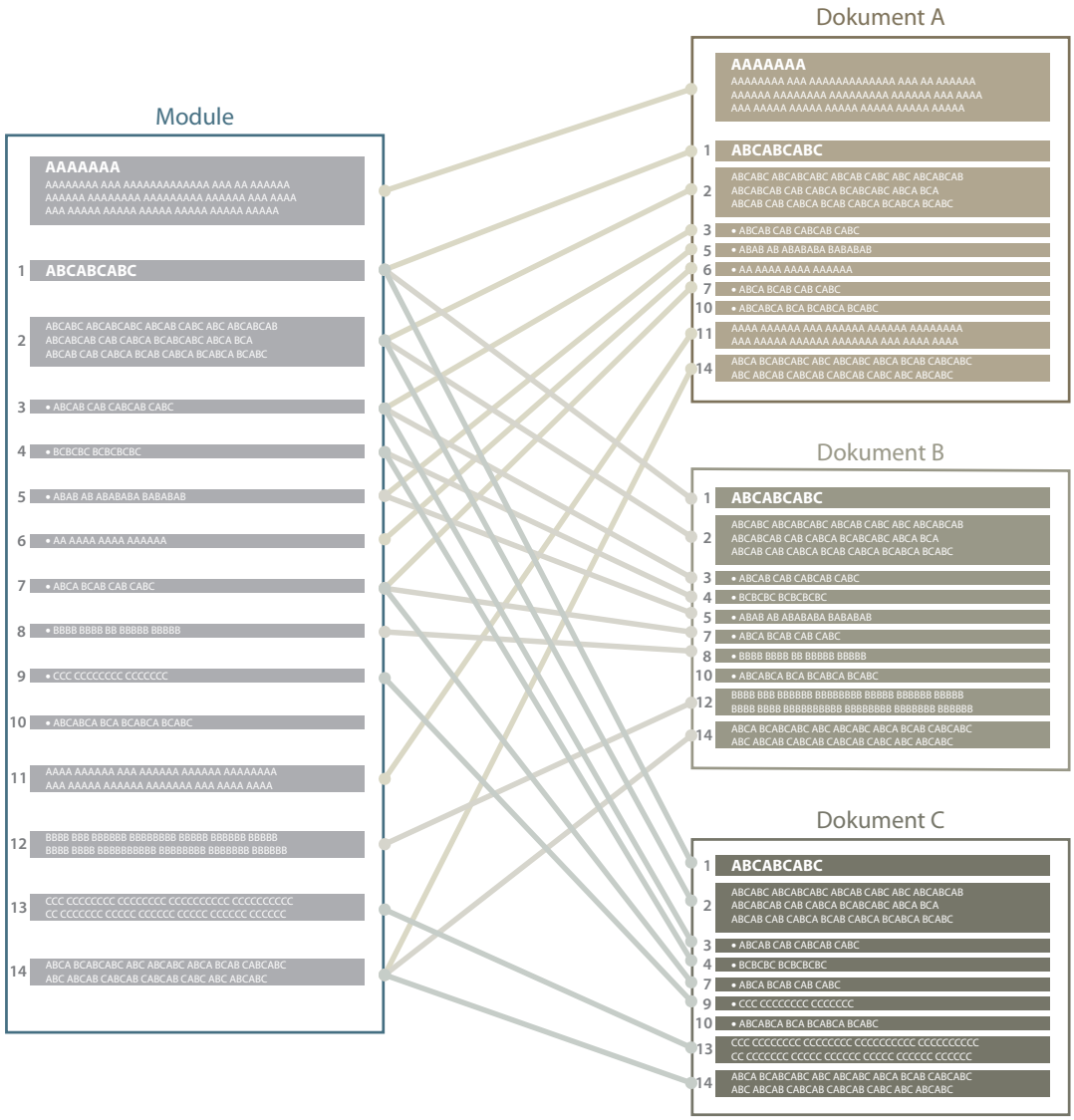


Abbildung 1.8: Schematischer Aufbau einer modularen Dokumentation mit drei Dokumentvarianten: bei feingranularen Modulen droht die Modulflut [Quelle: Schlenker 2006, S. 41]

kommen. Dienen die Module lediglich der besseren Strukturierung der Dokumente, lässt sich dies auch ohne den Einsatz eines CMS realisieren.

Entscheidender als die Mehrfachverwendung von Modulen ist die Implementierung von Sprachtechnologien wie Terminologiemanagement sowie Schnittstellen für verschiedene Translation-Memory-Systeme. Module können so zum Management des Übersetzungsworkflows dienen.

Wirklich sinnvoll kann die Einführung eines CMS für die Technische Dokumentation sein, wenn ein Unternehmen einen sehr großen Umfang an Technischer Dokumentation zu bewältigen hat und das CMS in die Gesamtprozesse des Unternehmens im Sinne einer zentralen Datenhaltung integriert wird. Anderenfalls bringt der Einsatz eines CMS einen für mittelständische Unternehmen kaum zu rechtfertigenden Aufwand der Implementierung mit sich.

1.3 Softwarelösungen in der Technischen Dokumentation

Im letzten Abschnitt wurde eine Vielzahl von Konzepten der modernen EDV aufgeführt, die für die Technische Dokumentation von Nutzen sein können. Für deren praktische Umsetzung stellt sich die Frage, in wie weit diese Konzepte bereits in Softwarelösungen realisiert und damit am Markt verfügbar sind. Es wird nun versucht zu klären, welche Softwarelösungen in Technischen Redaktionen im Einsatz sind und wie leistungsfähig diese Systeme in Bezug auf die Unterstützung der in Abschnitt 1.1 erarbeiteten Qualitätssicherheitsaspekte sind.

1.3.1 Eingesetzte Software

Zwei im Jahr 2006 durchgeführte Umfragen zur Softwarelandschaft in der Technischen Dokumentation kommen zu ähnlichen Ergebnissen. Am häufigsten wird demnach Microsoft Word eingesetzt. Bei der Umfrage der SDL International gaben dies 56 % der Teilnehmer an. An zweiter Stelle rangiert Adobe FrameMaker, dass 44 % der Befragten der SDL-Umfrage benutzen [vgl. Hurst 2006]. Die tekom-Softwareumfrage zeigt ein ähnliches Bild, differenziert aber genauer. In Dokumentationsabteilungen mit bis zu drei Mitarbeitern liegt Microsoft Word demnach mit einem Anteil von 73 % klar vorn, Adobe FrameMaker kommt mit 39 % auf Platz 2, die anderen aufgeführten Programme (InDesign, PageMaker, QuarkXPress, Quicksilver) spielen eine untergeordnete Rolle. In Abteilungen mit vier bis zehn Technischen Redakteuren liegen Word und FrameMaker fast gleich auf (Word 62 %, FrameMaker 55 %). Arbeiten schließlich mehr als zehn Redakteure in einer Abteilung, liegt FrameMaker knapp vorn (Word 52 %, FrameMaker 59 %) [Straub und Ziegler 2007, S. 33].

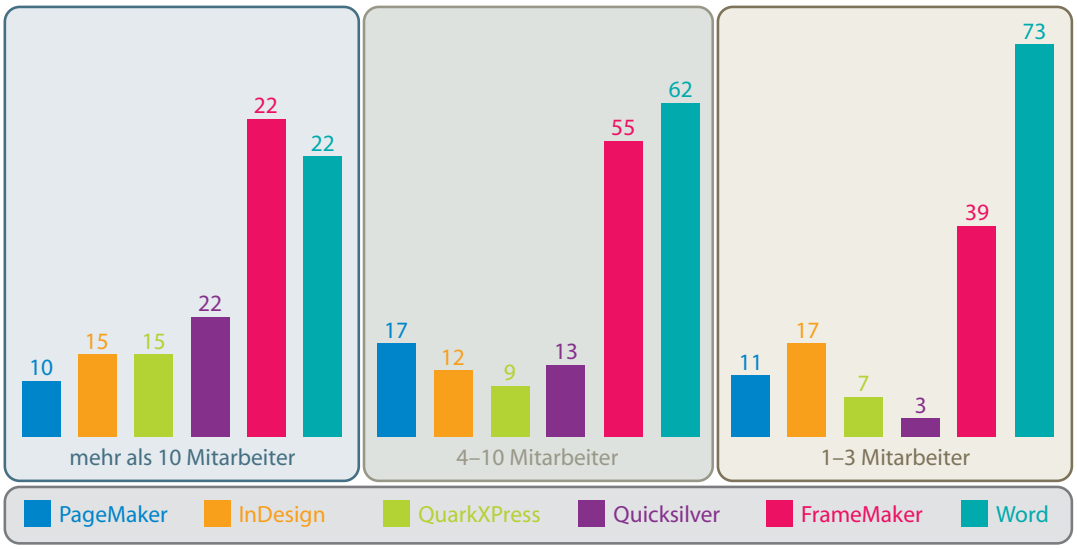


Abbildung 1.9: Eingesetzte Software in Abhängigkeit von der Abteilungsgröße [Quelle: Straub und Ziegler 2007, S. 33]

Somit ist Word auch nach der tekom-Umfrage insgesamt mit 65 % das Tool mit der höchsten Verbreitung. Straub und Ziegler [2007, S. 32] konstatieren, dass mit zunehmender Größe der Abteilung speziellere Software eingesetzt wird. In Redaktionen mit

mehr als zwei Mitarbeitern kommen zudem durchschnittlich zwei Textverarbeitungs-Programme zum Einsatz. Für komplexe Dokumentationen hat sich dafür Adobe FrameMaker etabliert. Die Ergebnisse der tekom-Umfrage zeigen zudem, dass große Dokumentationsabteilungen verstärkt auf Content-Management-Systeme bauen. Bei den Teilnehmern mit Redaktionsgrößen von mehr als zehn Mitarbeitern haben demnach bereits 77 % ein CMS im Einsatz [Straub und Ziegler 2007, S. 33f]. Ein ähnlicher Trend zeigt sich beim Einsatz von Translation-Memory-Systemen (TM-System) und Terminologiesoftwareprogrammen ab [Straub und Ziegler 2007, S. 34–36]. Zusätzlich kommen klassische Bildbearbeitungsprogramme (Photoshop 56 %, Illustrator 38 %, Corel Draw 35 %) zum Einsatz.

Die ähnlichen Umfrage-Ergebnisse in den beiden Artikeln werden sehr unterschiedlich interpretiert. Während Hurst [2006] titelt, dass sich XML noch im Wartestand befindet und dies an der geringen Verbreitung typischer XML-Autorenumgebungen (Arbortext Editor 9,8 %, Blast Radius XMetaL 4,9 %) festmacht, schlussfolgern Straub und Ziegler [2007, S. 36], dass der Einsatz von CMS, TM-Systemen und Terminologiesystemen sich deutlich auf Unternehmen mit mittleren und kleineren Redaktionsabteilungen verlagert. Die Umfrage der tekom ist dabei aktueller, stützt sich jedoch auf einen Online-Fragebogen. Straub und Ziegler [2007, S. 32] räumen ein, dass sie daher nicht repräsentativ sein kann.

1.3.2 Möglichkeiten und Grenzen

Je spezialisierter die eingesetzte Software ist, desto schwieriger wird es, abzuschätzen, was diese Systeme im Einzelfall in der Technischen Dokumentation zu leisten vermögen. Beim Einsatz eines genau an den eigenen Workflow angepassten CMS sollte nach Überwindung der Startschwierigkeiten bei der Einführung eine deutlichen Effizienzsteigerung feststellbar sein. Dies wird z. B. durch die Erfahrungsberichte von Lüthy und Wetzl [2001] und Ambrus [2004] bestätigt. Gleichzeitig ist der Ablauf der Prozesse unter Umständen sehr starr festgelegt und es kann weniger flexibel auf neue Anforderungen reagiert werden. Das gilt gerade dann wenn man das System nicht selbst warten kann, da hier im Unternehmen die Kapazitäten fehlen. Mallok [2006] warnt deshalb mit Verweis auf den entstehenden Implementierungsaufwand und die daraus resultierenden Kosten mittelständische Unternehmen vor der XML-Falle.

Betrachtet man die Ergebnisse der Umfragen, die in Abschnitt 1.3.1 vorgestellt wurden, wird deutlich, dass die meisten Unternehmen ihre Dokumentationsaufgaben noch nicht mittels angepassten Content-Management-Systemen, sondern mit Microsoft Word oder Adobe FrameMaker lösen.

Microsoft Word mit der höchsten Verbreitung wird von Dieter Gust in Schmidt [2007b, S. 28] schlicht als „geniale Katastrophe für die Technische Dokumentation“ bezeichnet. Andere Experten kommen im Artikel zu ähnlichen Schlussfolgerungen, konstatieren Word durchaus geniale Ansätze im Funktionsumfang, der Unicode-Unterstützung, Rechtschreibprüfung und der kompletten Anpassbarkeit, kritisieren aber, dass die meisten Redakteure die Funktionen mit den Voreinstellungen des Programms nicht effizient nutzen können. Die Nummerierungsfunktionen, Zentral- und Filialdokumente sind laut Gust wirklich katastrophal. Ohne Zusatzprogrammierung sind die seit der Version 2003 angebotenen XML-Funktionen nicht sinnvoll einsetzbar. Im gleichen

Sinn äußert sich Bergerhoff [2006, S. 31]: „Demgegenüber kann Word 2003 mit seiner XML-Funktionalität nicht wirklich guten Gewissens mit den anderen vorgestellten XML-Editoren verglichen werden.“ Problematisch für einen zukünftigen Einsatz des Programms ist die neueste Version Word 2007, die von den Experten für die Technische Dokumentation als ungeeignet bezeichnet wird [vgl. Gust 2007; Schmidt 2007b, S. 29].

Adobe FrameMaker wird häufig für umfangreiche Dokumentationen eingesetzt. Das Programm hebt sich beim Funktionsumfang für den Mengensatz deutlich von Word und auch von den gängigen DTP-Programmen ab. Mitschke und Schulze [2003, S. 20] nennen hier z. B. die ausgefeilte Buchfunktion, automatisch generierte Verzeichnisse und Listen, Tabellen, Formeln, Querverweise, Indizes und Hyperlinks. Sie betonen, dass sich viele Satzvorgänge mit FrameMaker automatisieren lassen und die Bearbeitung dabei seitenbasiert, d. h. nach WYSIWYG-Prinzip erfolgt. Die Automatisierungsfunktionen der gängigen DTP-Programme sind dagegen nicht so weit entwickelt, da der Fokus dort eher auf den Layoutfunktionen liegt. Hellfritsch [2006, S. 30] bemängelt die notorisch schlechte Dokumentation und den Support durch Adobe. Außerdem kritisiert er, dass in den letzten zehn Jahren lediglich die Strukturierungsfunktionen verbessert wurden, die Oberfläche des Programms aber sehr spartanisch geblieben ist. Der Autor kann dies aus eigener Erfahrung bestätigen. Während Nummerierungs- und Strukturierungsfunktionen des Programms Alleinstellungsmerkmale unter den WYSIWYG-Programmen sind, lässt die Ergonomie bei der Bedienung zu wünschen übrig. So kann der Benutzer z. B. erst mit der Version 7.2 Arbeitsschritte mehrfach rückgängig machen und das Scrollrad der Maus auch ohne Zusatzprogramm einsetzen. Die Verlaufspalette (Mehrfach-Rückgängig-Funktion) ist dabei nicht optimal implementiert. Programmschritte wie Zoomen der Ansicht lassen sich rückgängig machen, wichtigere Funktionen wie das Importieren von Formaten allerdings nicht. Zudem wird der Benutzer in der Voreinstellung durch einen Warnhinweis bei all diesen wichtigen Arbeitsschritten darauf aufmerksam gemacht, dass diese nicht rückgängig gemacht werden können. Das von anderen Programmen bekannte Feld „diese Warnung nie wieder anzeigen“ mit einer Checkbox zum Aktivieren fehlt im FrameMaker-Hinweis. Die Deaktivierung des Hinweises gelingt erst umständlich über die allgemeinen Voreinstellungen des Programms.

1.4 Die Entscheidung für Adobe FrameMaker

Ursprüngliches Ziel dieser Arbeit war die prototypische Evaluierung eines Workflows für die Technische Dokumentation auf XML-Basis. Die Daten sollten mithilfe von XML besser strukturiert und zukunftssicherer gemacht werden. Dafür kamen verschiedene Ansätze in Frage. Die Entscheidung fiel letzten Endes auf eine XML-Anwendung mit Adobe FrameMaker. Im folgenden Abschnitt werden die Gründe für diese Entscheidung erläutert.

1.4.1 Integration bestehender Dokumente

Im konkreten Fall liegen bereits sehr umfangreiche Dokumentationen im FrameMaker-Format vor. Bei diesen Dokumentationen wurde das Programm unstrukturiert verwendet, visuell und logisch sind die Dokumente trotzdem sehr klar strukturiert. Es

lag daher nahe, von dieser Struktur ausgehend einen XML-Prozess zu entwickeln. Aus einigen Quellen wurde ersichtlich, dass FrameMaker als XML-Editor einsetzbar ist und dabei brauchbare Ergebnisse liefert [vgl. Bergerhoff 2006]. Beim XML-Export der Daten entsteht der Vorteil, dass die FrameMaker-eigenen Zeichencodes in Unicode umgewandelt werden. Dadurch wird der Austausch der Daten mit Übersetzungsdienstleistern vereinfacht. Außerdem sind die Daten bei einer Speicherung in einem XML-Format, wie bereits erläutert, zukunftssicher archiviert, da mit einer Unterstützung durch eventuell bessere oder verbesserte Programme in der Zukunft zu rechnen ist. Für ein strukturiertes Arbeiten mit komplexen XML-Daten nach dem WYSIWYG-Prinzip gibt es momentan schlicht keine brauchbare Alternative zu Adobe FrameMaker.

Beim Verwenden des Programms als XML-Editor ergeben sich im konkreten Fall dadurch wesentliche Vorteile: Zunächst muss kein neues Programm installiert werden und die Einarbeitungszeit in ein neues Programm entfällt. Die Implementierung lässt sich so parallel zum Tagesgeschäft durchführen. Der neue Workflow ist eine Optimierung des alten, in dem sich die Redakteure deshalb sehr schnell zurechtfinden. FrameMaker kann weiterhin für die bestehenden Dokumentationen und dort wenn nötig wie bisher unstrukturiert verwendet werden. Die Übernahme in die neue Struktur ist durch den Einsatz des gleichen Programmes leichter möglich, als wenn hier mehrere Programme parallel geöffnet werden und über die Zwischenablage gearbeitet werden muss. Ein großer Vorteil bei der Entwicklung der XML-Anwendung ist, dass die Formate der bestehenden Dokumente einen ersten Ansatzpunkt für die Entwicklung der neuen Dokumentstruktur bilden und dadurch, dass FrameMaker ein DTP-Programm mit nachträglich integrierter XML-Funktionalität ist, diese Formate leicht übernommen werden können. Ein auf reinen XML-Prozessen basierender Workflow würde die Entwicklung eines XSL-FO-Stylesheets für die PDF- bzw. Druckausgabe nötig machen und den Abschied vom WYSIWYG-Prinzip bedeuten. FrameMaker deckt diese Funktionen hingegen intern ab.

Der Einsatz eines CMS lohnt sich im konkreten Fall – einer Redaktion, die zusammen in einem Büro sitzt und aus zwei bis drei Redakteuren besteht – nicht. Eine XML-Anwendung mit Adobe FrameMaker bietet trotzdem die Möglichkeiten von XML. Die entwickelte Struktur kann zudem bei einer entsprechenden Weiterentwicklung in ein CMS integriert werden. Die bereits mit der XML-Anwendung generierten Daten ließen sich dabei leicht übernehmen.

1.4.2 Geschichte der Software

Adobe FrameMaker wurde ursprünglich für UNIX-Workstations von der Firma Frame Technology Corp. entwickelt und vertrieben. Zielkunden des Programms waren professionelle Autoren umfangreicher und stark technisch geprägter Publikationen. Nach einigem – auch wirtschaftlichen – Erfolg des Programms portierten es die Entwickler auf Microsoft Windows. Mit einem niedrigen Preis des Programms für diese Plattform (500 $ statt vorher 2500 $) wollte man Marktanteile gewinnen, zerstörte damit aber das bisher einheitliche Kundenprofil. Das Profi-Programm konnte im Consumer-Bereich mit seiner komplizierten Bedienung nicht punkten. Die Verkaufszahlen sanken und brachten die Firma in wirtschaftliche Bedrängnis. In dieser Situation kaufte Adobe Systems FrameMaker Mitte der Neunziger Jahre auf und konzentrierte den Vertrieb wieder auf die professionelle Kundschaft [Wikipedia 2007b].

Die Entwicklung durch Adobe verläuft seitdem schleppend. Die Hauptfunktionen sind seit 1995 unverändert geblieben, lediglich Fehlerkorrekturen und SGML/XML-orientierte Funktionen wurden implementiert. 2004 gab Adobe die Weiterentwicklung von FrameMaker für die Macintosh-Plattform auf und nährte damit Gerüchte über ein generelles Ende der Entwicklung zu Gunsten anderer Adobe-DTP-Programme wie z. B. InDesign.

Dem entgegengesetzt wird mittlerweile eine Version 8.0 angekündigt, die als wichtigstes Feature komplette Unicode-Unterstützung bieten soll [vgl. FrameMaker 8 – angebliche Features 2007]. Eine Weiterentwicklung wird auch vom Poduct Manager von FrameMaker in seinem Webblog mit den folgenden Worten bestätigt [Dokania 2007]:

> Let me assure you, as the Product Manager of FrameMaker, that FrameMaker is here to stay.

1.4.3 Einsatzbereit für komplexe Dokumentationen

FrameMaker ist seit den Anfängen der Software als Satzprogramm für Technische Dokumentationen entwickelt. Immer wieder wird bestätigt, dass das Programm für umfangreichste Dokumente eingesetzt werden kann, Hunderte von Seiten sind demnach kein Problem [vgl. Mitschke und Schulze 2003, S. 19], dies macht vor allem die zuverlässig funktionierende Buchfunktion möglich [vgl. Hellfritsch 2006, S. 33]. Aber auch andere Funktionen sind sehr hilfreich für komplexe Dokumentationen. Die komplexe Nummerierungsfunktion für Absätze ist z. B. ein Alleinstellungsmerkmal unter den DTP-Programmen [vgl. Mitschke und Schulze 2003, S. 142f]. FrameMaker ist dank seiner Anpassbarkeit für verschiedenste Einsatzgebiete geeignet. Adobe führt diese im FrameMaker Solutions Guide auf [vgl. FrameMaker Solutions Guide 2002].

Legt man die FrameMaker-Dateien auf einem Netzwerklaufwerk ab, können verschiedene Kapitel-Dateien von verschiedenen FrameMaker-Arbeitsplätzen aus gleichzeitig bearbeitet werden. Versucht ein Redakteur, eine bereits durch einen anderen Benutzer geöffnete Datei zu öffnen, gibt das Programm eine Fehlermeldung aus, nach der der Redakteur die Datei nur schreibgeschützt zum Lesen öffnen kann. Ein rudimentäres Mehrbenutzer-System lässt sich so leicht herstellen – ein schnelles Netzwerk und mehrere FrameMaker-Lizenzen vorausgesetzt. Über die Anbindung an eine Datenbank, die ebenfalls möglich ist, lässt sich dieses Mehrbenutzerkonzept noch erheblich erweitern.

Die vielen Funktionen und Erweiterungsmöglichkeiten sind für den durchschnittlichen Anwender jedoch schwer zugänglich. Zum Erlernen des eigenwilligen Bedienkonzepts benötigt man viel Zeit. Manche Erweiterungen lassen sich nur über die Programmierung von Plugins über die API, das sogenannte Frame Developer's Kit (FDK) erreichen. Eine einfachere Möglichkeit bietet das Verwenden von FrameScripts, die über den Einsatz eines umfangreichen Plugins der Firma ElmScript die meisten Funktionen des FDK auch ohne C-Programmierung zur Verfügung stellen.

Eine aussagefähige Referenz für den erfolgreichen Einsatz des Programms ist die umfangreiche Softwaredokumentation von Adobe für die Creative Suite, die nicht etwa mit InDesign, sondern mit FrameMaker bewerkstelligt wird.

1.4.4 FrameMaker als SGML- und XML-Editor

Neben dem proprietären FM-Speicherformat lassen sich mit FrameMaker Formate erzeugen, deren Spezifikation offen gelegt und die auch im Klartext lesbar sind. Dazu zählen nicht nur XML/SGML-Formate, sondern auch die älteren Formate MMF und MIF. Das Maker Markup Format (MMF) ist mit dem aus Textverarbeitungssoftware bekannten RTF-Format vergleichbar. Es handelt sich um ein relativ einfaches Format, mit dem sich die Textflüsse in FrameMaker automatisieren lassen. Dadurch, dass die Daten nicht sehr stark gekapselt sind, eignet es sich für den Einsatz im Database-Publishing. Das Maker Interchange Format (MIF) ist für den Austausch zwischen den verschiedenen FrameMaker-Plattformen (Microsoft Windows, Macintosh, UNIX) gedacht. Mit MIFs lassen sich vollständige FrameMaker-Dokumente beschreiben und dadurch von einer Plattform zur anderen transportieren.

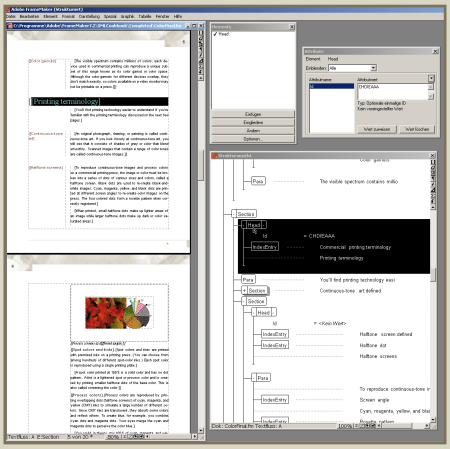
MMF und MIF sind beides reine Textformate, die im Zeichencode ASCII kodiert sind. Sämtliche über den ASCII-Zeichensatz hinausgehenden Sonderzeichen werden in einer FrameMaker-eigenen Syntax geschrieben. Für die Verwendung außerhalb von FrameMaker müssen diese Kodierungen daher konvertiert werden, damit die Sonderzeichen auch in anderen Anwendungen zugänglich sind. Die Struktur bilden in MMF- und MIF-Dokumenten lediglich hintereinander angeordnete, formatierte Absätze.

Mit der Version 5.12 FrameMaker und SGML hat Adobe im Jahr 1996 eine FrameMaker-Version auf den Markt gebracht, mit der verschachtelte Strukturen in Dokumenten selbst definiert und in einem SGML-Format gespeichert werden können. Seit der Version 7.0 kann die Struktur auch in XML-Syntax übersetzt werden und die beiden Programmversionen für unstrukturiertes und strukturiertes Arbeiten werden nicht mehr getrennt vertrieben, sondern sind in einer zusammengefasst. Zwischen „normalem“ und „strukturiertem“ FrameMaker kann einfach umgeschaltet werden.

Ab der Version FrameMaker 7.0 lässt sich also XML speichern. Nutzt man dazu keine sogenannte XML-Anwendung, die den internen FrameMaker-Strukturen die offenen XML-Strukturen zuweisen, dann wird eine XML-Datei in einer MMF-ähnlichen Struktur erzeugt. Mit der Entwicklung einer XML-Anwendung für die Technische Dokumentation beschäftigt sich das nächste Kapitel. Auch ohne Verwenden einer solchen strukturierten Anwendung wird beim Speichern von XML der interne Zeichensatz des FrameMaker automatisch in den Unicode-Standard übersetzt, so dass diese Daten auch in anderen Anwendungen besser nutzbar sind.

1.4.5 WYSIWYG, Strukturansicht und PDF-Engine unter einer Oberfläche

Ist die Oberfläche des Programms auf „strukturierten FrameMaker“ eingestellt, stehen zusätzliche Dialoge im Programm zur Verfügung. Dabei handelt es sich primär um die Strukturansicht und den Elementkatalog. In einem strukturierten Dokument kann man so im Elementbaum navigieren, Elemente markieren, kopieren und einfügen (auch per „Drag and Drop“). Aus dem Elementkatalog kann man die für dieses Dokument definierten Elemente einfügen. Normalerweise werden dort nur die Elemente angezeigt, die in der Struktur an der Stelle, wo man sich gerade mit dem Cursor im Dokument befindet, gültig sind. Dieses Vorgehen entspricht der Arbeitsweise mit gängigen, reinen XML-Editoren.



In strukturierten FrameMaker-Dokumenten wird mithilfe der Strukturansicht innerhalb der Elementstruktur navigiert. Mit dem Elementkatalog können die an der Cursor-Position gültigen Elemente eingefügt werden. Über Attributemanager werden die Attribute von Elementen verwaltet. Die bekannte WYSIWYG-Ansicht im Dokumentfenster steht dem Benutzer dabei weiterhin, wie von unstrukturierten Dokumenten gewohnt, zur Verfügung.

Im Screenshot ist das Beispieldokument aus dem „XML-Cookbook“ zu sehen, einem Tutorial, das der Software für einen leichteren Einstieg in das Thema „FrameMaker und XML“ im gleichnamigen Unterordner des Programmverzeichnisses beiliegt

Abbildung 1.10: FrameMaker (Strukturiert)

Zusätzlich zu dieser strukturierten Arbeitsweise steht immer noch die WYSIWYG-Seitenansicht zur Verfügung, in der Text eingegeben und das Layout sofort im Programm kontrolliert wird. FrameMaker kann direkt die Druckausgabe mittels PDF erzeugen, wo beim Einsatz eines reinen XML-Editors ein XSL-FO-Prozess nötig wäre. Dadurch, dass es sich um ein Adobe-Produkt handelt (Adobe hat PostScript entwickelt), ist die Druckausgabe sehr ausgereift. PDFs können Verweis-sensitiv erzeugt werden, so dass Querverweisen im Dokument leicht gefolgt werden kann. Zusätzlich werden PDF-Lesezeichen unterstützt.

Adobe FrameMaker bietet so den Funktionsumfang eines DTP-Programms mit spezifisch auf die Technische Dokumentation zugeschnittenen Möglichkeiten und integriert außerdem die Funktionen eines XML-Editors. Da es kein Programm mit vergleichbarem Funktionsumfang im WYSIWYG-Bereich gibt, lohnt es sich, in der Technischen Dokumentation auf FrameMaker zu bauen, auch wenn die Umstellung auf einen XML-Workflow mit dem Programm eine zeitaufwendige und komplexe Entwicklung voraussetzt.

2 Erstellen einer XML-Anwendung mit Adobe FrameMaker für die Technische Dokumentation

Die Entwicklung einer XML-Anwendung zur Verwendung mit Adobe FrameMaker ist ein langwieriger und komplexer Vorgang. Es gibt unterschiedlichste Einsatzmöglichkeiten von XML-Anwendungen und daher auch unterschiedlichste angepasste Varianten. In der vorliegenden Arbeit kann daher nur die generelle Vorgehensweise bezogen auf die Eigen-Entwicklung dargestellt werden. Zusätzlich werden einige ausgewählte Ergebnisse vorgestellt.

Im folgenden Abschnitt 2.1 „Grundlagen von XML-Anwendungen mit Adobe FrameMaker“ wird ein Überblick über die Bestandteile einer XML-Anwendung und deren Zusammenwirken gegeben. Danach wird in Abschnitt 2.2 „Ausgangssituation und Zielsetzung“ erörtert, wie die Anwendung konzipiert ist. In Abschnitt 2.3 „Erstellen der Anwendung“ wird schließlich gezeigt, wie das Konzept Schritt für Schritt umgesetzt wurde.

2.1 Grundlagen von XML-Anwendungen mit Adobe FrameMaker

XML-Anwendungen mit Adobe FrameMaker sind modular aufgebaut. Die verschiedenen Bestandteile einer XML-Anwendung werden im folgenden Abschnitt 2.1.1 kurz vorgestellt. Später werden sie in einzelnen Abschnitten genauer beschrieben.

2.1.1 Bestandteile der Anwendung

Eine XML-Anwendung besteht aus den folgenden Bestandteilen:

- Anwendungsdefinitionen
  - bestimmen Namen der Anwendung und zugehörige Dokumententypen
  - teilen FrameMaker mit, wo die übrigen Bestandteile einer Anwendung gespeichert sind
  - werden zentral in einer Datei im Installationsverzeichnis von FrameMaker verwaltet
- strukturiertes Template
  - eine Vorlagedatei, die sämtliche benötigte Formate und zusätzlich einen Elementkatalog enthält
  - Elementkatalog wird über ein EDD (Element Definition Document) erzeugt und in das Template importiert

- Lese-/Schreibregeln
  - bestimmen, wie strukturierte FrameMaker-Dokumente beim Importieren und Exportieren von XML übersetzt werden
- DTD
  - enthält die Strukturinformationen für die zu importierenden bzw. exportierten XML-Daten
  - vor dem Import bzw. nach dem Export werden XML-Daten gegen die DTD validiert
- XSLT-Stylesheet (nur mit FrameMaker-Version 7.2)
  - möglicherweise ganz am Anfang eines XML-Imports noch vor den Lese-/Schreibregeln
  - und/oder ganz am Ende eines XML-Exports nach den Lese-/Schreibregeln angewendet
  - vor und nach dem Import/Export müssen XML-Daten gegen die DTD valide sein
  - der implementierte Prozessor ist Xalan
- Structured API Client
  - ein selbst in C programmiertes und kompiliertes Plugin für FrameMaker, dass beim Import und Export der XML-Daten zusätzliche Funktionen übernehmen kann, die sich nicht mithilfe von Lese-/Schreibregeln oder XSLT-Stylesheet realisieren lassen
  - wird kein API-Client angegeben, wird der minimale Standard-Client verwendet, der die Daten ohne Prüfung und Bearbeitung weitergibt

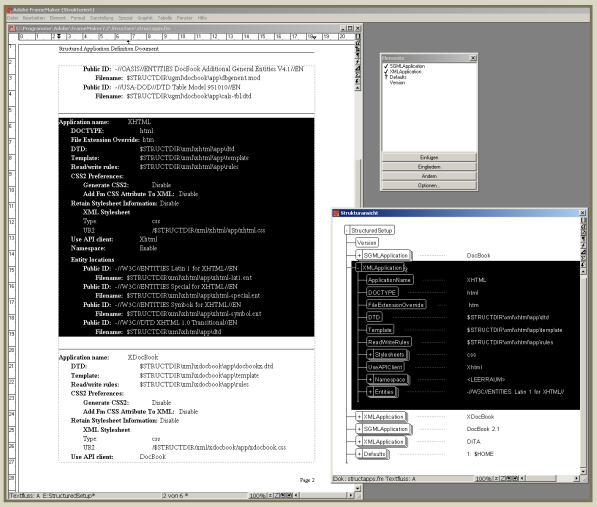
Eine minimale XML-Anwendung für Adobe FrameMaker enthält ein strukturiertes Template, Lese-/Schreibregeln und eine DTD, die in den Anwendungsdefinitionen neben dem Namen der Anwendung und den zugehörigen Dokumenttypen festgelegt werden. Wenn keine Lese-/Schreibregeln für eine Anwendung existieren, werden Voreinstellungen zum Übersetzen der Daten von und nach XML verwendet, deshalb müssen laut Dokumentation Lese-/Schreibregeln nicht angegeben werden. Die Ergebnisse des Autors zeigen aber, dass die Übersetzung nur mit einem Minimum an Lese-/Schreibregeln richtig funktioniert, da das dabei entstehende XML sonst nicht wohlgeformt ist (leere Querverweiselemente werden nicht geschlossen).

Ein XSLT-Stylesheet wird in der erstellten XML-Anwendung verwendet und daher auch in dieser Arbeit genauer erläutert. XSLT funktioniert aber nur mit der neuesten FrameMaker-Version 7.2. Die Möglichkeiten eines Structured API Clients können den Funktionsumfang einer XML-Anwendung um ein Vielfaches erweitern, setzen aber Programmierkenntnisse in C und ein gutes Verständnis der FrameMaker-API voraus. Da ein Structured API Client in der entwickelten Anwendung nicht zum Einsatz kommt und eine Beschäftigung mit diesem Anwendungsbestandteil den Umfang dieser Arbeit überschreiten würde, wird an dieser Stelle nicht weiter darauf eingegangen.

Nähere Informationen zu den Bestandteilen von XML-/SGML-Anwendungen sind in der Softwaredokumentation zu finden [vgl. Struct. App. Dev. Guide 2005, S. 33–40]. Der Structure Application Developer's Guide ist das Handbuch für angehende Anwendungs-Entwickler. Praktisch ist die darin enthaltene Referenz für die Lese-/Schreibregeln. Einen leichteren Einstieg in das Thema ermöglichen Informationen im Internet, z. B. die Whitepapers von „Publishing Smarter“ [vgl. Publishing Smarter 2006].

2.1.2 Die Anwendungsdefinitionen

Die Anwendungsdefinitionen werden in der Datei mit dem Namen `structapps.fm` im Unterverzeichnis `Structure` des Programmverzeichnisses von Adobe FrameMaker gespeichert. Die Datei kann am einfachsten aus dem Programm über den Menüpunkt „Datei → Strukturierungswerkzeuge → Anwendungsdefinitionen bearbeiten“ geöffnet werden. `structapps.fm` ist selbst eine strukturierte Datei, die definiert, welche Anwendungen beim Import und Export von XML/SGML zur Verfügung stehen.



Die Anwendungsdefinitionen sind in der Datei *structapps.fm* im Unterverzeichnis *Structure* des FrameMaker-Programmverzeichnisses gespeichert. Es handelt sich dabei um eine strukturierte Datei.

Die Software wird mit einigen vorkonfigurierten Standard-Anwendungen für XML und SGML ausgeliefert. Im Screenshot ist die Struktur für die XHTML-Anwendung von FrameMaker abgebildet.

In der Praxis reichen die mitgelieferten Anwendungen nicht aus. Sie müssen durch eigene Entwicklungen oder die Entwicklungen externer Dienstleister ergänzt werden.

Abbildung 2.1: Die Anwendungsdefinitionen

Mithilfe der Strukturansicht und des Elementkatalogs kann der Benutzer eigene Anwendungen definieren oder bestehende Anwendungsdefinitionen ändern.

Jede in dieser Datei definierte Anwendung beschreibt eine Art von FrameMaker-eigenen strukturierten Dokumenten, mit denen der Anwender arbeitet, sowie ein dazugehörendes Markup-Format (XML- oder SGML-basiert) und wie beim Einlesen bzw. Ausgeben zwischen Markup- und FrameMaker-Struktur übersetzt wird.

In einer Anwendungsdefinition wird dem Programm mitgeteilt, aus welchen Bestandteilen eine Anwendung besteht und wo die entsprechenden Dateien auf der Festplatte des Benutzers gespeichert sind. Zusätzlich ordnet man einer Anwendung Doctypes zu. Doctypes betreffen in einer XML-Anwendung mit FrameMaker die „Wurzelelemente“ in den FrameMaker-Dokumenten, aus denen die XML-Daten erzeugt wird. Beim Speichern eines entsprechenden FrameMaker-Dokumentes in XML wird automatisch die

dazu gehörende XML-Anwendung verwendet. Gibt es den Doctype mehrfach in verschiedenen Anwendungen, muss der Benutzer beim Speichern in XML die richtige Anwendung auswählen.

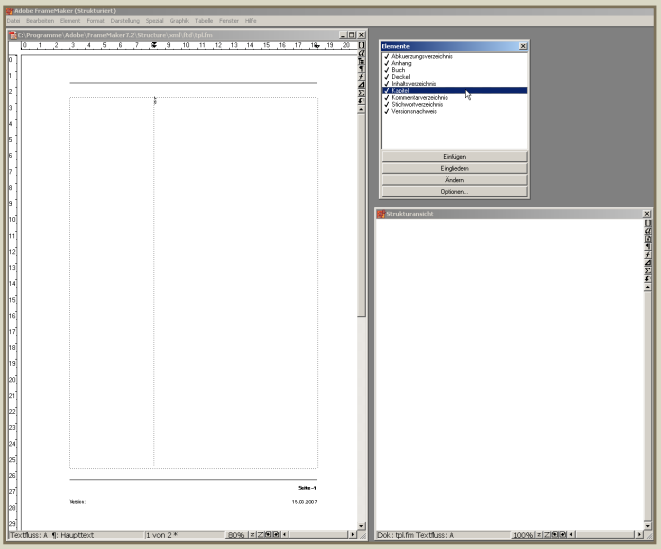
FrameMaker bringt einige strukturierte Anwendungen von Haus aus mit. Bei Version 7.2 handelt es sich um DocBook und DocBook 2.1 für SGML sowie XHTML, XDocBook und DITA für XML. Die vordefinierten Anwendungen funktionieren schlecht und sind kaum dokumentiert. Adobe hat nachgebessert und bietet zusätzliche sogenannte AppPacks im Internet kostenlos zum Download an [vgl. DITA und S1000D Application Packs 2006]. Es gibt auch Alternativen, z. B. das kostenlose „FrameDita lite“ der Firma „Publishing Smarter“, zu denen es umfangreiche Online-Handbücher gibt [vgl. Publishing Smarter 2006]. Solche vorgefertigten Anwendungen stellen aber meist nur den Ausgangspunkt für nötige eigene Anpassungen und Entwicklungen dar.

2.1.3 Das Template

Ein Template ist eine Dokumentvorlage für FrameMaker-Dateien. Darin sind vorgefertigte Text- und Seitenlayouts enthalten. Dazu gehören z. B. die Formate der Vorgabeseiten, darin enthaltene Variablen für Kopf und Fußzeilen, Querverweisformate und benannte Rahmen auf sogenannten Referenzseiten, die im Dokument an Absatzstile „gehängt“ oder diesen vorangestellt werden können. Das Einrichten und Verwalten von Templates für FrameMaker-Dateien ist schon bei der unstrukturierten Verwendung des Programms ein komplexes Unterfangen, das von erfahrenen Benutzern durchgeführt werden sollte. Mehr dazu findet sich z. B. bei Mitschke und Schulze [2003, S. 382–391].

Ein strukturiertes Template, wie es in XML-Anwendungen verwendet wird, enthält zusätzlich Strukturinformationen, die über die Elementdefinitionen (EDD) importiert werden. Vorhandene unstrukturierte Templates können relativ einfach als Ausgangsdatei für die Entwicklung eines strukturierten Templates verwendet werden. Den Elementen in der EDD weist man zu diesem Zweck einfach die bestehenden Absatzformate zu. Durch Veränderungen an den Absatzformaten lässt sich das Template dadurch später auch ohne eine Veränderung der Elementdefinitionen grundlegend anpassen.

Absatzformate können aber auch in den importierten Elementdefinitionen enthalten sein. In den Elementdefinitionen wird dann die Formatierung der Elemente neben deren Anordnung und Hierarchie im Dokument gleich mit definiert. Dadurch ergeben sich, gerade für komplexere Dokumente, viele Vorteile. Formate, die innerhalb der EDD definiert sind, werden automatisch vererbt. Ein Kindelement übernimmt so die Absatzformatierung seines Elternelements, falls für das Kindelement keine eigenen Formatabweichungen definiert werden. Gerade bei einer Vielzahl von Formaten ist das von Vorteil, da so z. B. die Änderung der Grundschriftart eines Dokuments nur in einem übergeordneten Element erfolgt und nicht alle Formate eines umfangreichen Absatzkatalogs angepasst werden müssen. In der EDD gibt es außerdem die Möglichkeit bedingter Formatierungen, d. h. Elemente erscheinen im Dokument je nach ihrem Kontext (Elternelement, vorgeordnete und nachgeordnete Elemente) anders formatiert. Damit sieht beispielsweise ein Element „Ueberschrift“ in einem Elternelement „Kapitel“ anders aus, als in einem Elternelement „Abschnitt“. Das ließe sich



Ein strukturiertes Template ist ein leeres strukturiertes FrameMaker-Dokument.

Es dient als Dokumentvorlage.

Durch die Verknüpfung eines bestimmten strukturierten Templates mit einer bestimmten Struktur-Anwendung wird die Konsistenz der Elementstruktur und ein einheitliches Printlayout der Dokumente innerhalb einer Anwendung sichergestellt.

Auch ohne XML-Anwendung lassen sich mit strukturierten Templates bereits strukturierte FrameMaker-Dokumente komfortabel anfertigen.

Abbildung 2.2: Das strukturierte Template

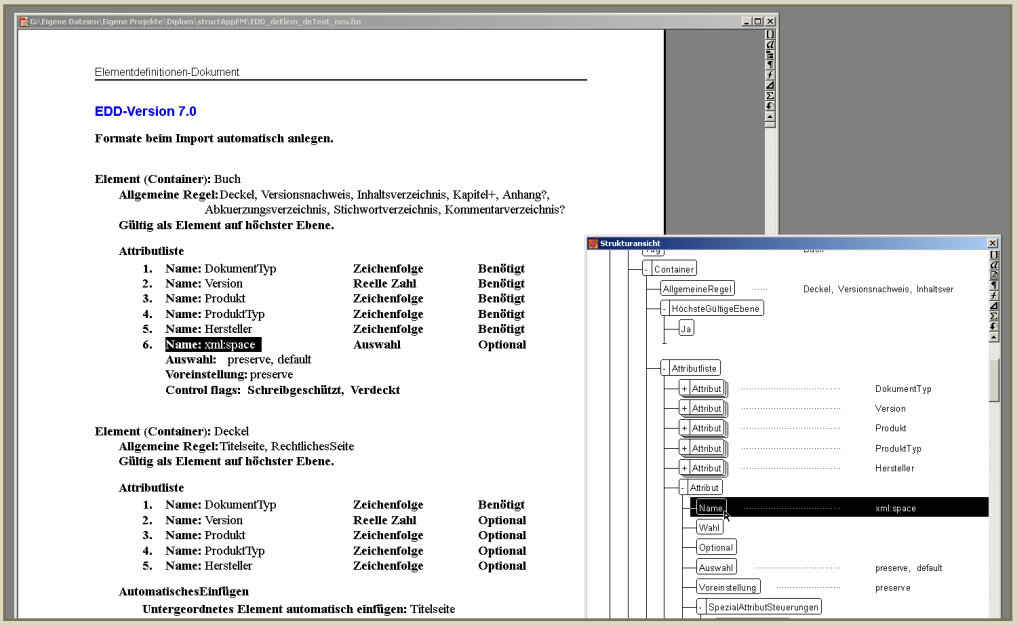
zwar grundsätzlich über die Kontext-bedingte Zuweisung von Absatzformaten erreichen, das Problem ist aber, dass bei einer großen Anzahl der Elemente und Kontexte der Absatzkatalog im Template sehr umfangreich und damit unübersichtlich wird.

Es gibt Gründe, warum das Element „Ueberschrift“ mit „ue“ statt mit „ü“ geschrieben wird, siehe Abschnitt 2.1.5.

2.1.4 Die Elementdefinitionen (EDD)

Das „Element Definition Document“ bzw. EDD enthält die zur Strukturierung von FrameMaker-Dokumenten notwendigen Informationen. Darin ist die Anordnung und Hierarchie von Elementen sowie deren Attribute enthalten, vergleichbar mit der Strukturinformation in einer DTD für XML/SGML-Daten. Zusätzlich sind aber auch Informationen bezüglich der Formatierung der Elemente enthalten, was einer über XSL-FO realisierten Funktionalität in einem reinen XML-Prozess entspricht. Nicht alle Formate eines Dokuments lassen sich mit dem EDD zuweisen, sondern üblicherweise nur die Absatzformate, sowie Startformate für spezielle Elemente wie Tabellen oder Querverweise. Die Formate für Tabellen, Querverweise, etc. und andere, z. B. auf Referenzseiten angelegte, benannte Grafikrahmen oder in Vorgabeseiten definierte Seitenlayouts, können im EDD nicht angegeben werden. Diese befinden sich in einem Template, in das die Elementdefinitionen aus dem dazu passenden EDD importiert werden. Erst dieses strukturierte Template ist wirklicher Bestandteil der in der `structapp.fm`-Datei definierten Anwendung.

Ein EDD ist selbst eine strukturierte Datei mit einer von FrameMaker vorgegebenen Datei. Der Suffix ist darum auch „.fm“. Um ein neues EDD anzulegen wird im Menü der Befehl „Datei → Stukturierungswerkzeuge → Neue EDD“ gewählt. Ist ein Elementkatalog aus einem EDD in einer strukturierten Datei enthalten, kann der Inhalt dieser Datei auf Gültigkeit nach diesem EDD geprüft werden. Dazu wählt der Anwender im



Im Elementdefinitionen-Dokument (EDD) wird die Struktur des Dokumentes mit allen Elementen und Attributen definiert. Zusätzlich werden die Formatierungen der Elemente für das FrameMaker-Printlayout angegeben.

Im Screenshot ist die Definition des *xml:space*-Attributs zu sehen. Das Attribut wird in der EDD als verstecktes Attribut mit der Voreinstellung *preserve* definiert. Für dieses Attribut wird keine Lese-/Schreibregel angegeben, damit es genauso in die XML-Datei übernommen wird. Beim XML-Export wird dadurch beim Einsatz eines XSLT-Stylesheets der Leerraum am Anfang und am Ende eines Elementes nicht entfernt (Es gibt in FrameMaker leider noch kein anderes Konzept für Namespaces).

Diese Regel ist sinnvoll, da FrameMaker intern eine Leerzeichenkontrolle mitbringt und dadurch meißt nur Leerraum entfernt wird, den man wirklich haben will (z. B. mit Shift+Enter eingefügte Zeilenumbrüchen in Tabellenzellen zur Ausrichtung des Textes in Spalten).

Abbildung 2.3: Die Elementdefinitionen (EDD)

Menü den Befehl „Element → Prüfen“. Von der als erlaubt definierten Struktur abweichende Elemente oder unvollständige festgelegte Abfolgen von Elementen innerhalb anderer Elemente können so leicht gefunden werden.

2.1.5 Die Lese-/Schreibregeln

Die Lese-/Schreibregeln übersetzen zwischen XML (oder SGML) und FrameMaker-eigener Struktur beim Import und Export. Dadurch kann beispielsweise erreicht werden, dass sich Namen von Elementen im Markup und innerhalb der FrameMaker-Dokumente unterscheiden. Nicht jede Funktionalität von FrameMaker will man im Markup genauso abgebildet haben wie innerhalb der FrameMaker-Struktur. Oft gibt es verschiedene mögliche Abbildungen der Strukturen, sowohl innerhalb des Programms, als auch im Markup. Deshalb wird der Übersetzungsvorgang beim Import bzw. Export von XML (oder SGML) durch Lese-/Schreibregeln konfiguriert. Dort werden die Entscheidungen für die unterschiedlichen möglichen Varianten getroffen.

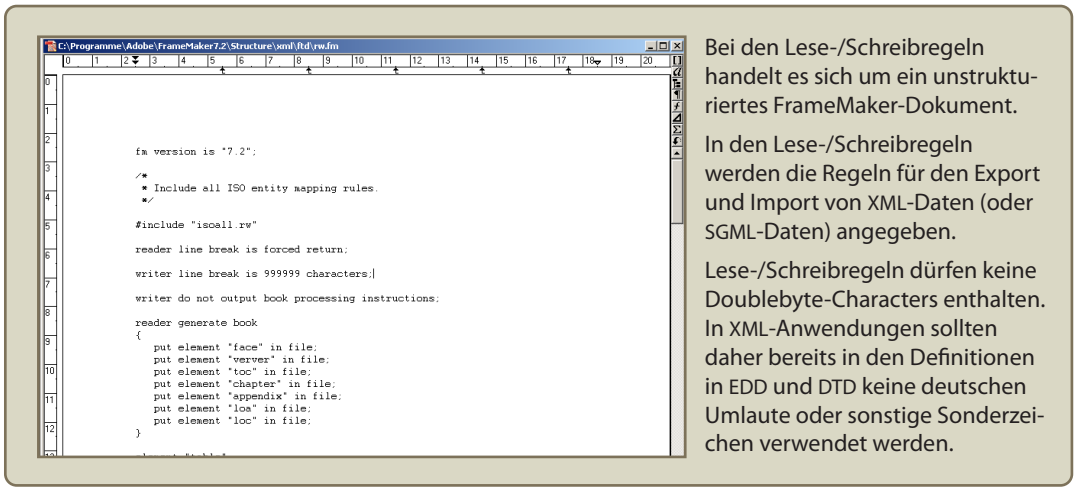


Abbildung 2.4: Die Lese-/Schreibregeln

Für eine funktionierende XML-Anwendung muss ein Basissatz von Lese-/Schreibregeln angegeben werden. Das gilt auch für einige nicht funktionierende Voreinstellungen bei der Übersetzung. Beim Export von XML ganz ohne Lese-/Schreibregeln werden z. B. alle Querverweiselemente, die als leere Elemente im Markup stehen, nicht geschlossen. Dadurch ist das erzeugte XML nicht wohlgeformt und der Exportvorgang bricht mit einer Fehlermeldung ab. Mit der Angabe

```
element "xml-crossrefelement" is fm crossref element;
```

in den Schreibregeln funktioniert der Vorgang hingegen problemlos.

Die Lese-/Schreibregeln verwenden eine spezielle Syntax, die im Struct. App. Dev. Guide [2005] genau beschrieben wird. Im Online-Handbuch findet sich die Referenz für die Skriptsprache. Gespeichert werden die Lese-/Schreibregeln wiederum mit dem Suffix „.fm“. Neue Lese-/Schreibregeln kann der Anwender mit dem entsprechenden Befehl unter „Datei → Strukturierungswerkzeuge“ anlegen.

Es gibt eine für eine deutschsprachige XML-Anwendung ungünstige Beschränkung der Lese-/Schreibregeln: Sie können keine Doublebyte-Characters enthalten, d. h. keine über den ASCII-Zeichensatz hinausgehenden Zeichen. Davon sind z. B. auch die deutschen Umlaute betroffen. Dadurch kommt es zu einer paradoxen Situation: In einem EDD lässt sich beispielsweise ein Element „Überschrift“ definieren, das innerhalb der FrameMaker-Struktur problemlos funktioniert. Das Übersetzen dieses Elements in das XML-Element „heading“ scheitert jedoch an den Beschränkungen der Lese-/Schreibregeln. Schon im EDD sollten deshalb keine Elemente oder Attribute mit Umlauten im Namen definiert werden.

Einige der Funktionalitäten von Lese-/Schreibregeln können ab FrameMaker-Version 7.2 auch durch den Einsatz eines XSLT-Stylesheets erreicht werden. Dadurch lassen sich die Probleme mit Doublebyte-Characters bzw. Umlauten umgehen. Dies gilt jedoch nicht für Querverweise, Tabellen und Grafiken, deren Modelle „tiefer“ im Programm implementiert sind. Einige ihrer Eigenschaften sind nicht über Attribute der Struktur – teilweise auch nicht über die Dialoge im Programm – steuerbar. Diese Eigenschaften sind die sogenannten FM-Properties, für die zwingend Lese-/Schreibregeln angegeben werden müssen.

Es gibt weitere Beschränkungen der Lese-/Schreibregeln. Ein Beispiel dafür sind mehrere Tabellenelemente innerhalb des EDD, die dort funktionieren, sich jedoch wiederum nicht in Markup übersetzen lassen, da mit Lese-/Schreibregeln offensichtlich nur ein einziges definiertes Tabellenmodell möglich ist. Diese Beschränkung ist, wie viele andere, nicht im Struct. App. Dev. Guide [2005] dokumentiert. Eventuell lässt sie sich aber durch den Einsatz eines Structured API Clients aufheben. Darauf verweist die Dokumentation jedenfalls bei den wenigen dokumentierten Beschränkungen. Diese Möglichkeit wird innerhalb dieser Arbeit nicht weiter untersucht, da bei der Entwicklung der Anwendung dafür keine Ressourcen zur Verfügung standen.

2.1.6 Die DTD

Die Document Type Definition (DTD) gehört nicht zur FrameMaker-eigenen Struktur, sondern zum Markup. DTDs werden in XML-Sprachen zur Beschreibung der festgelegten Hierarchie der Elemente und deren Attribute verwendet. Informationen zur Formatierung der Elemente, ähnlich wie im EDD, sind nicht enthalten. Die Ursprünge von DTDs liegen in SGML. Für XML gibt es zur Definition der Struktur weiterhin die Möglichkeit, XML-Schema (XSD) einzusetzen. Für XML-Anwendungen mit Adobe FrameMaker können auch XML-Schema Dateien an Stelle von DTDs eingesetzt werden. Da FrameMaker diese Dateien aber nicht direkt verarbeitet, sondern intern wieder in DTDs umwandelt, ergibt sich daraus kein Vorteil. In dieser Arbeit wird der Einsatz von XSD daher nicht weiter behandelt.

Gegen die zu einer XML-Anwendung gehörende DTD werden die erzeugten XML-Dateien validierend geparst. Bei Fehlern wird der Prozess mit einer Fehlermeldung abgebrochen. Die Zeilennummer, bei der der Fehler aufgetreten ist, wird in der Meldung mit angegeben, was eine Fehlersuche erleichtert. Gegen die DTD wird immer ganz am Anfang oder ganz am Ende eines XML-Imports oder Exports validiert, kommt ein XSLT-Stylesheet zum Einsatz, muss das XML zusätzlich vor und nach der Anwendung des Stylesheets gültig sein.

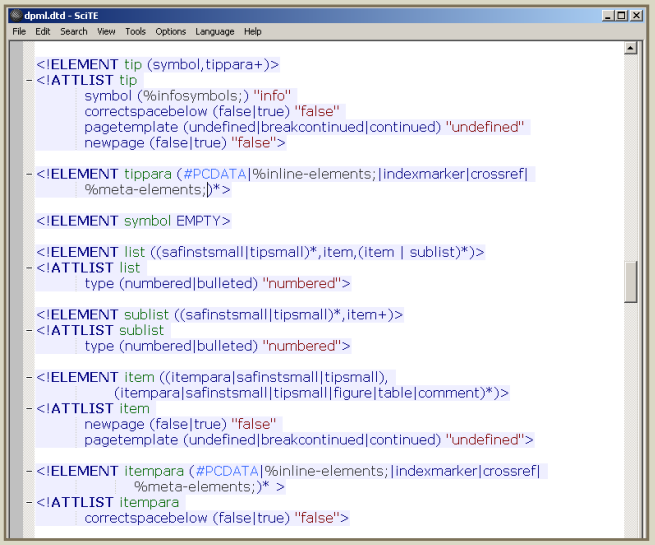


Abbildung 2.5: Die DTD

Wenn man eine Anwendung für eine XML-Standardsprache erstellt, kann deren DTD als Ausgangspunkt der Entwicklung dienen. Die DTD kann in diesem Fall in das Grundgerüst eines EDD konvertiert werden.

Bei einem selbst entwickelten EDD, die in eine DTD übersetzt werden soll kann ebenfalls ein Automatismus genutzt werden. Bei der Konvertierung eines EDD in eine DTD werden auch die Lese-/Schreibregeln berücksichtigt, so dass deren Funktionsweise dadurch leicht überprüfbar ist. Es gibt allerdings ein Problem bei den automatisch erzeugten DTDs, wenn diese als Elementinhalt Mixed Content, d. h. gleichzeitig Text (PCDATA) und/oder andere Elemente enthalten. In XML können diese nur in einer einzigen gültigen Form in der DTD angegeben werden:

```
(#PCDATA|element1|element2| ... |elementN)*
```

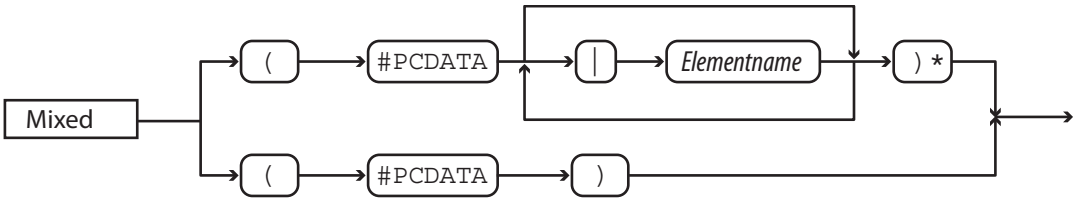


Abbildung 2.6: Mixed Content [Quelle: Behme und Mintert 2000, S. 79]

In EDDs gibt es diese Beschränkung nicht. Beim automatischen Übersetzen der EDD in eine DTD wird das Elementmodell aus dem EDD komplett übernommen. Bei Mixed Content muss daher noch manuell nachgearbeitet werden.

2.1.7 XSLT (nur FrameMaker 7.2)

Das Einbinden von XSLT-Stylesheets in XML-Anwendungen mit Adobe FrameMaker ist nur mit der neuesten Version 7.2 möglich. XSLT ist ein vom W3C entwickelter

Standard zur Konvertierung von XML-Daten. Der Einsatz von XSLT erfordert einen Prozessor. In FrameMaker 7.2 wurde Xalan implementiert.

Die Verarbeitung der XML-Daten mit dem XSLT-Stylesheet erfolgt ganz am Anfang des XML-Import- und/oder ganz am Ende des XML-Exportvorgangs. Dies wird in den Anwendungsdefinitionen eingestellt.

Das Stylesheet ist eine Textdatei, die mit einem beliebigen Editor erzeugt und bearbeitet werden kann. Mit XSLT kann man den Inhalt eines XML-Dokuments in andere Formate umwandeln. Als mögliche Ausgabeformate stehen dabei XML, HTML und Text zur Verfügung. Mit der Programmiersprache eröffnen sich noch viele weitere Möglichkeiten, z. B. können nur bestimmte Informationen aus einem Dokument gefiltert werden. Bei der erstellten Anwendung wird XSLT benutzt, um strukturierte Inhaltsverzeichnisse und andere Listen zu erzeugen.

Einen weiterführenden Einstieg in XSLT bieten Tidwell und Lichtenberg [2003].

2.2 Ausgangssituation und Zielsetzung

Im folgenden Abschnitt wird erläutert, wie bei der Entwicklung der XML-Anwendung die Ausgangssituation analysiert wurde und welche Zielsetzungen sich daraus ableiten ließen. Diese Vorarbeit zur Entwicklung ist wichtig, um die gefundene Lösung genau an die konkreten Bedürfnisse in der Technischen Redaktion anpassen zu können.

2.2.1 Unterschiedliche Entwicklungsszenarien

Am wichtigsten bei der Entwicklung einer XML-Anwendung mit Adobe FrameMaker sind Testläufe mit realen Daten. Dafür sollten aktuelle Dokumente mit vielen Seiten und allen möglichen Varianten von Elementanordnungen und Formatierungen verwendet werden. Es finden sich immer wieder Fehler in einer Anwendung. Die meisten, aber längst nicht alle dieser Fehler werden vom Benutzer verursacht (es reicht eine falsch gesetzte Klammer im EDD). Manche unlogische Fehler sind auf Beschränkungen bzw. Programmfehler von FrameMaker zurückzuführen. Dazu gehört z. B., dass in Lese-/Schreibregeln nur ein Tabellenelement verarbeitet werden kann.

Wenn ein Bestandteil der Anwendung funktioniert, heißt das noch nicht, dass die gesamte Anwendung funktioniert. Deshalb muss die Anwendung insgesamt getestet werden, d. h. XML muss auch exportiert und wieder importiert werden. Ist das Ergebnis in der WYSIWYG-Ansicht des FrameMakers bzw. im erstellten PDF vor und nach dem Export und Reimport der Daten gleich, dann funktioniert die Anwendung – oder besser gesagt zumindest der von diesem Testlauf betroffene Teil der Anwendung.

Wie eine XML-Anwendung funktioniert, hängt immer vom Einsatzgebiet der Anwendung ab und bedeutet einen unterschiedlichen Entwicklungsaufwand. Wenn XML-Daten z. B. nicht exportiert werden, sondern nur zum Formatieren und Drucken in FrameMaker geöffnet werden sollen, ist dieser Aufwand relativ gering. Bei einer komplexen Anwendung wie der vom Autor erstellten ist der Aufwand ungleich höher, da das Programm hier ebenfalls als Editor dient. Da zusätzlich XSLT verwendet wird, müssen die XML-Dateien nach dem Speichern wieder importiert werden, da XSLT nur mit XML und nicht innerhalb von FrameMaker funktioniert.

Allgemein kann man drei Hauptentwicklungsszenarien unterscheiden:

- Es gibt bereits einen XML-Workflow:
  - Gegebene DTD Ausgangspunkt für Entwicklung der EDD und der Lese-/Schreibregeln
  - Evtl. Integration bestehender XSLT-Stylesheets
- Es soll in einer XML-basierten Standardsprache publiziert werden (z. B. DocBook, DITA, Spec1000D):
  - DTD des entsprechenden Standards Ausgangspunkt weiterer Entwicklungen
  - Evtl. kann schon auf vorkonfigurierte Anwendungen zurückgegriffen bzw. diese angepasst werden (z. B. hat FrameMaker 7.2 vorgefertigte Anwendungen für DITA und DocBook)
- Neuentwicklung oder Strukturieren eines bestehenden FrameMaker-Dokumentationsworkflows:
  - Zuerst EDD entwickeln (evtl. bestehende unstrukturierte FrameMaker-Dokumente dafür analysieren)
  - Dann DTD entwickeln (Ausgangspunkt: automatische Funktion zur EDD-Umwandlung in DTD)
  - Gleichzeitig zur EDD: Lese-/Schreibregeln entwickeln
  - Danach evtl. zusätzliche Funktionen durch XSLT entwickeln

Der Struct. App. Dev. Guide [2005, S. 49–54] beschäftigt sich in einem Kapitel ausführlich mit verschiedenen Entwicklungsszenarien und den damit verbundenen Entwicklungsschritten.

2.2.2 Integration bestehender Dokumente

Der Ausgangspunkt bei der Eigen-Entwicklung ist ein funktionierender Redaktionsprozess, bei dem sich Adobe FrameMaker bereits im Einsatz befindet. Das Programm wird zwar noch unstrukturiert verwendet, die Dokumente werden jedoch bereits klar und einheitlich nach logischen und visuellen Gesichtspunkten in Anlehnung an die Methoden Funktionsdesign und Information Mapping® strukturiert. Durch den bereits bestehenden Workflow mit FrameMaker sind umfangreiche Dokumente mit Absatzkatalogen und Formaten entstanden, die stetig nach Aspekten der Qualitätssicherung weiterentwickelt und verbessert wurden. Die bestehenden Formatierungen der Dokumente können daher leicht als Ausgangspunkt der Strukturierung in der XML-Anwendung verwendet werden.

Das umfangreichste und aktuellste Dokument wird während der Entwicklung in die neue Struktur überführt, dabei bleiben die Texte, Tabellen und Grafiken über Kopieren und Einfügen bestehen. Die Anwendung wird so bei der Entwicklung gleich mit realen Daten getestet.

2.2.3 Konzipieren der neuen Dokument-Struktur

Zum Konzipieren der neuen Dokument-Struktur wird zunächst die bestehende Dokumentation anhand des aktuellsten, am weitesten entwickelten Dokuments analysiert. Der bestehende Workflow kann so in einer groben Skizze der zukünftigen Struktur abgebildet und die nötigen Formate daraus abgeleitet werden.

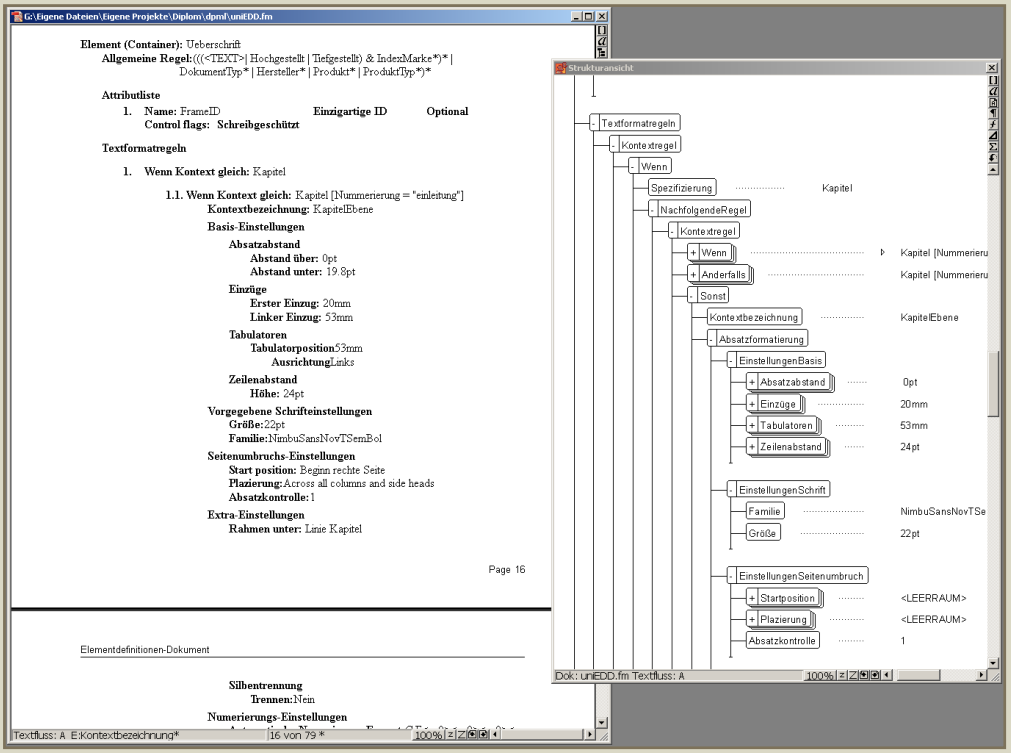
Eine wichtige Frage ist, inwiefern nach Standards, wie z. B. DocBook gearbeitet werden kann. Die Ergebnisse der Recherchen des Autors zeigen jedoch, dass die bestehenden Formate sich in einer Standard-DTD nur schlecht abbilden ließen, daher wird eine Eigen-Entwicklung in Angriff genommen. Bestehende Standards werden trotzdem berücksichtigt. Am wichtigsten erscheint dies beim Tabellenmodell. Es orientiert sich sehr stark am CALS-Modell, einige kleinere Anpassungen werden vorgenommen.

Damit keine inkonsistenten Formate auftreten und die Verwaltung der Absatzformate zentral geschehen kann, werden die Elemente soweit wie möglich bereits im EDD formatiert. Die Möglichkeit, Formate aus dem Absatzformatkatalog des Templates zuzuweisen bleibt damit weitestgehend ungenutzt. Diese Entscheidung wurde getroffen, nachdem der Absatzformatkatalog zunächst durch kontextbedingte Formatzuweisung aus dem EDD immer mehr Formate enthielt, die aufgrund einer stetig ansteigenden Anzahl von Bedingungen in der EDD angelegt werden mussten. Der Absatzformatkatalog war so groß geworden, dass eine kontinuierliche Pflege der Formate schwierig wurde. Innerhalb der EDD werden Formate hingegen vererbt, so dass sich hier Anpassungen durch Formatierungsänderungen weniger Schlüsselelemente realisieren lassen.

Da Layoutsteuerungen durch den Benutzer nicht mehr über Absatzkataloge vorgenommen werden sollen, müssen diese, soweit nötig, über Attribute in der Struktur ermöglicht werden, die der Benutzer verändern kann. Bedingte Formatierungen in der EDD werten diese Attribute aus und verändern dementsprechend das Erscheinungsbild innerhalb des Dokuments. Ein Attribut für die Seitenumbruchssteuerung wurde in dieser Weise eingeführt. Ändert der Benutzer das Attribut `AufNeueSeite` von der Voreinstellung `nein` auf `ja`, wird das entsprechende Element automatisch mithilfe bedingter Formatierung auf einer neuen Seite platziert. Die Layoutinformationen werden so auch in den XML-Dateien gespeichert.

Die Grenzen der Buchfunktion in XML-Anwendungen müssen mitbedacht werden. Das kleinstmögliche Modul im Buch ist ein Kapitel, die im Buch erzeugten Inhaltsverzeichnisse und andere Listen sind unstrukturiert. Jede strukturierte Datei im Buch hat ein Wurzelelement (Kapitel, Anhang), das im EDD zur höchsten gültigen Ebene deklariert werden muss. Gleichzeitig ist das Buch eine höchste gültige Ebene im EDD. In XML-DTDs gibt es hier einen wichtigen Unterschied: In XML kann es nur ein Wurzelelement geben, das alle anderen Elemente enthält. Daraus folgt, dass nur ein gesamtes Buch als XML-Datei bzw. als eine Hauptdatei mit mehreren eingebetteten Entity-Dateien, die ebenso nur als Gesamtheit funktioniert, exportiert werden kann. Der XML-Export wird deshalb immer aus einem strukturierten Buch gestartet. Beim Importieren bzw. Öffnen von XML-Dateien, die mit der Anwendung erstellt wurden, wird umgekehrt automatisch eine FrameMaker-Buchdatei erzeugt.

Beim Konzipieren der Dokument-Struktur wurde auch über den Einsatz erweiterter Technologien, wie XSLT, einem Structured API Client oder einer Datenbankanbindung nachgedacht. Der Implementierungsaufwand dieser Lösungen schien allerdings



In den Elementdefinitionen der erstellten Anwendung wird oft in den Textformatregeln eine bedingte Formatierung über Kontextregeln festgelegt. Dadurch ist es z.B. möglich, dass das Element *Ueberschrift* in einem Kapitel durch FrameMaker anders formatiert wird als in einem Abschnitt.

Abbildung 2.7: Bedingte Formatierung mithilfe der EDD

im Vergleich zum Nutzen zu hoch zu sein. Der Einsatz eines XSLT-Stylesheets wurde erst beschlossen, nachdem die Beschränkungen der Buchfunktion in strukturierten Anwendungen beim Entwickeln deutlich wurden (siehe Abschnitt 2.3.6). Ein Structured API Client ist momentan nicht umsetzbar und wird als mögliche spätere Erweiterung der Möglichkeiten der XML-Anwendung verstanden.

2.3 Erstellen der Anwendung

Die vorhergehenden Abschnitte dieser Arbeit beschäftigten sich mit Vorüberlegungen, nach denen die Dokumentstruktur für die XML-Anwendung entwickelt wurde. In diesem Abschnitt wird schließlich dokumentiert, wie die Anwendung Schritt für Schritt erstellt wurde.

2.3.1 Entwickeln der Elementdefinitionen (EDD)

Da der Ausgangspunkt der Entwicklung eine bestehende Dokumentation in FrameMaker ist, wird zuerst das EDD entwickelt. Die dazu nötigen Informationen werden aus dem Struct. App. Dev. Guide [2005, S. 125–158] gewonnen.

Die Dokumentstruktur wird in den sogenannten allgemeinen Regeln im EDD angegeben. Allgemeine Regeln bestimmen, welche Elemente in welchen Elementen enthalten sind, wodurch sich eine Verschachtelung bzw. baumartige Hierarchie der Elemente ergibt. Dabei müssen auch die höchsten gültigen Elemente definiert werden. Die höchsten gültigen Elemente entsprechen jeweils einem Wurzelelement eines FrameMaker-Dokuments, dessen Struktur durch das EDD bestimmt wird. Zusätzlich können Attributlisten für die einzelnen Elemente angegeben werden.

Die Formatierung der Elemente wird in sogenannten Textformatregeln festgelegt. Dabei wird das Konzept der Vererbung von Formaten eines Elements an dessen Kindelemente genutzt. Formate, die für alle Absätze im gesamten Dokument gelten sollen, wie z. B. die Grundschrift, können so in den Wurzelelementen einfach definiert und angepasst werden. Erweiterte Möglichkeiten ergeben sich zudem durch bedingte Formatierung. Dabei verändert sich die Formatierung eines Elements je nach seinen vorgeordneten, nachgeordneten oder übergeordneten Elementen und je nach seinen oder deren Attributwerten.

Es gibt eine große Anzahl spezieller Elemente und Attribute, die verwendet werden müssen, um sämtliche Funktionalitäten von FrameMaker in einer Struktur abzubilden. Dazu gehören z. B. Querverweiselemente, Tabellenelemente, Grafikelemente oder Elemente mit Textbereichformatierung (Inline-Elemente). Eine komplette Übersicht würde den Rahmen dieser Diplomarbeit sprengen. Deshalb sei an dieser Stelle auf den Struct. App. Dev. Guide [2005] verwiesen.

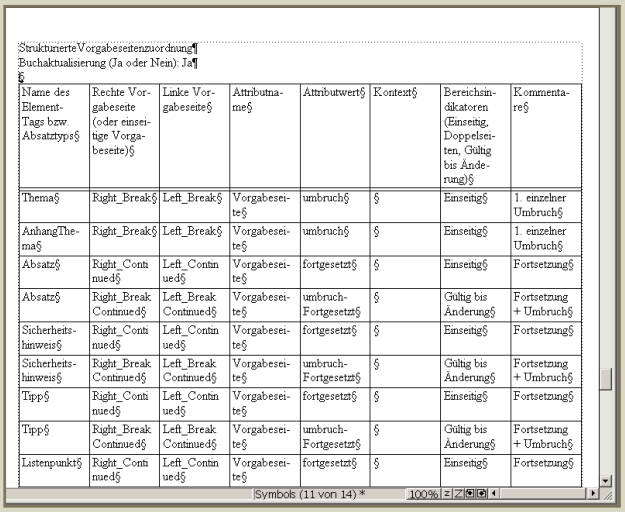
2.3.2 Entwickeln des Templates

Das Entwickeln des Templates geschieht nicht losgelöst vom Entwickeln der Elementdefinitionen. Wichtigster Bestandteil eines strukturierten Templates ist ein Elementkatalog, der mithilfe eines EDD importiert wird. Die Formatierungsinformationen, die im EDD stehen, benötigen dabei aber auch die unstrukturierten Komponenten eines

„normalen“ Templates, die z. B. in benannten Rahmen auf Referenzseiten bestehen. Die Namen dieser Rahmen müssen mit den im EDD aufgerufenen Namen übereinstimmen (benannte Rahmen können vor oder nach Absätzen an ihr Format „angehängt“ werden). Beim Anlegen eines neuen Templates wird ein leeres FrameMaker-Dokument mit Formaten versehen. Durch Importieren der Elementdefinitionen aus einem EDD wird ein strukturiertes Template daraus.

Es gibt in strukturierten Templates im Vergleich zu unstrukturierten Templates eine Reihe von Veränderungen. An Stelle von Absatzformatnamen müssen z. B. Elementnamen oder Attribute aus dem EDD in Format- und Seitenvorlagen angegeben werden. Das gilt u. a. für die Variablen in den Kopf- und Fußzeilendefinitionen und für die Querverweisformate.

Die Definitionen der Seitenlayouts können nicht in der EDD gespeichert werden und müssen daher im unstrukturierten Teil des Templates, den Vorgabeseiten, angegeben werden. Mithilfe des strukturierten Templates können den Arbeitsseiten (also den Seiten im späteren Dokument) Vorgabeseiten automatisch zugewiesen werden. Das nach dem „Try and Error“-Verfahren erschlossene Prinzip nennt sich „strukturierte Vorgabeseitenzuordnung“. Es handelt sich um ein undokumentiertes Feature, mit dem durch eine Tabelle auf den Referenzseiten bestimmten Elementen bestimmte Vorgabeseiten zugeordnet werden können. Dadurch ist es möglich, ein bestimmtes Seitenlayout auf spezielle Seiten oder Abschnitte im Dokument anzuwenden (z. B. Firmenlogo auf jeder ersten Seite eines Kapitels).



StrukturierteVorgabeseitenzuordnung¶
Buchaktualisierung (Ja oder Nein): Ja¶
§

Name des Element-Tags bzw. Absatztyps§	Rechte Vorgabeseite (oder einseitige Vorgabeseite)§	Linke Vorgabeseite§	Attributname§	Attributwert§	Kontext§	Bereichsindikatoren (Einseitig, Doppelseiten, Gültig bis Änderung)§	Kommentare§
Thema§	Right_Break§	Left_Break§	Vorgabeseite§	umbruch§	§	Einseitig§	1. einzelner Umbruch§
AnhangThema§	Right_Break§	Left_Break§	Vorgabeseite§	umbruch§	§	Einseitig§	1. einzelner Umbruch§
Absatz§	Right_Continued§	Left_Continued§	Vorgabeseite§	fortgesetzt§	§	Einseitig§	Fortsetzung§
Absatz§	Right_BreakContinued§	Left_BreakContinued§	Vorgabeseite§	umbruch-Fortgesetzt§	§	Gültig bis Änderung§	Fortsetzung + Umbruch§
Sicherheitshinweis§	Right_Continued§	Left_Continued§	Vorgabeseite§	fortgesetzt§	§	Einseitig§	Fortsetzung§
Sicherheitshinweis§	Right_BreakContinued§	Left_BreakContinued§	Vorgabeseite§	umbruch-Fortgesetzt§	§	Gültig bis Änderung§	Fortsetzung + Umbruch§
Tipp§	Right_Continued§	Left_Continued§	Vorgabeseite§	fortgesetzt§	§	Einseitig§	Fortsetzung§
Tipp§	Right_BreakContinued§	Left_BreakContinued§	Vorgabeseite§	umbruch-Fortgesetzt§	§	Gültig bis Änderung§	Fortsetzung + Umbruch§
Listenpunkt§	Right_Continued§	Left_Continued§	Vorgabeseite§	fortgesetzt§	§	Einseitig§	Fortsetzung§

Symbols (11 von 14) * 100%

Die strukturierte Vorgabeseitenzuordnung wird mit einer Tabelle auf den Referenzseiten gesteuert.

Anstelle von Absatzformaten, die bei der ähnlich funktionierenden unstrukturierten Vorgabeseitenzuordnung ausgewertet werden, berücksicht der strukturierte Automatismus Elemente in einem bestimmten Kontext (Attribute, vorgeordnete, nachgeordnete, übergeordnete Elemente, usw.).

Bei der erstellten Anwendung wird die strukturierte Vorgabeseitenzuordnung benutzt, um Umbrüche in Topics automatisch mit Fortsetzungszeilen an den entsprechenden Seitenanfängen und -enden zu versehen.

Abbildung 2.8: Die strukturierte Vorgabeseitenzuordnung

2.3.3 Das Übersetzen in XML – Die Lese-/Schreibregeln

Die Lese-/Schreibregeln bedienen sich einer eigenen Skriptsprache, die ausführlich im Struct. App. Dev. Guide [2005, S. 259–265] erklärt wird. Für Elemente wird in den Lese-/Schreibregeln jeweils eine Regel angegeben, die bestimmt, wie diese Elemente von und in Markup übersetzt werden. Die „Sicht“ der Lese-/Schreibregeln ist dabei auf das XML-Element bezogen, die Regeln funktionieren aber zum größten Teil beim Import und Export. Die folgende Regel

```
element "book" is fm element "Buch";
```

übersetzt das XML-Element „book“ in das FrameMaker-Element „Buch“. Umgekehrt kann sie nicht angegeben werden (das ist mit der „Sicht“ der Lese-/Schreibregeln gemeint). Aus der Regel wird auch ersichtlich, wie die Lese-/Schreibregeln in der Anwendung benutzt werden. Zwischen Elementen mit deutschen Namen in der FrameMaker-Struktur und den in der XML-Struktur abgebildeten Elementen mit universellen englischen Namen wird übersetzt. Als Besonderheit der Anwendung werden die Namen der FrameMaker-Elemente sowohl in den Lese-/Schreibregeln als auch im EDD als (gleichnamige) FrameMaker-Variablen angegeben. Dadurch steht das gesamte „Interface“ der Anwendung, das über die Strukturansicht und den Elementkatalog im Programm zugänglich ist, in der Muttersprache der bedienenden Redakteure (also zunächst erst einmal auf Deutsch) zur Verfügung. Eine Lokalisierung der Anwendung ist sehr leicht zu bewerkstelligen, indem die Variablendefinitionen im EDD angepasst werden und die veränderten Variablen in die Lese-/Schreibregeln importiert werden. Die Namen der Elemente, Attribute und Attributwerte bleiben dabei in den XML-Daten trotzdem stets gleich.

Bei der Lokalisierung ist zu beachten, dass in Lese-/Schreibregeln keine Doublebyte Characters, d. h. keine über den ASCII-Zeichensatz hinausgehende Zeichen enthalten sein dürfen. Da dies auch die deutschen Umlaute betrifft, dürfen diese in den Elementdefinitionen mithilfe von Variablen nicht in die Variablenwerte geschrieben werden. Eine korrekte Regel für ein Überschriftelement lautet dementsprechend:

```
element "head" is fm element "Ueberschrift";
```

Werden für Elemente keine Lese-/Schreibregeln angegeben, dann würden die FrameMaker-Elemente mithilfe von Defaultmechanismen nach XML übersetzt. Das obige Element „Ueberschrift“ würde ohne die entsprechende Regel beim XML-Export als „Ueberschrift“ im Markup erscheinen. Andersherum würde das XML-Element „head“ beim Import als „head“ im FrameMaker erscheinen. Damit diese Mechanismen funktionieren, müssen die entsprechenden Elemente jeweils im EDD und in der DTD richtig definiert sein, sonst bricht der Vorgang mit einer Fehlermeldung ab.

Zu Fehlern kommt es vor allem bei erweiterten Regeln für spezielle Elemente wie Querverweise oder Tabellen. Erschwerend kommt bei der Entwicklung hinzu, dass hier die Default-Mechanismen und ihre Erklärungen in der Dokumentation nicht zufrieden stellend funktionieren. Das trifft vor allem auf Tabellen zu, die laut Dokumentation bei einer Einhaltung bestimmter Elementnamen im EDD automatisch in das CALS-Modell umgewandelt werden, sofern die CALS-Tabellenelemente auch in der DTD definiert sind. Nach den Ergebnissen des Autors funktioniert dieser Mechanismus nicht, was an der deutschen Version des FrameMaker liegen kann. Auch für andere spezielle Elemente müssen Regeln angegeben werden, da es sonst zu Fehlern kommt. Verlässt sich der Entwickler z. B. auf den Defaultmechanismus bei der Übersetzung von Querverweiselementen, werden die in XML erzeugten leeren Elemente nicht geschlossen, wodurch das XML nicht wohlgeformt ist.

Für eine funktionierende XML-Anwendung, die das CALS-Tabellenmodell benutzt, müssen nach den Ergebnissen des Autors mindestens die folgenden Lese-/Schreibregeln angegeben werden:

```
reader line break is forced return;

writer line break is 999999 characters;

element "table"
{
   is fm element "Tabellencontainer";
   attribute "tabstyle"    is fm property table format;
   attribute "frame"
   {
      is fm property table border ruling;
      value "top"          is fm property value top;
      value "bottom"       is fm property value bottom;
      value "topbot"       is fm property value top and bottom;
      value "all"          is fm property value all;
      value "sides"        is fm property value sides;
      value "none"         is fm property value none;
   }
   attribute "colsep"       is fm property column ruling;
   attribute "rowsep"       is fm property row ruling;
   attribute "orient"       is fm attribute;
   attribute "pgwide"       is fm property page wide;
}

element "tgroup"
{
   is fm table element "TABELLE";
   attribute "cols"         is fm property columns;
   attribute "tgroupstyle"  is fm property table format;
   attribute "colsep"       is fm property column ruling;
   attribute "rowsep"       is fm property row ruling;
   attribute "align"        is fm attribute;
   attribute "charoff"      is fm attribute;
   attribute "char"         is fm attribute;
}

element "colspec"
{
   is fm colspec;
   attribute "colnum"   is fm property column number;
   attribute "colname"  is fm property column name;
   attribute "align"    is fm property cell alignment type;
   attribute "charoff"  is fm property cell alignment offset;
   attribute "char"     is fm property cell alignment character;
   attribute "colwidth" is fm property column width;
   attribute "colsep"   is fm property column ruling;
   attribute "rowsep"   is fm property row ruling;
}

element "spanspec"
```

```
{
   is fm spanspec;
   attribute "spanname" is fm property span name;
   attribute "namest"   is fm property start column name;
   attribute "nameend"  is fm property end column name;
   attribute "align"    is fm property cell alignment type;
   attribute "charoff"  is fm property cell alignment offset;
   attribute "char"     is fm property cell alignment character;
   attribute "colsep"   is fm property column ruling;
   attribute "rowsep"   is fm property row ruling;
}

element "thead"
{
   is fm table heading element "KOPFZEIL";
   attribute "valign" is fm attribute;
}

element "tfoot"
{
   is fm table footing element "FUSSZEIL";
   attribute "valign" is fm attribute;
}

element "tbody"
{
   is fm table body element "HAUPTTXT";
   attribute "valign" is fm attribute;
}

element "row"
{
   is fm table row element "REIHE";
   attribute "valign" is fm attribute;
   attribute "rowsep" is fm property row ruling;
}

element "entry"
{
   is fm table cell element "ZELLE";
   attribute "colname"  is fm property column name;
   attribute "namest"   is fm property start column name;
   attribute "nameend"  is fm property end column name;
   attribute "spanname" is fm property span name;
   attribute "morerows" is fm property more rows;
   attribute "colsep"   is fm property column ruling;
   attribute "rowsep"   is fm property row ruling;
   attribute "rotate"   is fm property rotate;
   attribute "valign"   is fm attribute;
   attribute "align"    is fm attribute;
```

```
   attribute "charoff"  is fm attribute;
   attribute "char"     is fm attribute;
}

element "title" is fm table title element "TITEL";

element "crossref" is fm crossref element "Querverweis";
```

Die FrameMaker-Namen in dieser Übersicht können natürlich auch anders vergeben werden. Die versalen Namen der Tabellenelemente beziehen sich auf die Elementnamen, die in der Strukturansicht eines strukturierten Dokuments sichtbar werden, wenn man eine aus einem unstrukturierten Dokument kopierte Tabelle einfügt. Dadurch, dass diese Namen im EDD und in den Lese-/Schreibregeln verwendet werden, wird das Übernehmen von Tabellen aus unstrukturierten Dokumenten durch einfaches Kopieren und Einfügen möglich. Die Tabelle muss jedoch für eine funktionierende Übersetzung in das CALS-Modell in einen Tabellencontainer eingebettet werden (wie in den Regeln oben angegeben). Der Struct. App. Dev. Guide [2005] erläutert andere Möglichkeiten, die aber nach den Ergebnissen des Autors nicht zufriedenstellend sind, da so nicht alle Eigenschaften der FrameMaker-Tabellen in XML abgebildet werden können.

Die am Anfang angegebenen `line break`-Regeln konfigurieren die XML-Ausgabe von FrameMaker so, dass die Texte, die man in einem strukturierten Dokument innerhalb eines Absatzes schreibt, auch genauso in die XML-Dateien übernommen werden. Das Problem, das ohne Angabe dieser Regeln auftaucht lässt sich darauf zurückführen, dass in Texten innerhalb eines XML-Elements nach der XML-Spezifikation nicht zwischen Leerzeichen und Seitenumbruchzeichen unterschieden wird. Dementsprechend ist das durch FrameMaker erzeugte XML normalerweise von allen durch den Benutzer mit Shift+Return eingegebenen Zeilenumbrüchen innerhalb eines Absatzes befreit. Zur besseren Lesbarkeit würde außerdem etwa alle 80 Zeichen ein Zeilenumbruch in der XML-Datei eingefügt. Dieses Verhalten des Programms wird mit den `line break`-Regeln unterbunden. Die erste Regel sorgt dafür, dass FrameMaker Zeilenumbrüche im Text innerhalb von XML-Elementen als `forced return`, also als mit Shift+Return erzwungene Zeilenumbrüche interpretiert. Die zweite Regel bewirkt, dass das Programm nicht mehr nach ca. 80 Zeichen einen Zeilenumbruch in XML-Dateien erzeugt. Dazu wird hier ein möglichst hoher Wert für die Anzahl der Zeichen angegeben (`999999` sollte reichen).

2.3.4 Automatisches Erzeugen und manuelles Korrigieren der DTD

Aus einer EDD kann eine DTD automatisch erzeugt werden. Wenn man dafür eine Anwendung mit Lese-/Schreibregeln benutzt, werden diese berücksichtigt. Es ist darum günstig, die Lese-/Schreibregeln nach dem EDD zu entwickeln und dabei gleichzeitig die DTD nach jeder neu angegebenen Regel automatisch zu erzeugen. Dabei kann man auch abschätzen, wie sich die Lese-/Schreibregel auswirkt und ob diese schon in der gewünschten Art und Weise funktioniert. Zum Erzeugen einer DTD aus einer EDD wird im Menü „Datei → Strukturierungswerkzeuge → Sichern als DTD. . .“ gewählt.

Wird der Automatismus zum Erzeugen von DTDs benutzt, kommt es jedoch zu Problemen bei Mixed Content. In XML-DTDs ist für solche Inhaltsmodelle von Elementen,

die sowohl Text als auch andere Elemente innerhalb eines Elements als gültigen Inhalt definieren, nur eine Schreibweise zulässig, die wie folgt aussieht:

```
<!ELEMENT irgendeins (#PCDATA | unterelement1 | unterelement2)* >
```

Diese Elementdefinition ließe sich um beliebig viele Unterelemente erweitern. Entscheidend ist, dass der Text (`#PCDATA`) nicht an einer anderen Stelle der Definition oder mit anderen Elementen gruppiert, sondern genau so wie oben angegeben, stehen darf. Der Automatismus des FrameMaker lässt diese Einschränkung des DTD-Modells, die es im EDD nicht gibt, unberücksichtigt. Die komplexen Inhaltsmodelle für Mixed Content, die im EDD leicht entstehen können, werden genauso wie dort angegeben in das DTD übernommen.

Deshalb müssen alle Mixed Contents in von FrameMaker automatisch erzeugten DTDs später von Hand mit einem beliebigen Editor in die oben gezeigte Form gebracht werden. Bei XML-Anwendungen mit zahlreichen Elementen kann dies ein komplexer Vorgang werden. Dieser Vorgang könnte eventuell abgekürzt werden, indem hier z. B. ein kleines Perl- oder PHP-Script als externes Zusatzprogramm zum Einsatz gebracht wird. Wichtig ist hierfür eine Programmiersprache, die reguläre Ausdrücke unterstützt.

In Teil B des Anhangs werden Auszüge aus der DTD, die zu der entwickelten XML-Anwendung gehört, wiedergegeben.

2.3.5 Aktualisieren der Anwendungsdefinitionen

In den Anwendungsdefinitionen wird eine neue XML-Anwendung angelegt. Darin werden die Speicherorte von Template und Lese-/Schreibregeln angegeben. Nachdem die DTD mithilfe des in Abschnitt 2.3.4 gezeigten Mechanismus erzeugt wurde, wird deren Speicherort ebenfalls in der Anwendungsdefinition angegeben. Wird zusätzlich ein XSLT-Stylesheet verwendet, muss hierfür ebenfalls angegeben werden, wo dieses auf der Festplatte des Benutzers gespeichert ist.

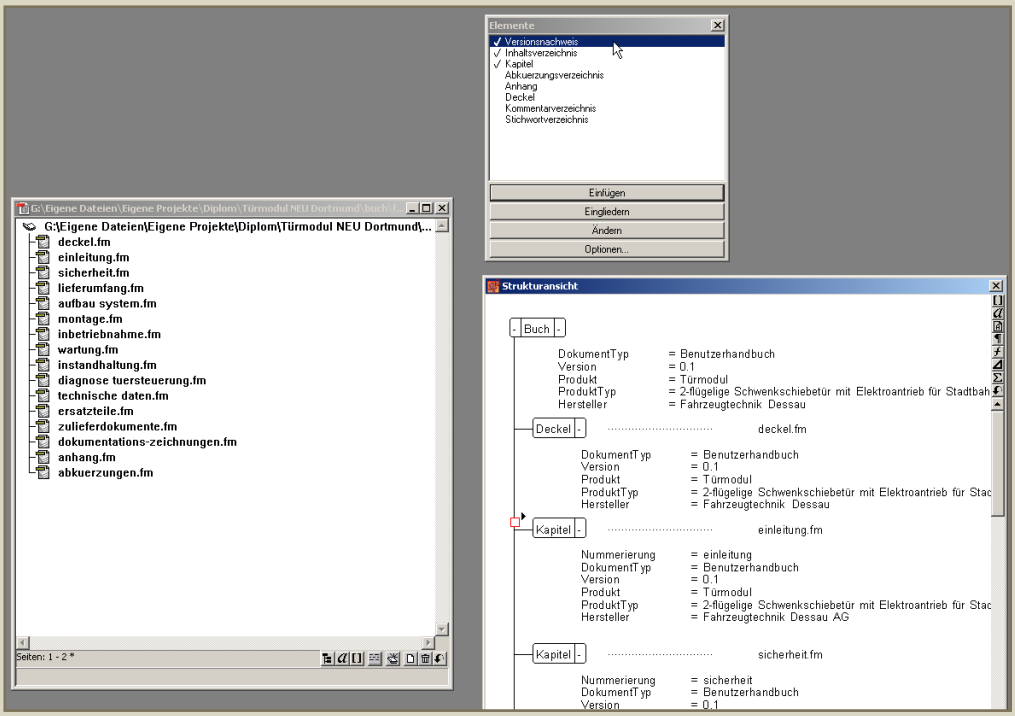
Für eine konsistente `structapps.fm`-Datei (in der die Anwendungsdefinitionen gespeichert sind), die auch zwischen verschiedenen Rechnern ausgetauscht werden kann, sollte man sich an den Speicherorten der vorinstallierten XML-Anwendungen von FrameMaker orientieren. Die Dateien liegen unterhalb des Installationspfads des Programms im Unterordner `Structure\xml\`. Für die Angabe des Unterordners `Structure` unterhalb des FrameMaker-Installationspfads gibt es in den Anwendungsdefinitionen die Schreibweise `$STRUCTDIR`, die man genauso bei der Definition eigener Anwendungen verwenden kann. Die Dateien der Anwendung werden unterhalb dieses Ordnders abgelegt. So kann die Anwendung leichter auf einem anderen Rechner installiert werden, indem man die `structapps.fm`-Datei mit den Anwendungsdefinitionen zusammen mit den zur Anwendung gehörenden Dateien auf den Rechner kopiert.

Beim Aktualisieren der Anwendungsinformationen muss natürlich auch ein eindeutiger Name für die neue XML-Anwendung angegeben werden. Als Arbeitstitel wurde hier das Akronym „dpML“ gewählt. „dpML“ steht dabei für „Documentation Publisher's Markup Language“. In den Anwenderdokumentationen, die in den Kapiteln 3 und 4 dieser Arbeit wiedergegeben werden, wird daher die Bezeichnung „dpML“ für

die Anwendung verwendet. Die Dateien tragen ebenso „dpML“ im Namen, das EDD heißt beispielsweise `dpmlEDD.fm`.

2.3.6 Verwenden der Buchfunktion in der Anwendung

In der Anwendung wird die Buchfunktion zum Zusammenstellen von Kapiteln und zum Festlegen von Seitennummerierungs-Einstellungen verwendet. Im strukturierten Buch fügt der Anwender für Inhaltsverzeichnis, Versionsnachweisliste (listet die Versionsnummern der unterschiedlichen Kapitel auf) und evtl. Kommentarliste (dient bei Korrekturversionen zum schnellen Auffinden von Kommentaren) leere Elemente dort ein, wo diese Listen später stehen sollen.



Die Buchfunktion von FrameMaker ist auch strukturiert sehr praktisch. Leider ist die Übersetzung aller Funktionen von FrameMaker-Büchern in XML-Daten mit den Lese-/Schreibregeln nicht möglich. In der erstellten Anwendung konnte diese Beschränkung durch XSLT teilweise aufgehoben werden.

Abbildung 2.9: Die strukturierte Buchfunktion

Die Buchdatei bildet die Quelle für den XML-Export. Dabei werden aus den leeren Elementen Inhaltsverzeichnis, Versionsnachweisliste, evtl. Kommentarliste und auch die Kapitelinhaltsverzeichnisse, die es in jedem Kapitel gibt, generiert. Dazu kommt ein XSLT-Stylesheet zum Einsatz.

Das exportierte XML wird wieder in das Programm importiert. Dabei entsteht automatisch eine Buchdatei. Die Seitennummerierungs-Einstellungen der einzelnen Kapitel müssen neu eingestellt werden, da es keine Möglichkeit gibt, die ursprünglichen Einstellungen beim XML-Export zu übernehmen. Evtl. erzeugt der Anwender noch einen Standardindex. Diese Liste wird beim Export nicht automatisch durch das XSLT

generiert, sondern wird bisher nur unstrukturiert verwendet. Danach wird das PDF erzeugt.

Generell gibt es große Beschränkungen der Möglichkeiten eines Buches innerhalb von XML-Anwendungen. Die Buchfunktion ist an sich sehr praktisch, es gibt aber zu wenige Möglichkeiten, den Export und Import von Markup aus und in Buchdateien mithilfe von Lese-/Schreibregeln zu konfigurieren. Beim Export von über die Buchfunktion generierten Listen (z. B. Inhaltsverzeichnisse) verlieren diese nach dem Reimport ihren Status als generierte Listen, wodurch sie nutzlos werden. Den Exportvorgang kann man mit generierten Listen nur starten, wenn diese als „Buchkomponenten“ in der Strukturansicht des Buches erscheinenden Elemente in ein übergeordnetes Element eingebettet werden. Das übergeordnete Element kann diese Buchkomponenten nur aufnehmen, wenn sein Inhaltsmodell mit „ANY“ definiert ist. Die entsprechenden Elemente werden mithilfe von Lese-/Schreibregeln beim XML-Export geleert. In der XML-Struktur bleibt so zumindest die Information erhalten, an welcher Stelle im Dokument eine generierte Liste stehen soll. Zu den Beschränkungen der Buchfunktion in XML-Anwendungen gibt es zwei Diskussionen in Adobe Foren [Adobe Foreneintrag 2006b; Adobe Foreneintrag 2006a].

2.3.7 XSLT für strukturierte Inhaltsverzeichnisse, Versionsnachweisliste und Kommentarverzeichnis

Ein XSLT-Stylesheet wird in der XML-Anwendung verwendet, um Inhaltsverzeichnisse, Versionsnachweisliste und Kommentarverzeichnis strukturiert zu erstellen. Bei der Versionsnachweisliste gibt es keine Alternative zu dieser Vorgehensweise, da sie Inhalte enthält, auf die über unstrukturierte Listen mit der Buchfunktion nicht zugegriffen werden kann. Bei den Inhaltsverzeichnissen ersetzt das Stylesheet die unstrukturierten Listen der Buchfunktion und sorgt für die automatische Generierung beim Speichern von XML.

Dabei werden diese Verzeichnisse als Listen formatierter Querverweise verstanden, die mithilfe von XSLT generiert werden. Querverweise mit strukturiertem FrameMaker funktionieren nach einem einfachen Prinzip, das auch aus XML bekannt ist. Jedes Element, das als Ziel eines Querverweis dienen können soll, muss ein eindeutiges ID-Attribut besitzen. Jedes Querverweiselement besitzt ein IDREF-Attribut. Damit das Ziel eines Querverweis bestimmt ist, muss dessen IDREF-Attribut mit dem ID-Attribut eines Elements im Dokument übereinstimmen.

Voraussetzung für den Einsatz von XSLT beim Generieren der Listen ist, dass die Struktur sowohl in der EDD als auch in der DTD darauf abgestimmt ist, dass die entsprechenden Elemente (z. B. Inhaltsverzeichnis) sowohl leer sein, als auch die Liste der Querverweise mit ihrer spezifischen Struktur enthalten können. Die Inhalte im strukturierten Dokument werden vor dem Speichern in XML auf Gültigkeit gegenüber dem EDD geprüft. Nach dem Reimport müssen die enthaltenen Daten ebenfalls valide sein. Das gleiche gilt für das gespeicherte XML, das doppelt geprüft wird: einmal vor dem Anwenden von XSLT und einmal danach.

Das XSLT-Stylesheet funktioniert relativ einfach für Inhaltsverzeichnisse, indem aus allen Elementen, die aufgenommen werden sollen, die Werte der ID-Attribute für die Werte der IDREF-Attribute der Querverweise übernommen werden. Weil FrameMaker

aber erst ein ID-Attribut für ein Element automatisch erzeugt, wenn auf dieses Element verwiesen wird, haben normalerweise nicht alle Elemente, die in das Verzeichnis aufgenommen werden sollen, ein ID-Attribut. Deshalb muss für alle Elemente, die im Inhaltsverzeichnis aufgeführt werden sollen und die noch kein ID-Attribut besitzen, dieser Attributwert mit der XSLT-Funktion `generate-id` erzeugt werden. Ein kurzer Ausschnitt aus dem Stylesheet soll das verdeutlichen. Das ID-Attribut heißt in der Anwendung „fmid“, der Einfachheit halber werden für dieses Beispiel nur Kapitelüberschriften in das Inhaltsverzeichnis übernommen:

```
<!-- ID-Wert entweder übernehmen
     oder erzeugen, wenn nicht vorhanden -->
<!-- (template zum aufrufen in anderen templates) -->
<xsl:template name="make_fmid-att">
  <xsl:attribute name="fmid">
    <xsl:choose>
      <xsl:when test="not(@fmid)">
        <xsl:value-of select="generate-id()"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="@fmid"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
</xsl:template>

<!-- ID-REF-Wert des current()-Kontextknoten entweder übernehmen
     oder erzeugen, wenn nicht vorhanden -->
<!-- (template zum aufrufen in anderen templates) -->
<xsl:template name="make_current-fmidref-att">
  <xsl:attribute name="fmidref">
    <xsl:choose>
      <xsl:when test="not(current()/@fmid)">
        <xsl:value-of select="generate-id(current())"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="current()/@fmid"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
</xsl:template>

<!-- Template für Überschrift -->
<xsl:template match="head">
  <xsl:copy>
    <xsl:call-template name="make_fmid-att"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

```
<!-- Template für Inhaltsverzeichnis -->
<xsl:template match="toc">
  <xsl:copy>
    <tochead />
    <xsl:for-each select="/book/chapter/head">
      <xsl:element name="tocentry">
        <xsl:attribute name="tocentrytype">chapterlevel</xsl:attribute>
        <xsl:element name="tocref">
          <xsl:call-template name="make_current-fmidref-att"/>
          <xsl:attribute name="format">TocRefChapter</xsl:attribute>
        </xsl:element>
      </xsl:element>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>
```

Aus dem Template wird ersichtlich, dass für die einzelnen Querverweise in der Liste neben dem IDREF-Attribut (`fmidref`) noch ein weiteres Attribut (`format`), das das Format des Querverweises angibt, erzeugt wird. Im umschließenden `tocentry`-Element gibt es zudem noch ein Attribut namens `tocentrytype`. Beide Attribute bestimmen das Format des Querverweises. Das eine wird für die bedingte Formatierung des Listenpunkts im EDD benutzt, das andere gibt das Querverweisformat und damit den Text an, den der Querverweis im Element schließlich erzeugt. Damit Inhaltsverzeichnisse mittels XSLT in der Anwendung funktionieren, müssen diese Strukturen genauso definiert werden, die bedingte Formatierung im EDD entsprechend vorgenommen werden und die benötigten Querverweisformate im unstrukturierten Teil des Templates zur Verfügung stehen.

Das Template für das Inhaltsverzeichnis verwendet u. a. noch die Methoden `xsl:copy` (kopiert den Knoten), `xsl:for-each` (Mehrfachverarbeitung; wirkt auf alle ausgewählten Knoten) und `xsl:call-template` (ruft ein benanntes Template innerhalb eines anderen auf). Eine nähere Beschreibung und Erläuterung des XSLT-Standards und seiner vielfältigen Anwendungsmöglichkeiten, von denen in der Anwendung noch einige mehr genutzt werden, unterbleibt an dieser Stelle. Nähere Informationen zu XSLT finden sich z. B. bei Tidwell und Lichtenberg [2003], ein Tutorial, das sich ebenfalls mit Verzeichnissen beschäftigt, bei Behme [2001].

2.3.8 Hinweise zum Einsatz von Grafiken

Hochwertige Grafiken in der erstellten XML-Anwendung verwenden zu können war eine wichtige Zielstellung der Entwicklung. Das Thema FrameMaker und Grafik ist nicht trivial [vgl. Gust 2005]. In der XML-Anwendung treten zusätzliche Probleme auf, da die Grafikdateien beim XML-Export automatisch mit exportiert und beim XML-Import genauso reimportiert werden. Um hier einen Qualitätsverlust zu vermeiden, sollten einige Hinweise beachtet werden.

Wegen der höheren Qualität bei kleineren Dateigrößen sollten Grafiken soweit wie möglich in Vektorformaten vorliegen. Die Ergebnisse des Autors zeigen jedoch, dass nicht alle Vektorformate in einer XML-Anwendung funktionieren. SVG wird nicht

richtig unterstützt, sondern gerastert, WMF führt zu eckigen Rundungen und seltsamen Ausfransungen. EPS funktioniert grundsätzlich gut in FrameMaker, scheitert aber beim XML-Export, wenn die EPS-Datei eine TIFF-Vorschau hat. Der Exportfilter scheitert an den für ihn doppelt (EPS und TIFF) vorliegenden Grafikinformationen und bricht mit einer Fehlermeldung ab. Da jede EPS-Datei, die im Programm angezeigt werden soll eine TIFF-Vorschau braucht, scheidet dieses Format für XML-Anwendungen vorerst aus.

Die Ergebnisse des Autors zeigen, dass der XML-Export am Besten mit CGM-Dateien funktioniert. Dieses Format ist für den Export bereits voreingestellt. Damit werden in die ausgegebenen Grafikdateien auch Beschriftungen, die mit den FrameMaker-eigenen Grafikwerkzeugen erstellt wurden, übernommen. Das Dateiformat CGM sollte daher in Grafiken für die XML-Anwendung stets verwendet werden. Kommt es trotzdem zu Problemen, kann der Import bzw. Export mithilfe von Lese-/Schreibregeln konfiguriert werden [vgl. Struct. App. Dev. Guide 2005, S. 348–371].

In der Elementdefinition wurde für das Grafikelement der Name „GRAPHIK“ vergeben. Analog zu Tabellen können dadurch Grafiken aus unstrukturierten FrameMaker-Dokumenten mittels „Copy and Paste“-Verfahren übernommen werden. Dabei sollte aber darauf geachtet werden, dass die eingefügte Grafik im CGM-Format vorliegt.

Entgegen den Empfehlungen des Struct. App. Dev. Guide [2005, S. 350] werden Grafiken in der XML-Anwendung nicht mithilfe von unparsed Entities im XML eingebettet, sondern mit dem eigentlich nur für die Verwendung mit SGML vorgesehenem „file“-Attribut [vgl. Struct. App. Dev. Guide 2005, S. 352] referenziert. Dies ist deshalb so realisiert worden, weil XSLT eingesetzt wird. Unparsed Entities haben damit nicht richtig funktioniert. Es gibt keine Möglichkeit in XSLT eine Anweisung zu schreiben, mit der auf den internal Subset der DTD eines XML-Dokuments, in dem FrameMaker die Unparsed Entities mit den Verweisen auf die Grafikdateien normalerweise abspeichert, zugegriffen werden kann. Deshalb gehen diese Informationen beim Einsatz von XSLT zwangsläufig verloren und es kommt zu Fehlermeldungen beim Import der XML-Dateien.

2.3.9 Hinweise zum Einsatz von Tabellen

Tabellen aus unstrukturierten Dokumenten lassen sich – ebenso wie Grafiken – mittels „Copy and Paste“-Verfahren in die mit der Anwendung erstellten Dokumente übernehmen. Deshalb haben die Tabellenelemente auch seltsam anmutende Namen (TABELLE, KOPFZEIL, FUSSZEIL, etc.). Die Namen sind nach der Struktur gewählt worden, die sichtbar wird, wenn man eine aus einem unstrukturierten Dokument kopierte Tabelle in ein strukturiertes Dokument einfügt.

Die Möglichkeit des Einfügens von Tabellen aus unstrukturierten Dokumenten sind allerdings begrenzt. Werden in der Tabelle Absatz- oder Zeichenformate verwendet, dann sehen diese zwar auch im strukturierten Dokument zunächst genauso wie im unstrukturierten aus, da hier die entsprechenden Formate übernommen werden. Damit die Formatinformationen nicht verloren gehen müssen sie aber trotzdem noch in Elementstrukturen abgebildet werden. Ein mittels Zeichenformat hervorgehobenes Wort innerhalb einer Tabelle muss vom Anwender z. B. in ein entsprechendes Element eingebettet werden. Wird das vernachlässigt, gehen die Formatierungen beim XML-Export und -Reimport verloren.

Die Tabellen werden in XML mithilfe des CALS-Standards abgebildet. Der im Anhang des Struct. App. Dev. Guide [2005, S. 545–550] angegebene DTD-Abschnitt ist – auch wenn dies zu erwähnen vergessen wird – nur für SGML gültig und konnte deshalb nicht für die XML-Anwendung verwendet werden. Für ein entsprechendes Modell in XML-DTD-Syntax wurde die Spezifikation des XML Exchange Table Model der OASIS verwendet, da diese ebenfalls CALS-basiert ist [vgl. OASIS 1999]. Für maximale Kompatibilität mit dem FrameMaker-Tabellenmodell wurden die darin fehlenden Attribute des CALS-Modells, die im Struct. App. Dev. Guide [2005] beschrieben werden, ergänzt.

Für die Übersetzung von Tabellen reicht laut Struct. App. Dev. Guide [2005] die Angabe des CALS-Tabellenmodells in der DTD, damit die Übersetzung automatisch funktioniert. Die Ergebnisse des Autors bestätigen dies jedoch nicht. Evtl. funktioniert dieser Automatismus nur mit einer SGML-Anwendung zufrieden stellend oder das Tabellenmodell muss anders angegeben werden. Aus der Dokumentation war dies aber nicht ersichtlich. Mithilfe von Lese-/Schreibregeln für Tabellen (siehe Abschnitt 2.3.3 auf Seite 43) gelingt die Verwendung des CALS-Modells trotzdem. Die entsprechenden Lese-/Schreibregeln werden interessanterweise in ähnlicher Form auch im Struct. App. Dev. Guide [2005, S. 551–554] angegeben. Dort wird aber dargelegt, dass diese Regeln nicht benötigt werden, sondern nur zeigen sollen, wie die implementierten Automatismen funktionieren. Der Benutzer soll auf der Basis dieser Anschauung eine Übersetzung in ein anderes Tabellenmodell als CALS in Angriff nehmen können.

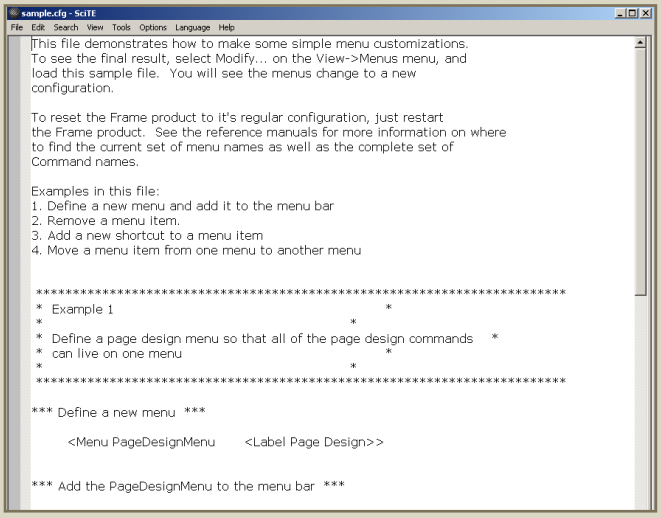
Damit möglichst viele der Eigenschaften von Tabellenelementen des FrameMaker-eigenen Modells nach CALS übersetzt werden, muss auch im EDD ein Containerelement um die eigentliche Tabelle definiert werden (siehe Abschnitt 2.3.3 auf Seite 43). Daraus ergibt sich außerdem der Vorteil, dass ein Tabellenelement dadurch am Anfang einer Seite allein stehen kann. Normalerweise wird ein Tabellenelement immer an einem Absatz verankert, der auf der gleichen Seite wie die Tabelle stehen muss. Dieser Absatz wird nun durch den Tabellencontainer erzeugt.

Eine weitere Beschränkung beim Einsatz von Tabellen ergibt sich dadurch, dass mehrere Tabellenelemente durch die XML-Anwendung nicht unterstützt werden. Diese Beschränkung ergibt sich noch nicht bei der Definition in der EDD, wo verschiedene Tabellen für unterschiedliche Formatierungszwecke mit unterschiedlichen Anfangsformaten effektiv verwendet werden können. Die Beschränkung wird durch die Lese-/Schreibregeln verursacht, mit denen nach den Ergebnissen des Autors ein XML-Export und -Import von mehreren Tabellen nicht bewerkstelligt werden kann.

2.3.10 Installation der Anwendung

Um eine Anwendung, die nach vielen Tests erfolgreich auf dem Rechner des Entwicklers läuft, auf den Rechner eines Redakteurs, der mit dieser Anwendung arbeiten soll, zu übertragen, müssen alle zur Anwendung gehörenden Dateien auf seinen Rechner kopiert werden. Dazu gehören das Template, die DTD, die Lese-/Schreibregeln und die Anwendungsdefinitionen.

Die Speicherorte müssen dabei wie auf dem Ausgangsrechner gewählt werden bzw. genau so, wie in den Anwendungsdefinitionen (der `structapps.fm`-Datei) festgelegt.



Im Unterordner *fminit\configui* des FrameMaker-Verzeichnisses gibt es einige Dateien mit der Endung *.cfg*. In der Datei *sample.cfg* wird erklärt, wie mithilfe dieser Dateien die Menüs von FrameMaker komplett angepasst werden können.

Dadurch ist es möglich, die Oberfläche für ein auf FrameMaker basierendes Redaktionssystem genau an die Erfordernisse der eigenen Anwendung anzupassen.

Die CFG-Dateien sind reine Textdateien, die mit einem beliebigen Editor bearbeitet werden können.

Abbildung 2.10: Anpassung der FrameMaker-Menüs

Voraussetzung dafür ist natürlich eine korrekt installierte und lizenzierte FrameMaker-Version auf dem Computer des Redakteurs. Zusätzlich müssen alle Schriften installiert sein, die in der Anwendung verwendet werden. Die Konfiguration von FrameMaker kann für die Bedienung der Anwendung optimiert werden. Dazu gibt es in der FrameMaker-Installation einige Dateien mit der Endung „.ini“ oder „.cfg“, die ebenfalls angepasst und mit der Anwendung weitergegeben werden können [vgl. FrameMaker Anpassungen 2002].

Fazit

In dieser Diplomarbeit wurde erörtert, wie sich mithilfe von XML-Anwendungen für Adobe FrameMaker die Redaktionsprozesse in der Technischen Dokumentation optimieren lassen. Es wurde aufgezeigt, dass das FrameMaker-Konzept der XML-Anwendung die Vorteile des klassischen DTP mit den Vorteilen von XML verbindet und dabei in seiner Konsequenz bisher einzigartig ist. Die Möglichkeiten, die sich durch den Einsatz von XML-Anwendungen für Adobe FrameMaker ergeben, sind daher enorm.

Die FrameMaker-internen Strukturierungsmöglichkeiten sind durch den mithilfe eines EDD erzeugten Elementkatalogs sehr weit entwickelt. Die strukturierte Arbeitsweise mit dem Programm, die mit den zusätzlichen Bedienfenstern Elementkatalog und Strukturansicht möglich wird, vereinfacht die Bedienung des ansonsten sehr komplexen Programms nach Ansicht des Autors deutlich.

Während der Entwicklung der XML-Anwendung zeigte sich allerdings ebenso deutlich, dass Adobe die Strukturierungsfunktionen für SGML und XML nachträglich in das bestehende DTP-Programm implementiert hat und einige der Übersetzungsfunktionen zwischen Markup und FrameMaker-eigener Struktur nicht oder nicht so, wie dokumentiert, funktionieren. Hier sind bei der Entwicklung viele Workarounds nötig, zumal die Beschreibungen in der Dokumentation zum Teil nicht ausführlich genug ausfallen oder sogar widersprüchlich sind. Die Entwicklung einer eigenen XML-Anwendung wird dadurch gebremst.

Es gibt einige nach Ansicht des Autors wichtige Möglichkeiten, die sich in der vorgenommenen Eigen-Entwicklung einer XML-Anwendung durch die derzeitigen Beschränkungen beim XML-Export und -Import nicht bewerkstelligen ließen. Dazu gehört z. B. eine Abbildung aller Möglichkeiten der Buchfunktion in XML-Strukturen. Die Nummerierungseinstellungen für die einzelnen Kapitel innerhalb einer FrameMaker-Buchdatei lassen sich mithilfe von Lese-/Schreibregeln nicht in XML speichern, genauso fehlt ein Konzept zur Übernahme von Listen, die mithilfe der Buchfunktion generiert wurden.

Die Beschränkungen, die sich bei der Entwicklung ergeben haben, könnten evtl. durch den Einsatz eines sogenannten Structured-API-Clients aufgehoben werden. Diese Möglichkeit zu prüfen wäre ein weiterer Schritt hin zur Integration des Systems in größeren Zusammenhängen der Informationsverarbeitung innerhalb eines Gesamtsystems. Eine Datenbankanbindung ist dabei genauso denkbar, wie der Aufbau von Schnittstellen, um die entwickelte XML-Anwendung zu einem vollwertigen, in die informationellen Gesamtprozesse eines Unternehmens integrierten, XML-basierten Content-Management-System auszubauen.

Für die Realisierung solcher Systeme wären allerdings große Entwicklungsressourcen nötig. Bei Structured API Clients handelt es sich um in der Programmiersprache C geschriebene und kompilierte Zusatzprogramme, die beim XML-Export und -Import aktiv werden können.

In der bestehenden entwickelten Anwendung standen diese Ressourcen nicht zur Verfügung. Einige der fehlenden Funktionen ließen sich jedoch mit der FrameMaker-Version 7.2 realisieren, da bei dieser Version ein XSLT-Stylesheet beim Import oder Export verarbeitet werden kann.

Insgesamt bleibt abzuwarten, inwieweit die noch nicht voll ausgereiften Funktionen zum XML-Export und -Import in zukünftigen FrameMaker-Versionen weiter verbessert werden. Die Ankündigung einer vollständigen Unicode-Unterstützung in FrameMaker 8 lässt diese Verbesserungen jedoch erwarten. Schon heute sind mit einer XML-Anwendung für Adobe FrameMaker die Vorteile von XML in einem DTP-Programm nutzbar. Diese zukunftsweisende Lösung dürfte gerade für kleine Redaktionen interessant sein, bei denen der Nutzen eines XML-basierten CMS in keiner vernünftigen Relation zum Implementierungsaufwand und den dabei anfallenden Kosten stehen würde.

Literaturverzeichnis

Adobe Foreneintrag 2006a
Adobe Forums - How to create FrameMaker index and toc files from markup? `http://www.adobeforums.com/cgi-bin/webx?13@@.3bbf3e46`, Abruf: 03.03.2007

Adobe Foreneintrag 2006b
Adobe Forums - How do book files get converted to structure? `http://www.adobeforums.com/cgi-bin/webx/.3bc0b98b`, Abruf: 03.03.2007

Ambrus 2004
Ambrus, Thomas: Anwendungsfall „CMS“. Einführung eines CMS aus Sicht der Technischen Redaktion. In: *technische kommunikation* 26 (2004), Nr. 6, S. 32–37

Badenbrink und Ried 2001
Badenbrink, Stefan ; Ried, Tilo: Vom Informationsmanagement zum Wissensmanagement mit SGML/XML. In: Hennig, Jörg (Hrsg.): *Informations- und Wissensmanagement für technische Dokumentation.* 2001, S. 99–111

Bauer u. a. 1994
Bauer, Carl-Otto ; Hinsch, Christian ; Eidam, Gerd: *Produkthaftung. Herausforderung an Manager und Ingenieure.* Berlin : Springer-Verl., 1994. – ISBN 3–540–55519–6

Behme 2001
Behme, Henning: *Mutabor. XSLT-Tutorial.* Version: 2001. `http://www.heise.de/ix/artikel/2001/01/167/`, Abruf: 21.02.2007. – iX 1/2001, S. 167: XML-Programmierung

Behme und Mintert 2000
Behme, Henning ; Mintert, Stefan: *XML in der Praxis.* 2., erw. Aufl. München u.a. : Addison-Wesley, 2000. – ISBN 3–8273–1636–7. – professionelles Web-Publishing mit der Extensible Markup Language

Bergerhoff 2006
Bergerhoff, Thomas: XML-Editoren. Welcher Editor darf's denn sein? In: *Produkt Global* 2 (2006), Nr. 4, S. 29–31

Bienert 1998
Bienert, Peter: *Information und Kommunikation. Technik und Anwendung in Wirtschaft und Medien.* Berlin u.a. : Springer, 1998. – ISBN 3–540–64057–6. – Technik und Anwendung in Wirtschaft und Medien

Böhler 2001
Böhler, Klaus: Modulare Informationseinheiten nach Information Mapping® als Basis für effizientes Informationsmanagement. In: Hennig, Jörg (Hrsg.): *Informations- und Wissensmanagement für technische Dokumentation.* 2001, S. 126–139

Bohn 2006a
Bohn, Kai: DIN EN ISO 12100 in Kraft. In: *technische kommunikation* 28 (2006), Nr. 2, S. 55

Bohn 2006b
Bohn, Kai: Gefahren erkennen – Gefahren dokumentieren. Grundlagen und Durchführung der Gefahrenanalyse. In: *technische kommunikation* 28 (2006), Nr. 1, S. 25–29

Dentz 2001
Dentz, Dorothea: Informationsdesign für technische Dokumentation in einem Dokumentenmanagementsystem. In: Hennig, Jörg (Hrsg.): *Informations- und Wissensmanagement für technische Dokumentation*. 2001, S. 140–158

DITA und S1000D Application Packs 2006
FrameMaker 7.2 Application Packs for DITA and S1000D. Version: 2006. `http://labs.adobe.com/technologies/framemaker_ap/`, Abruf: 10.03.2007

Dokania 2007
Dokania, Aseem: *FrameMaker is and will remain to be*. Version: 2007. `http://blogs.adobe.com/techcomm/2007/02/framemaker_is_and_will_remain.html`, Abruf: 10.03.2007. – Product Manager für FrameMaker im Technical Communication Blog von Adobe

FrameMaker 8 – angebliche Features 2007
Presentations, Proclamations, & Predictions: FrameMaker 8, RoboHelp & Technical Communication Tools. `http://thecontentwrangler.com/article/presentations_proclamations_and_predictions_adobe_framemaker_robohelp_techn/`, Abruf: 10.03.2007

FrameMaker Anpassungen 2002
Customizing FrameMaker. Version: 2002. `http://www.adobe.com/devnet/framemaker/pdfs/Customizing_Frame_Products.pdf`, Abruf: 10.03.2007. – PDF-Datei ist bei installiertem FrameMaker 7.2 unter „FrameMaker7.2\handbch\“ zu finden (gilt auch für Trialversion)

FrameMaker Solutions Guide 2002
Adobe FrameMaker 7.0 Solutions Guide. Version: 2002. `http://partners.adobe.com/public/developer/en/framemaker/FM7_Solutions_Guide.pdf`, Abruf: 10.03.2007

Gabriel 2001
Gabriel, Carl-Heinz: Informationsbeschaffung und Verwaltung von Normen und Richtlinien. In: Hennig, Jörg (Hrsg.): *Informations- und Wissensmanagement für technische Dokumentation*. 2001, S. 171–187

Gabriel 2005
Gabriel, Carl-Heinz: Normenkompass. Normenarten und ihre Bedeutungen. In: *technische kommunikation* 27 (2005), Nr. 2, S. 30–35

Gabriel 2006
Gabriel, Carl-Heinz: Neuer ANSI-Standard für Technische Dokumentation. Gestaltung von Sicherheitshinweisen nach ANSI Z535.6. In: *technische kommunikation* 28 (2006), Nr. 4, S. 46–48

Galbierz 2004
GALBIERZ, Martin: Verbraucherrechte erneut gestärkt. Geräte- und Produktsicherheitsgesetz ab 1.5. in Kraft. In: *technische kommunikation* 26 (2004), Nr. 2, S. 42–47

Galbierz 2005
GALBIERZ, Martin: GPSG fordert mehr Kontrolle. Produktrückrufe wegen falscher Instruktion. In: *technische kommunikation* 27 (2005), Nr. 5, S. 47–51

Gust 2005
GUST, Dieter: Praxistipps Adobe FrameMaker: Grafikimport – Teil 2. In: *technische kommunikation* 27 (2005), Nr. 6, S. 15–16

Gust 2006
GUST, Dieter: Praxistipps Adobe FrameMaker: Neue Rechtschreibprüfung. In: *technische kommunikation* 28 (2006), Nr. 6, S. 38–40

Gust 2007
GUST, Dieter: Praxistipps Word: Microsoft forciert Grußkartendesign. In: *technische kommunikation* 29 (2007), Nr. 1, S. 42–44

Gust und Plattner 2003
GUST, Dieter ; PLATTNER, Michael: Stiefkind ohne richtiges Zuhause? Auf der Suche nach neuem Selbstbewusstsein zwischen Marketing und Entwicklung. In: *technische kommunikation* 25 (2003), Nr. 6, S. 29–33

Hellfritsch 2006
HELLFRITSCH, Edgar: Von Schreibern und Rahmenmachern. Vergleich OpenOffice Writer und Adobe FrameMaker. In: *technische kommunikation* 28 (2006), Nr. 6, S. 29–35

Hennig 2001
HENNIG, Jörg (Hrsg.): *Tekom-Schriften zur technischen Kommunikation*. Bd. 4: *Informations- und Wissensmanagement für technische Dokumentation*. Lübeck : Schmidt-Römhild, 2001. – ISBN 3–7950–0774–7

Hentschel 2006
HENTSCHEL, Uwe: Tschüss Offset! Digitaldruck – Perspektiven jenseits der Technik. In: *technische kommunikation* 28 (2006), Nr. 6, S. 17–20

Heuer 2001
HEUER, Jens U.: Rechtliche Aspekte von Informations- und Wissensmanagement. In: HENNIG, Jörg (Hrsg.): *Informations- und Wissensmanagement für technische Dokumentation*. 2001, S. 33–49

Holzmann 2004
HOLZMANN, Martin: DMS/CMS – ein Systemüberblick. Systeme, Kategorien und Eigenschaften. In: *technische kommunikation* 26 (2004), Nr. 6, S. 24–30

Hurst 2006
HURST, Sophie: XML im Wartestand. Umfrage zu Tools und Prozessen in Technischen Redaktionen. In: *Produkt Global* 2 (2006), Nr. 3, S. 10

Juhl 2002
Juhl, Dietrich: *Technische Dokumentation. Praktische Anleitungen und Beispiele.* Berlin : Springer, 2002 (Engineering online library). http://www.gbv.de/du/services/agi/8D8900559423A3FFC1256D2E004EFCDE/420000088122. – ISBN 3–540–43127–6

Ley 2006
Ley, Martin: Aspekte der Informationsstrukturierung. Über Strukturierungsprinzipien, die Ebenen der Textstruktur und Dokumentgrammatiken. In: *technische kommunikation* 28 (2006), Nr. 4, S. 51–53

Lüthy und Wetzl 2001
Lüthy, Norbert ; Wetzl, Regina: Technische Dokumentation und Dokumenten-Management-Systeme – Ein Erfahrungsbericht. In: Hennig, Jörg (Hrsg.): *Informations- und Wissensmanagement für technische Dokumentation.* 2001, S. 66–79

Mallok 2006
Mallok, Henning: Die XML-Falle. Redaktionssysteme in mittelständischen Unternehmen. In: *Produkt Global* 2 (2006), Nr. 3, S. 36–37

Massion 2006
Massion, François: Intelligente Helfer. Translation-Memory-Systeme im Vergleich. In: *Produkt Global* 2 (2006), Nr. 3, S. 30–33

Mitschke und Schulze 2003
Mitschke, Ute ; Schulze, Bernhard: *FrameMaker 6 + 7. in der Praxis anwenden und beherrschen.* München u.a. : Addison-Wesley, 2003 (dpi). – ISBN 3–8273–1736–3

OASIS 1995
OASIS (Hrsg.): *CALS table model Document Type Definition.* Version: 1995. http://www.oasis-open.org/specs/tm9502.html, Abruf: 21.02.2007

OASIS 1999
OASIS (Hrsg.): *XML Exchange Table Model Document Type Definition.* Version: 1999. http://www.oasis-open.org/specs/tm9901.html, Abruf: 21.02.2007

Oehmig 2006
Oehmig, Peter: Gute Güte! Textqualität ermitteln. tekom AG entwickelt Kriterienkatalog. In: *technische kommunikation* 28 (2006), Nr. 4, S. 40–43

Publishing Smarter 2006
Publishing Smarter (Hrsg.): *FrameDITA and FrameDITA-lite.* Version: 2006. http://www.publishingsmarter.com/pages/develop/framemaker_DITA.html, Abruf: 03.03.2007

Rath 2006
Rath, Hans H.: Standards mit Zukunft. XML & Co. – Stand und Perspektive. In: *technische kommunikation* 28 (2006), Nr. 2, S. 19–23

Reuther 2006
Reuther, Ursula: Gut für Struktur und Inhalt. Kontrollierte Sprache in strukturierten Dokumenten. In: *technische kommunikation* 28 (2006), Nr. 5, S. 53–55

Ried 2003
RIED, Tilo: Kreuz und Quer durchs Unternehmen. Analyse und Optimierung von Informationsprozessen. In: *technische kommunikation* 25 (2003), Nr. 2, S. 42–45

Rögner 2003
RÖGNER, Andrea: *Der Weg zur „haftungssicheren" Dokumentation.* Abtsgmünd : Schulz, 2003. – ISBN 3–930055–07–4

Romberg 2000
ROMBERG, Markus: *Konzept und Entwicklung eines Redaktionssystems zur Produktion von Anwenderdokumentationen.* 2000 (Fortschritt-Berichte pak ; 1). – ISBN 3–925178–36–8. – anwendergerechte Online- und Papierdokumentation aus einer Datenquelle

Schlenker 2006
SCHLENKER, Reiner: Rettung vor der Modulflut. Modularisierung Technischer Dokumentation. In: *technische kommunikation* 28 (2006), Nr. 1, S. 39–45

Schmidt 2006
SCHMIDT, Axel: Unternehmensgröße ist sekundär. Content-Management-Systeme. In: *Produkt Global* 2 (2006), Nr. 3, S. 38–39

Schmidt 2007a
SCHMIDT, Axel: Übersetzungsgerechtes Schreiben. Beim Quelltext ansetzen. In: *Produkt Global* 3 (2007), Nr. 1, S. 24–25

Schmidt 2007b
SCHMIDT, Axel: Word in der Technischen Dokumentation – noch zeitgemäß? „Geniale Katastrophe". In: *Produkt Global* 3 (2007), Nr. 1, S. 28–29

Schmitz 2001
SCHMITZ, Klaus-Dirk: Terminologieverwaltung. In: HENNIG, Jörg (Hrsg.): *Informations- und Wissensmanagement für technische Dokumentation.* 2001, S. 188–202

Straub 2006
STRAUB, Daniela: Technische Redakteure immer qualifizierter. Neue tekom-Studie über Bildung und Arbeitsmarkt. In: *technische kommunikation* 28 (2006), Nr. 6, S. 49–53

Straub und Ziegler 2007
STRAUB, Daniela ; ZIEGLER, Wolfgang: Ein Blick in die Werkzeugkiste. Ergebnisse der Softwareumfrage 2006. In: *technische kommunikation* 29 (2007), Nr. 2, S. 42–44

Struct. App. Dev. Guide 2005
Adobe FrameMaker 7.2 Structure Application Developer's Guide Online Manual. Version: 2005. `http://www.adobe.com/devnet/framemaker/pdfs/Structure_Dev_Guide.pdf`, Abruf: 10.03.2007. – PDF-Datei ist bei installiertem FrameMaker 7.2 unter „FrameMaker7.2\handbch\" zu finden (gilt auch für Trialversion)

Tidwell und Lichtenberg 2003
TIDWELL, Doug ; LICHTENBERG, Kathrin: *XSLT. XML-Dokumente transformieren.* Beijing : O'Reilly, 2003. – ISBN 3–89721–292–7

Wikipedia 2007a
WIKIPEDIA (Hrsg.): *Desktop Publishing – Wikipedia*. Version: 2007. `http://de.wikipedia.org/wiki/Desktop_Publishing`, Abruf: 03.03.2007

Wikipedia 2007b
WIKIPEDIA (Hrsg.): *FrameMaker – Wikipedia*. Version: 2007. `http://de.wikipedia.org/wiki/FrameMaker`, Abruf: 21.02.2007

Wikipedia 2007c
WIKIPEDIA (Hrsg.): *WYSIWYG – Wikipedia*. Version: 2007. `http://de.wikipedia.org/wiki/WYSIWYG`, Abruf: 21.02.2007

Ziegler 2004
ZIEGLER, Wolfgang: Lohnt sich Content Management? Aspekte zum Einsatz von Redaktionssystemen. In: *technische kommunikation* 26 (2004), Nr. 6, S. 19–23

Abbildungsverzeichnis

Abkürzungsverzeichnis

ANSI	American National Standards Institute
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CALS	Computer-Aided Acquisition and Life-Cycle Support
CAT	Computer Aided Translation
CEN	Europäisches Komitee für Normung (französisch: Comité Européen de Normalisation)
CENELEC	Europäisches Komitee für elektrotechnische Normung (französisch: Comité Européen de Normalisation Electrotechnique)
CGM	Computer Graphics Metafile
CMS	Content-Management-System
DIN	Deutsches Institut für Normung
DITA	Darwin Information Typing Architecture
DLL	Dynamic Link Library
DMS	Document-Management-System
DPML	Documentation Publisher's Markup Language
DTD	Document Type Definition
DTP	Desktop-Publishing
ECM	Enterprise-Content-Management
ERP	Enterprise-Resource-Planning
EDD	Element Definition Document
EDV	Elektronische Datenverarbeitung
ETSI	Europäisches Institut für Telekommunikationsnormen (englisch: European Telecommunications Standards Institute)
EPS	Encapsulated Postscript
FDK	Frame Developer's Kit
GPSG	Geräte- und Produktsicherheitsgesetz
HTML	Hypertext Markup Language
ISO	Internationale Organisation für Normung
MIF	Maker Interchange Format

MML	Maker Markup Format
OASIS	Organization for the Advancement of Structured Information Standards
SGML	Standard Generalized Markup Language
SVG	Scalable Vector Graphics
TM-System	Translation-Memory-System
W3C	World Wide Web Consortium
WMF	Windows Metafile
WYSIWYG	What You See Is What You Get
PDF	Portable Document Format
RTF	Rich Text Format
TIFF	Tagged Image File Format
XML	Extensible Markup Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations
XSL-FO	Extensible Stylesheet Language – Formatting Objects

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich die Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, habe ich als solche kenntlich gemacht. Ich weiß, dass bei Abgabe einer falschen Versicherung die Prüfung als nicht bestanden zu gelten hat.

Anhang

Im nun folgenden Anhang wird in Teil A die Bedienungsanleitung für die selbst erstellte XML-Anwendung – DPML genannt – wiedergegeben. Diese Anleitung richtet sich an Technische Redakteure, die mit DPML arbeiten und schon Erfahrung im Umgang mit dem Programm Adobe FrameMaker haben. Sie wird als eine Ergänzung zu zusätzlichen Schulungen verstanden.

In Teil B des Anhangs werden Auszüge aus der DTD, die zu der entwickelten XML-Anwendung gehört, wiedergegeben.

dpML
documentation publisher's
Markup Language

A Bedienung der XML-Anwendung

Kapitelüberblick

Zweck In diesem Kapitel erhalten Sie grundlegende Hinweise zur Bedienung der dpml-Anwendung für Adobe FrameMaker.

Wenden Sie sich bei weiteren Fragen an in dpml geschulte Redakteure. Gibt es in Ihrem Team einen FrameMaker Application-Administrator oder Template-Designer, der eigene Anpassungen an dpml vornimmt, dann lassen Sie sich eventuelle Änderungen der Funktionalität erklären.

Informieren Sie sich, ob Sie mit einer für Ihren Zweck angepassten dpml-Anwendung arbeiten und ob dafür eine spezialisierte Dokumentation zur Verfügung steht.

Inhalt Dieses Kapitel enthält die folgenden Abschnitte:

A.1 Der Publikationsworkflow mit dpML

Überblick

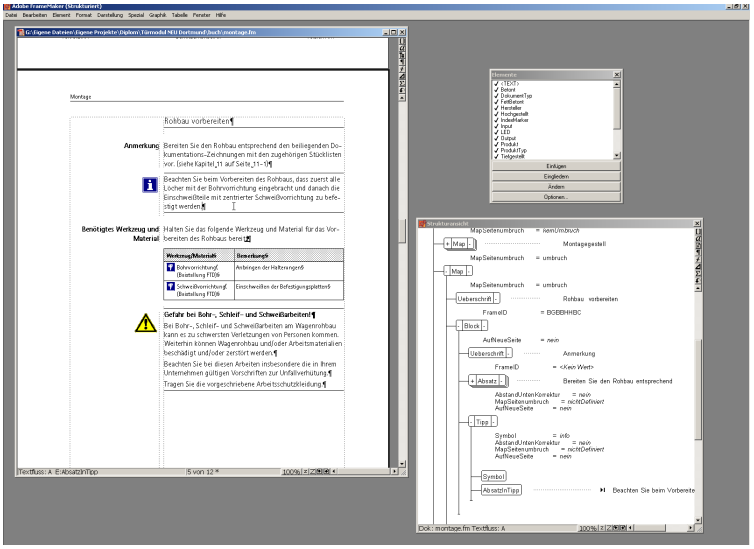
Zweck In diesem Abschnitt finden Sie eine Übersicht über die fünf Phasen des Publikationsworkflow mit der dpML-Anwendung für Adobe FrameMaker.

Inhalt Dieser Abschnitt enthält die folgenden Informationen:

Inhalte kapitelweise erstellen

Erste Phase In der ersten Phase des Publikationsworkflows mit dpml erstellen Sie kapitelweise die Inhalte der Dokumentation.

- Durch den Einsatz der strukturierten Oberfläche von Adobe FrameMaker mit den dpml-Templates vereinfacht sich die Bedienung des Programms.
- Die „sprechenden" Elementnamen und der logische Aufbau der dpml-Struktur unterstützen Sie beim Erfassen der Inhalte in konsistenter Form – auch bei komplexen Dokumentationsprojekten.



- Sie arbeiten gleichzeitig im Dokumentfenster und in der FrameMaker-Strukturansicht. So können Sie das endgültige Printlayout Ihrer Dokumentation genauso kontrollieren, wie den richtigen Aufbau der dpml-Struktur.
- Die dpml-Struktur ist so aufgebaut, dass Sie leicht Inhalte aus bestehenden FrameMaker-Dokumentationen übernehmen können.

Weitere Informationen zum Erstellen strukturierter Dokumente mit der dpml-Anwendung finden Sie im Abschnitt „Strukturierte Dokumente erstellen" auf Seite A-8.

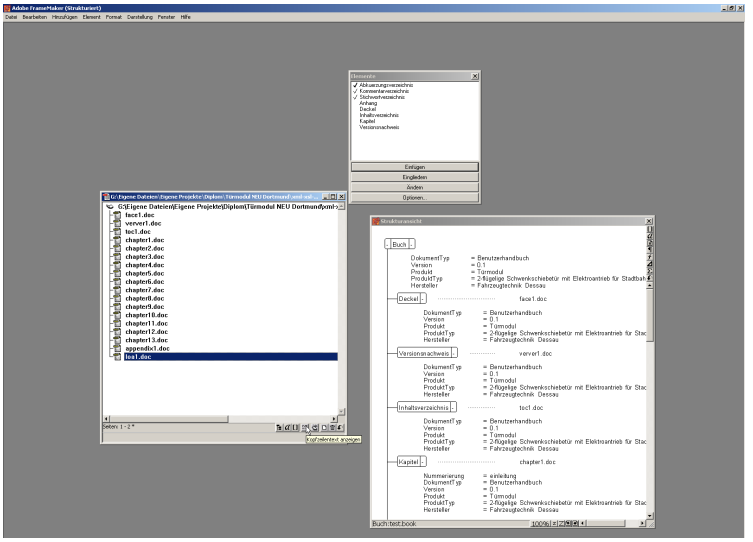
Beachten Sie ebenfalls den Abschnitt „Grafik einbinden" auf Seite A-25.

Falls Sie Informationen aus unstrukturierten FrameMaker-Dokumenten in dpml übernehmen wollen, lesen Sie den Abschnitt „Unstrukturierte Dokumente importieren" auf Seite A-29.

Buchdatei zusammenstellen

Zweite Phase In der zweiten Phase des Publikationsworkflows mit dpml stellen Sie die gesamte Dokumentation als Buchdatei zusammen.

- Das dpml-Buch ist eine strukturierte FrameMaker-Buchdatei.
- Im dpml-Buch fügen Sie die mit dpml strukturierten Kapiteldateien zu einer Gesamtdokumentation zusammen.



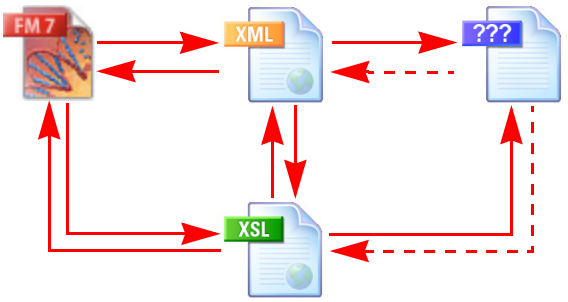
- Im dpml-Buch können Sie die Zusammenstellung der Kapitel Ihrer Dokumentation leicht anpassen: entfernen Sie nicht mehr benötigte Kapitel, fügen Sie neue Kapitel hinzu und verändern Sie ihre Reihenfolge per „Drag and Drop".
- Im dpml-Buch verwalten Sie zentral die Versionsnummern Ihrer Buchdatei und der enthaltenen Kapiteldateien, sowie die weiteren über Attribute gesteuerten Metainformationen Ihrer Dokumentation.

Weitere Informationen zum Zusammenstellen der mit der dpml-Anwendung strukturierten Dokumente zum dpml-Buch finden Sie im Abschnitt „Strukturierte Dokumente publizieren" ab Seite A-35.

XML-Daten generieren

Dritte Phase In der dritten Phase des Publikationsworkflows mit dpml generieren Sie aus der Buchdatei, die alle Inhalte der Dokumentation enthält, XML-Daten.

- Durch XML erhalten Sie zukunftssichere Daten, die in einem offenen, von zukünftig eingesetzten Programmen oder Programmversionen unabhängigen Format vorliegen.



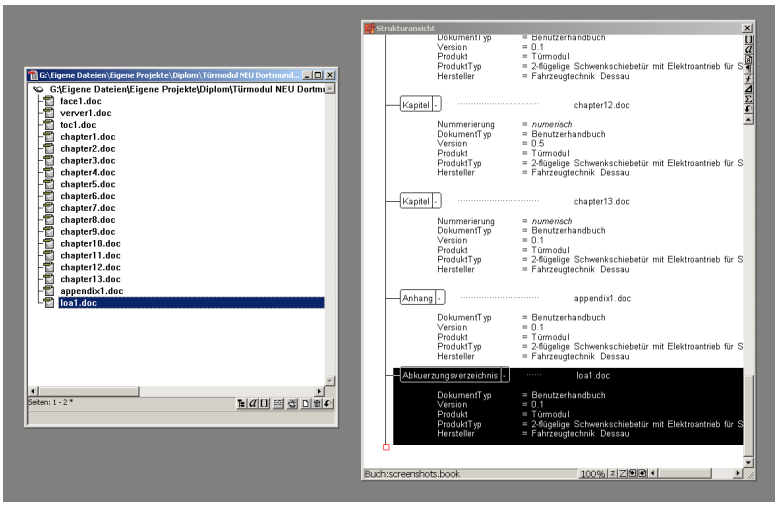
- Vor dem Speichern in XML überprüfen Sie die Struktur der Dokumente mithilfe der entsprechenden Funktion im FrameMaker, um einen problemlosen XML-Export sicherzustellen.
- Die Gesamtdokumentation wird aus der Buchdatei in XML-Daten exportiert.
- Beim XML-Export werden alle in den FrameMaker-Dokumenten enthaltenen Grafiken als einzelne Dateien im CGM-Format mitgespeichert.
- Beim XML-Export wird ein XSLT-Stylesheet angewendet, mit dem das Inhaltsverzeichnis, die Versionsnachweisliste, das bei Korrekturversionen mögliche Kommentarverzeichnis und die Inhaltsüberblicke am Anfang von Kapiteln und Abschnitten automatisch generiert werden.

Weitere Informationen zum Generieren von XML-Daten aus mit der dpml-Anwendung strukturierten Dokumentationen finden Sie im Abschnitt „Strukturierte Dokumente publizieren" ab Seite A-39.

Generierte XML-Daten zum Publizieren öffnen

Vierte Phase In der vierten Phase des Publikationsworkflows mit dpml werden die XML-Daten wieder in FrameMaker importiert. Danach liegen die FrameMaker-Daten inklusive aller beim XML-Export generierten Listen vor.

- Beim Wiederimport der XML-Daten wird automatisch ein neues FrameMaker-Buch angelegt, dass die Gesamtdokumentation mit allen Kapiteln und den beim XML-Export generierten Verzeichnissen enthält.



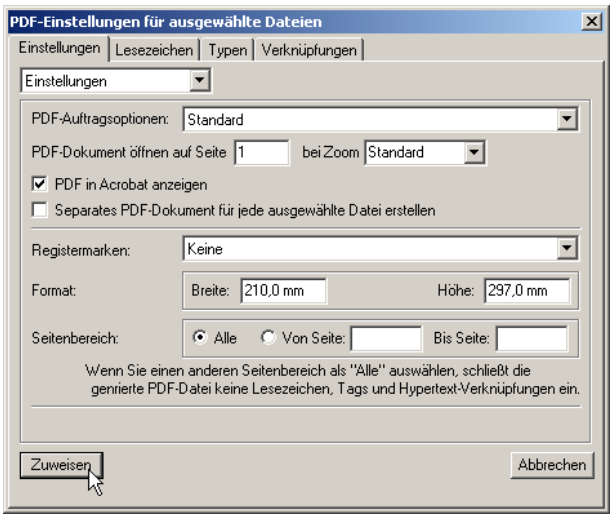
- In den aus den XML-Daten angelegten FrameMaker-Dokumenten sind die beim XML-Export generierten CGM-Grafiken eingebettet.
- Die durch das XSLT-Stylesheet beim XML-Export generierten Verzeichnisse werden als eine Liste strukturierter Querverweise in die FrameMaker-Dokumente übernommen.
- Beim XML-Import werden die Daten komplett neu eingelesen. Dadurch werden alle Querverweise aktualisiert.
- Beim Öffnen der mit der dpml-Anwendung erstellten XML-Daten zum Publizieren testen Sie diese zusätzlich auf Praxistauglichkeit und Strukturiertheit.

Weitere Informationen zum Öffnen von mithilfe von dpml generierten XML-Daten finden Sie im Abschnitt „Strukturierte Dokumente publizieren“ ab Seite A-42.

Printversion publizieren

Fünfte Phase In der fünften Phase des Publikationsworkflows mit dpml stellen Sie die Printversion der Dokumentation fertig und publizieren das Ergebnis als PDF.

- Nach dem XML-Wiederimport stellen Sie die Nummerierung der Absätze und Seiten in den Kapiteln der Dokumentation in der FrameMaker-Buchdatei richtig ein.
- Vor der PDF-Erstellung fügen Sie den Standardindex als unstrukturierte Liste mit der Buchfunktion hinzu, falls Sie diesen in Ihrer Dokumentation benötigen.
- Sind alle Bestandteile der Dokumentation komplett, publizieren Sie das Ergebnis als PDF.



- Das PDF-Dokument eignet sich sowohl zum Ausdrucken, als auch als Onlinedokument. Durch die entsprechenden Möglichkeiten beim Export enthält es Hypertextfunktionen und PDF-Lesezeichen.

Weitere Informationen zum Fertigstellen der Printversion und zum Publizieren als PDF finden Sie im Abschnitt „Strukturierte Dokumente publizieren" ab Seite A-43.

A.2 Strukturierte Dokumente erstellen

Überblick

Zweck In diesem Abschnitt wird erklärt, wie Inhalte mit der dpml-Anwendung für Adobe FrameMaker strukturiert erfasst und formatiert werden.

Inhalt Dieser Abschnitt enthält die folgenden Informationen:

FrameMaker strukturiert

Wenn Sie mit der unstrukturierten Variante von FrameMaker arbeiten, stehen Ihnen einige Dialoge und Menübefehle nicht zur Verfügung, die Sie zum Arbeiten mit dpml benötigen.

Einstellung prüfen

Prüfen Sie, dass Sie FrameMaker in der strukturierten Variante verwenden. Starten Sie dazu das Programm. Wenn Sie die strukturierte Variante verwenden, steht in der Titelleiste „Adobe FrameMaker (Strukturiert)".

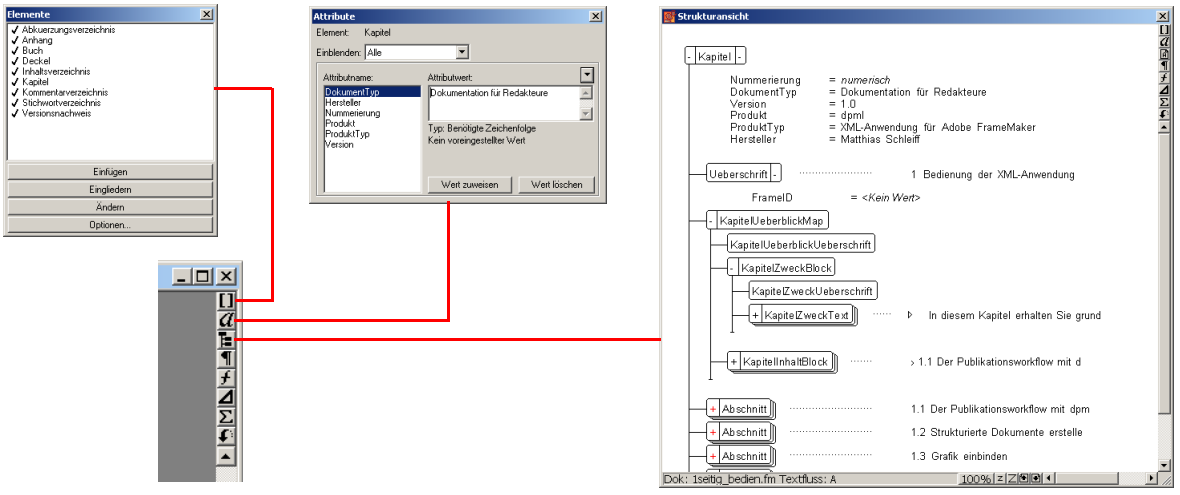
Oberfläche umschalten

Führen Sie die folgenden Schritte durch, um die FrameMaker-Oberfläche von unstrukturiert auf strukturiert umzuschalten:

1. Wählen Sie im Menü
   „Datei → Voreinstellungen → Allgemein..."
   ⇒ Der Dialog „Voreinstellungen" öffnet sich.
2. Wählen Sie im Dialog „Voreinstellungen" aus dem Auswahlfeld neben dem Punkt „Produktoberfläche" ganz unten die Einstellung „Strukturierter FrameMaker".
3. Drücken Sie die Schaltfläche „Zuweisen" im Dialog „Voreinstellungen".
   ⇒ Die Einstellung ist bestätigt und ab sofort wirksam.

Elementkatalog und Strukturansicht

Sämtliche aus der unstrukturierten Oberfläche bekannten Dialoge sind auch im strukturierten FrameMaker verfügbar. Zum Bearbeiten von dpml-Dokumenten benutzen Sie aber an Stelle von Absatz-, Zeichenkatalog, usw. fast ausschließlich die zusätzlich zur Verfügung stehenden Dialoge „Elementkatalog", „Attributemanager" und „Strukturansicht".



Dokumente mit dpml strukturieren

Um mit dpml strukturierte Dokumente zu erstellen, brauchen Sie als Dokumentvorlage eine leere Datei. Vorlagedateien werden auch als Templates bezeichnet.

Template verwenden

Verwenden Sie das zu ihrer dpml-Anwendung gehörende Template, wenn Sie ein neues Dokument anlegen.

1. Wählen Sie im Menü
   „Datei → Neu → Dokument..."
   ⇒ Der Dialog „Neu" öffnet sich.
2. Wählen Sie im Dialog „Neu" über das Dropdown-Menü neben „Suchen in:" und dem darunter liegenden Dateiauswahlfeld den Speicherort des dpml-Templates.

   Der Pfad des Templates wird Ihnen vom Systemadministrator mitgeteilt.
3. Klicken Sie auf die Schaltfläche „Neu", um Ihre Auswahl zu bestätigen.
   ⇒ Eine leere, auf dem Template basierte Datei öffnet sich.
4. Speichern Sie die Datei, um den Namen und Speicherort des Dokuments festzulegen.

Strukturiert arbeiten

Schalten Sie nach dem Anlegen des leeren Dokuments die zur strukturierten FrameMaker-Oberfläche gehörenden Dialoge „Elementkatalog" und „Strukturansicht" ein.

Informationen zum Einschalten der strukturierten FrameMaker-Oberfläche finden Sie auf der vorigen Seite.

Der Elementkatalog

Mit dem Elementkatalog werden Elemente in das Dokument eingefügt. Im Elementkatalog werden die Elemente angezeigt, die an der Stelle, an der der Cursor im Dokument oder in der Strukturansicht steht, gültig sind.

- Mit der Schaltfläche „Einfügen" wird das markierte Element aus dem Elementkatalog an der Cursorposition im Dokument platziert.
- Mit der Schaltfläche „Eingliedern" wird das momentan in der Strukturansicht markierte Element von einem anderen Element aus dem Elementkatalog „umschlossen".

Fortsetzung nächste Seite ...

... Fortsetzung Dokumente mit DPML strukturieren

– Mit der Schaltfläche „Ändern" wird das momentan in der Strukturansicht markierte Element durch ein anderes Element aus dem Elementkatalog ersetzt.

Die Strukturansicht

Mit der Strukturansicht navigieren Sie in strukturierten dpml-Dokumenten. Elemente können bequem markiert, verschoben, kopiert und eingefügt werden.

Die Strukturansicht visualisiert die Dokumentstrukturen als Baumansicht.

– Klicken Sie in der Baumstruktur auf den Namen eines Elements, um es zu markieren.

Markierte Elemente können Sie über den Elementkatalog ändern oder in andere Elemente eingliedern, außerdem können Sie es kopieren und an anderer Stelle im Dokument einfügen.

– Elemente können leicht in der Baumansicht verschoben werden, indem Sie:
  - zuerst den Namen eines Elements in der Strukturansicht anklicken und die linke Maustaste gedrückt halten
  - und dann das Element an die gewünschte Stelle in der Strukturansicht bewegen und die Maustaste loslassen.

– Klicken Sie auf das Minus- oder Plus-Symbol vor Elementen, um die darunterliegenden Äste in der Baumansicht ein und auszublenden.

Dadurch bewahren Sie die Übersicht bei der Bearbeitung komplexer Dokumente. Auch größere Abschnitte in Dokumenten können leicht verschoben oder kopiert werden, wenn ihre Unterelemente ausgeblendet sind.

Rote Farbe in der Strukturansicht weist Sie auf Fehler in der Dokumentstruktur hin. In diesem Fall sind Elemente im Dokument ungültig oder unvollständig.

Weiterführende Informationen zur Bearbeitung strukturierter Dokumente finden Sie auch in der FrameMaker-Hilfe.

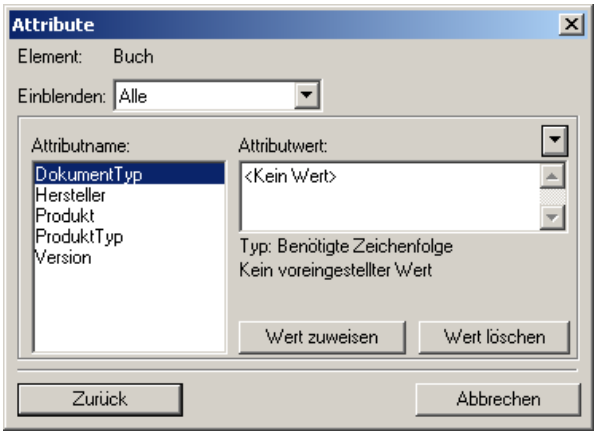
Zum Aufrufen wählen Sie im Programm-Menü „Hilfe → Hilfethemen" und suchen im Inhaltsregister der sich öffnenden Hilfe das Thema „Elemente in strukturierten Dokumenten".

Fortsetzung nächste Seite ...

... Fortsetzung Dokumente mit DPML strukturieren

Attributemanager für benötigte Attribute

Wenn Sie in ein leeres dpml-Dokument ein Kapitel-Element einfügen, öffnet sich automatisch ein Attributemanager und fordert sie damit zur Angabe der Attribute „Produkt", „Produkt-Typ", „Hersteller", „DokumentTyp" und „Version" auf. Diese Attribute des Elements Kapitel werden in der Struktur benötigt, d.h. ihre Angabe ist nicht optional, sondern zwingend erforderlich.



Um Attributen mit dem Attributemanager Werte zuzuweisen gehen Sie für jeden Attributwert in folgenden Schritten vor:

1. Wählen Sie ein Attribut des aktiven Elements, dessen Wert Sie angeben wollen, unter „Attributname" aus.
2. Geben Sie den Wert des Attributs unter „Attributwert" an.
3. Klicken Sie auf die Schaltfläche „Wert zuweisen", um die Einstellung für dieses Attribut zu bestätigen.

Es wird immer nur der Wert des gerade unter „Attributname" aktivierten Attributs zugewiesen. Mehrfachzuweisungen sind nicht möglich.

Damit Sie das Dokument nach dem Zuweisen aller benötigter Attributwerte weiter bearbeiten können, müssen Sie den Attributemanager für die benötigten Attributwerte schließen.

Fortsetzung nächste Seite ...

... Fortsetzung Dokumente mit DPML strukturieren

Attributemanager für optionale Attribute

Um Attribute anzugeben, die nicht benötigt werden, sondern optional sind, können Sie ebenfalls den Attributemanager verwenden. Um den Attributemanager für die Angabe optionaler Attribute zu öffnen gibt es zwei Möglichkeiten:

- Klicken Sie zum Öffnen das Symbol mit dem kleinen „a" in der oberen rechten Ecke des Dokumentfensters. Mit dem so aufgerufenen Attributemanager bearbeiten Sie das Element, bei dem sich gerade der Cursor befindet oder das Sie in der Strukturansicht markiert haben.

  Der so geöffnete Attributemanager muss nach dem Zuweisen von Attributen nicht geschlossen werden, sondern kann auf der Arbeitsfläche bei der Bearbeitung des Dokuments geöffnet bleiben.
  Benötigte Attribute werden in der Voreinstellung des Programms trotzdem in einem extra Attributemanager angefordert.

- In der Strukturansicht werden auch die Attributwerte der Elemente angezeigt. Öffnen Sie einen Attributemanager für ein Element, indem Sie auf eines seiner Attribute in der Strukturansicht doppelt klicken
  ⇒ Ein Attributemanager für das Element öffnet sich

  Einen so geöffneten Attributemanager müssen Sie nach dem Zuweisen der Attributwerte wie den Attributemanager für benötigte Attribute schließen, damit Sie das Dokument weiter bearbeiten können.

Die Arbeitsoberfläche anpassen

In der Strukturansicht zoomen

Wie im Dokumentfenster können Sie auch in der Strukturansicht des Dokuments verschiedene Zoomfaktoren einstellen, indem Sie auf die entsprechenden Schaltflächen am unteren Rand des „Strukturansicht"-Fensters klicken.

Elementkatalog-Optionen

Im Elementkatalog können die Elemente ausgewählt werden, die an der Stelle, an der sich der Cursor im Dokument befindet, gültig sind. Dieses Verhalten des Elementkatalogs kann verändert werden. Gehen Sie dazu in folgenden Schritten vor:

> ⚠ **Die Abbildung der Dokumentstrukturen in XML-Daten erfordert die genaue Einhaltung der definierten Struktur-Regeln. Die Voreinstellung für den Elementkatalog sollte nicht verändert werden. Verwenden Sie diese Anleitung, um die Elementkatalog-Optionen zurückzusetzen.**

1. Klicken Sie im Elementkatalog auf die Schaltfläche „Optionen..."
   ⇒ Der Dialog „Verfügbare Elemente zuweisen" öffnet sich.
2. Wählen Sie unter „Tags einblenden für:" die Voreinstellung „Alle zugelassenen nachgeordneten Elemente" und entfernen Sie den Haken in der Checkbox „Nach anderen gültigen Elementen auflisten" unter „Einschlüsse", falls dieser gesetzt ist.
3. Klicken Sie auf die Schaltfläche „Zuweisen", um die Einstellungen zu bestätigen.
   ⇒ Die Einstellungen werden übernommen und sind ab sofort wirksam.

Attributanforderung aus- oder einschalten

In der Voreinstellung des Programms werden alle benötigten Attributwerte angefordert, indem beim Einfügen der entsprechenden Elemente automatisch ein Attributemanager geöffnet wird. Wenn Sie lieber alle Attributwerte mithilfe der Strukturansicht oder dem über das entsprechende Symbol geöffneten Attributemanager angeben wollen, können Sie dieses Verhalten ändern.

Fortsetzung nächste Seite ...

... Fortsetzung Die Arbeitsoberfläche anpassen

Führen Sie die folgenden Schritte durch, um die Optionen für die automatische Attributanforderung zu ändern:

1. Wählen Sie aus dem Programm-Menü „Element → Neue Elementoptionen..."
   ⇒ Der Dialog „Neue Elementoptionen" öffnet sich.
2. Wählen Sie unter „Beim Einfügen des Elements" die gewünschte Einstellung:
   - Bei der Einstellung „Attributwerte immer anfordern" wird bei jedem Einfügen eines Elements, das Attribute besitzt, ein Attributemanager geöffnet. Dabei wird nicht zwischen optionalen und benötigten Attributen unterschieden.

     Da in dpml-Elementen viele optionale Attribute verwendet werden, deren Voreinstellungen selten verändert werden müssen, ist diese Einstellung für dpml ungeeignet.
   - Bei der Einstellung „Benötigte Attributwerte anfordern" wird beim Erstellen eines neuen Elements ein Attributemanager nur dann geöffnet, wenn das Element Attribute besitzt, die für eine gültige Struktur nötig sind.

     Dies ist die Voreinstellung. Da die wenigen Attributwerte, die in dpml angegeben werden müssen, für die Funktion der Anwendung unverzichtbar und eine korrekte Zuordnung von Dateien zu Dokumentationen hilfreich sind, wird diese Einstellung für dpml empfohlen.
   - Bei der Einstellung „Benötigte Attributwerte nicht anfordern" wird beim Erstellen eines Elements kein Attributemanager geöffnet – egal, ob das Element benötigte oder optionale Elemente besitzt.

     Dies ist eine geeignete Einstellung für Redakteure, die Erfahrung im Umgang mit dpml haben und lieber mit Attributemanagern arbeiten, die sie selbst öffnen, als mit solchen, die sich von selbst öffnen.
3. Lassen Sie den Haken in der Checkbox für „Automatisches Einfügen für untergeordnete Elemente zulassen" gesetzt und bestätigen Sie Ihre Einstellungen, indem Sie auf die Schaltfläche „Zuweisen" klicken.
   ⇒ Die Einstellungen werden übernommen und sind ab sofort wirksam.

Kapitel-Attribute angeben

Benötigte Attribute

Wenn Sie ein Kapitelelement in ein leeres dpml-Dokument einfügen, wird in der Voreinstellung sofort automatisch ein Attributemanager geöffnet, weil das Element Kapitel benötigte Attribute enthält, die Sie angeben müssen.

Weitere Informationen zur Arbeit mit Attributmanagern erhalten Sie unter „Attributemanager für benötigte Attribute" auf Seite A-12.

Weitere Informationen zur Veränderung der Voreinstellung für das automatische Öffnen von Attributmanagern erhalten Sie unter „Attributanforderung aus- oder einschalten" auf Seite A-14.

Überblick über die Attribute

Die folgende Liste enthält eine Übersicht, über die Attribute, die Sie für die höchsten gültigen Elemente in der FrameMaker-Struktur (z.B. Buch-, Kapitel- und Anhang-Element) unbedingt angeben müssen, sowie ihre Funktion im Dokument:

- Produkt
  - der Name des dokumentierten Produkts
  - taucht auf der Titelseite und in der Fußzeile des Dokuments auf (Attributwert in dieser Anleitung: „DPML")
  - bestimmt den Textinhalt des Elements „Produkt", das im Dokument wie eine Variable einsetzbar ist
- ProduktTyp
  - nähere Spezifikation des Produkts
  - taucht auf der Titelseite des Dokuments auf (Attributwert in dieser Anleitung: „XML-Anwendung für Adobe FrameMaker")
  - bestimmt den Textinhalt des Elements „ProduktTyp", das im Dokument wie eine Variable einsetzbar ist
- Hersteller
  - der Name des Herstellers des Produkts
  - taucht auf der Titelseite und in der Fußzeile des Dokuments auf (Attributwert in dieser Anleitung: „DPML-Team Halle")
  - bestimmt den Textinhalt des Elements „Hersteller", das im Dokument wie eine Variable einsetzbar ist

Fortsetzung nächste Seite ...

... Fortsetzung Kapitel-Attribute angeben

- DokumentTyp
  - nähere Spezifikation des Dokuments
  - taucht auf der Titelseite und in der Fußzeile des Dokuments auf (Attributwert in dieser Anleitung: „Dokumentation für Redakteure")
  - bestimmt den Textinhalt des Elements „DokumentTyp", das im Dokument wie eine Variable einsetzbar ist
- Version
  - Versionsnummer des Dokuments als Dezimalzahl
  - taucht auf der Titelseite, in der Versionsnachweistabelle (falls vorhanden) und in der Fußzeile des Dokuments auf (Attributwert in dieser Anleitung: „1.0")

    Mit Versionsnummern kann z. B. verwaltet werden, wann ein Dokument noch im Entwurfs-/Korrekturstadium und wann es zur Veröffentlichung freigegeben ist.

    Um die Versionsnummer aller Dokumente innerhalb eines dpml-Buchs auf einmal zu erhöhen reicht es, die Versionsnummer des Buches auf den gewünschten Wert zu setzen. Beim Speichern des Buches in XML wird automatisch die Versionsnummer des übergeordneten Buchs für alle Unterelemente übernommen, deren Versionsnummer kleiner ist.

Attribut für die Kapitelnummerierung

Zusätzlich zu den benötigten Attributen besitzen die „Kapitel"-Elemente ein optionales Attribut „Nummerierung". Bei diesem Attribut haben Sie die folgenden Einstellungsmöglichkeiten:

- *numerisch*
  - Die Kapitelnummer ist eine arabische Ziffer, die bei aufeinander folgenden Kapiteln mit dieser Nummerierung innerhalb eines dpml-Buchs ansteigt (Voreinstellung).

    Sie müssen die Absatznummerierungs-Einstellung im dpml-Buch anpassen, damit die Kapitelnummer richtig angezeigt wird. Ansonsten sehen sie stets eine „1".

    Weitere Informationen zum Einstellen der Absatznummerierung im dpml-Buch finden Sie im Thema „Nummerierung einstellen" auf Seite A-43.

Fortsetzung nächste Seite ...

... Fortsetzung Kapitel-Attribute angeben

- einleitung
  - Die Kapitelnummer ist ein großes „E"
  - Benutzen Sie diese Einstellung für das Einleitungskapitel innerhalb einer mit dpml erstellten Dokumentation, um es deutlicher vom Hauptteil abzuheben
- sicherheit
  - Die Kapitelnummer ist ein großes „S"
  - Benutzen Sie diese Einstellung für das Kapitel mit den allgemeinen Sicherheitshinweisen innerhalb einer mit dpml erstellten Dokumentation, um es deutlicher vom Hauptteil abzuheben

Andere Elemente auf Kapitelebene innerhalb der dpml-Struktur werden automatisch richtig nummeriert. Inhaltsverzeichnisse oder andere generierte Listen bekommen keine Nummer, der Anhang wird mit einem großen „A" gekennzeichnet.

Die Nummer der Elemente auf Kapitelebene wird auch vor die Seitenzahl geschrieben. Dadurch ist es möglich, dass jedes Element im dpml-Buch auf Seite „1" beginnt und die Seitennummern trotzdem unterscheidbar bleiben. Einzelne Kapitel innerhalb einer gedruckten Dokumentation lassen sich so leicht austauschen, wenn neue Versionen eingepflegt werden.

Die Nummerierung der Elemente im dpml-Buch funktioniert nicht mithilfe der Einstellung der Kapitelnummer im Buch, die Sie möglicherweise aus anderen Dokumentationen mit FrameMaker kennen, sondern über eine durchgehende Absatznummerierung innerhalb des dpml-Buchs und die in den Elementen und Ihren Attributen festgelegte Nummerierung.

Die Nummerierung der Kapitel eines dpml-Buchs funktioniert nur dann richtig, wenn Sie ihre Absatznummerierung richtig einstellen.

Weitere Informationen zum Einstellen der Absatznummerierung im dpml-Buch finden Sie im Thema „Nummerierung einstellen" auf Seite A-43.

In der Dokumentstruktur navigieren

Strukturierte Inhalte

Um in strukturierten FrameMaker-Dokumenten Inhalte zu erfassen, müssen Sie die Texte an der richtigen Stelle in der Struktur, also im richtigen Element, eingeben. Dazu müssen Sie den Mauszeiger dort platzieren. In der Standardeinstellung des Dokumentfensters ist das immer dann schwierig, wenn sich Elementgrenzen im Dokument „überlagern".

Elementgrenzen ein-/ ausblenden

Neben den aus unstrukturierten Dokumenten bekannten Steuerzeichen für Umbrüche, Tabulatoren, etc. können zur leichteren Navigation in strukturierten Elementen zusätzlich über die Option „Darstellung → Elementgrenzen" oder „Darstellung → Elementgrenzen (mit Tags)" die als eckige Klammern dargestellten Grenzen von Elementen oder deren Start- und Endtags angezeigt werden. Dadurch können Sie den Cursor in der Dokumentansicht gezielter im gewünschten Element platzieren.

Eingeblendete Elementgrenzen werden beim Erstellen des PDF-Dokuments nicht mit ausgegeben.

Zur besseren Kontrolle des Drucklayouts sollten Sie diese daher ausgeblendet lassen.

In der Strukturansicht navigieren

Für eine einfachere Navigation bei gleichzeitiger Kontrolle des endgültigen Layouts in der Dokumentansicht sollten Sie die Elementgrenzen ausgeblendet lassen und mithilfe der Strukturansicht in dpml-Dokumenten navigieren.

Die Navigation in der Strukturansicht und das Eingeben von Text in strukturierten Dokumenten werden ausführlich in der FrameMaker-Hilfe erklärt. Für weiterführende Informationen öffnen Sie die Hilfe über „Hilfe → Hilfethemen" und wählen dort unter „Inhalt → Bearbeiten von Text → Arbeiten mit Text in strukturierten Dokumenten" das entsprechende Thema.

Layout der Dokumente steuern

Achtung!

Wenn Sie in dpml-Dokumenten die Formatierungsfunktionen für unstrukturierte Dokumenten (Absatz-, Zeichenkatalog, Quick-Formatleiste, etc.) benutzen, gehen die geänderten Formate beim Speichern in dpml verloren.

Benutzen Sie ausschließlich die Formatierungsmöglichkeiten, die Ihnen über die dpml-Struktur zur Verfügung stehen und die im Folgenden erklärt werden.

Elemente statt Zeichenformate

Um Textstellen hervorzuheben, Zeichen hoch- oder tiefzustellen usw., werden in unstrukturierten Dokumenten Zeichenformate benutzt. In dpml stehen Ihnen Elemente zur Verfügung, die Sie in Texten anstelle von Zeichenformaten verwenden. Diese Elemente werden auch in der XML-Struktur gespeichert, so dass die Informationen außerhalb von FrameMaker nicht verloren gehen. Die Standard-dpml-Elemente für diesen Zweck sind:

- Betont: Textbereich wird – abhängig vom Kontext im Dokument – hervorgehoben, dabei meist kursiv geschrieben
- FettBetont: Textbereich wird durch fette Schrift im Text stärker hervorgehoben
- Hochgestellt: Textbereich wird hochgestellt
- Tiefgestellt: Textbereich wird tiefgestellt

Attribute steuern Layoutvarianten

In dpml steuern Sie bei einigen Elementen Layoutvarianten mithilfe von Attributen.

Das gilt beispielsweise für Listen und Unterlisten. Jede Liste und jede Unterliste besitzt ein Attribut, mit dem Sie angeben können, ob es sich um eine nummerierte Liste oder um eine Strichliste handeln soll. Die Voreinstellung dieses Attributs ist „nummeriert", über einen Attributemanager können Sie den Wert auf „strichliste" umschalten. Sobald der Wert eingestellt ist, wird die Veränderung des Layouts im Dokumentenfenster sichtbar.

Weitere Informationen zum Arbeiten mit Attributemanagern finden Sie unter „Dokumente mit dpml strukturieren – Attributemanager für optionale Attribute" auf Seite A-13

Fortsetzung nächste Seite ...

... Fortsetzung Layout der Dokumente steuern

Attribute steuern Symbole

Ein Sonderfall für die Steuerung von Layoutvarianten mit Attributen ist die Auswahl der Symbole für Sicherheitshinweise und Tipps in dpml-Dokumentationen.

Diese Symbole müssen nicht mithilfe von Grafikelementen platziert werden, sondern sie werden ebenfalls über ein Attribut im jeweiligen Element gesteuert. Die Voreinstellung des Attributwerts für Sicherheitshinweis und Tipp ist dabei „allgemein". In Sicherheitshinweisen wird bei dieser Einstellung ein allgemeines Gefahrenzeichen („gelbes Ausrufedreick") und in Tipps ein Informationszeichen („blaues Infoviereck") angezeigt.

Die Symbolgrafiken sind fest auf den Vorgabeseiten des Dokuments gespeichert. Für jedes dpml-Template steht Ihnen eine auf Ihren Anwendungszweck angepasste Teilmenge der riesigen Menge an in dpml verfügbaren Symbolen zur Verfügung, damit die FrameMaker-Dateien nicht zu groß werden. Wenden Sie sich an den Template-Designer, wenn Sie ein zusätzliches Symbol benötigen.

Wenn Sie in den Elementen für Tipps und Sicherheitshinweise in Listen (TippKlein, SicherheitshinweisKlein) Text eingeben, sieht es so aus, als ob dort kein Text geschrieben wird. Das ist ein Anzeigefehler von FrameMaker, der auftritt, solange Sie in der Zeile neben dem Symbol schreiben.

Um die Darstellung zu korrigieren drücken Sie die Tastenkombination „Strg+l". Die Dokumentansicht wird dadurch aktualisiert und richtig angezeigt.

Vorgabeseiten automatisch zuordnen lassen

In dpml werden anhand der vorhandenen Elemente und der Einstellungen des Attributs „Vorgabeseite", das einige Elemente besitzen, den Arbeitsseiten im Dokument automatisch Vorgabeseiten zugewiesen.

Die automatische Vorgabeseitenzuordnung wird nicht ständig in Echtzeit aktualisiert, sondern Sie müssen den Prozess selbst in Gang bringen.

Wählen Sie dazu im Menü „Format → Seitenlayout → Vorgabeseiten zuweisen..." und bestätigen Sie die erscheinende Meldung, um die automatische Zuordnung zu starten.

Fortsetzung nächste Seite ...

... Fortsetzung Layout der Dokumente steuern

Automatische Vorgabeseitenzuordnung steuern

Bei der automatischen Vorgabeseitenzuordnung bestimmen einige Elemente welche Vorgabeseite zugeordnet wird, z.B. Kapitel. Über das Attribut „Vorgabeseite", das Ihnen bei den meisten Blockelementen wie Absatz oder Tipp zur Verfügung steht, können Sie die Zuordnung selbst beeinflussen. Sinn dieser Einstellungen ist, dass zusammenhängende Informationseinheiten, die Seitenumbrüche enthalten, als solche gekennzeichnet werden, indem am Ende einer Seite die Zeile „Fortsetzung nächste Seite..." und am Anfang der nächsten Seite die Zeile „...Fortsetzung *Thema*" erscheint.

Die möglichen Werte des Attributs „Vorgabeseite" sind:

- **ohneUmbruch:** Voreinstellung; weist der aktuellen Seite die Standardvorgabeseite ohne Fortsetzungszeilen zu
- **umbruch:** weist der aktuellen Seite die Vorgabeseite mit Fortsetzungszeile am Ende der Seite zu
- **fortgesetzt:** weist der aktuellen Seite die Vorgabeseite mit Fortsetzungszeile am Anfang der Seite zu
- **umbruchFortgesetzt:** weist der aktuellen Seite die Vorgabeseite mit Fortsetzungszeilen am Anfang und am Ende der Seite zu; für Seiten innerhalb zusammenhängender Informationseinheiten, die sich über mehrere Seiten erstrecken

Bei der automatischen Vorgabeseitenzuordnung erhalten Sie möglicherweise eine Fehlermeldung, wenn ein Element, das die Zuordnung steuert, bei der Zuordnung auf die nächste Seite verschoben wird und die Vorgabeseite deshalb nicht zugeordnet werden konnte. Das ist möglich, weil die Textbereiche auf Seiten, die Fortsetzungszeilen enthalten, kleiner sind, als auf Seiten ohne Fortsetzungszeilen.

Starten Sie die automatische Zuordnung nach der Fehlermeldung neu. Tritt die Fehlermeldung wiederholt auf, müssen Sie auf der entsprechenden Seite bei einem anderen Element mit dem Attribut „Vorgabeseite" die Zuordnung steuern.

Für ein besseres Verständnis der automatischen Vorgabeseitenzuordnung sehen Sie sich die Tabelle „strukturierte Vorgabeseitenzuordnung" in einem dpml-Dokument auf den Referenzseiten an. Diese steuert die Zuordnung. Vergleichen Sie die in der Tabelle eingetragenen Werte mit den Vorgabeseiten.

Fortsetzung nächste Seite ...

... Fortsetzung Layout der Dokumente steuern

Seitenumbruch steuern

In unstrukturierten Dokumenten fügen Sie Seitenumbrüche ein, indem Sie das Format eines Absatzes so verändern, dass er auf einer neuen Seite beginnt. Die gleiche Funktionalität steht Ihnen mithilfe des entsprechenden Attributs „AufNeueSeite" in den meisten Blockelementen auch in dpml zur Verfügung.

Achtung!

Eine Veränderung des Absatzformates für Seitenumbrüche wie in unstrukturierten Dokumenten sollten Sie in dpml nicht vornehmen. Steuern Sie den Seitenumbruch ausschließlich über das Attribut „AufNeueSeite".

Ändern Sie den Attributwert des Attributs „AufNeueSeite" von der Voreinstellung „nein" auf „ja".
⇒ Das Element wird auf die nächste Seite verschoben

Setzen Sie manuelle Seitenumbrüche mit dem „AufNeueSeite"-Attribut sparsam ein. Wenn Sie vor einem solchen Seitenumbruch zusätzlich soviel Text einfügen, dass ein automatischer Seitenumbruch erzeugt wird, führt das durch den doppelten Umbruch zu ungewünschten Ergebnissen.

Elemente wie Kapitel oder Abschnitte erzeugen zur besseren Strukturierung bereits Umbrüche. Außerdem werden zusammenhängende Informationseinheiten – z. B. Sicherheitshinweise – nur zusammenhängend umgebrochen. Ein dpml-Dokument ist so auch ohne manuelle Umbrüche meist gut strukturiert.

Kapitel- und Abschnitts-Inhaltsverzeichnisse

Zusätzliche Verzeichnisse

In dpml gibt es in Kapiteln und in Kapitelabschnitten zusätzlich zum Gesamtinhaltsverzeichnis im Buch einzelne Übersichten über den Inhalt des jeweiligen Kapitels oder Kapitelabschnitts.

Diese Inhaltsverzeichnisse sind als Listen formatierter Querverweise angelegt.

Sie müssen die Kapitel- und Abschnitts-Inhaltsverzeichnisse nicht selbst mit den entsprechenden Elementen ausfüllen, wenn Sie dpml-Dokumente erstellen. Beim Speichern der XML-Daten werden diese automatisch erzeugt.

Automatismus nutzen

Führen Sie die folgenden Schritte durch, um Kapitel- und Abschnitts-Inhaltsverzeichnisse automatisch erstellen zu lassen:

1. Fügen Sie nach dem Kapitel- oder AbschnittZweckblock das jeweilige Element für den Inhaltsüberblick (Kapitel- oder AbschnittInhaltblock) ein
2. Lassen Sie das Element leer (Sie könnten es auch von Hand ausfüllen)

   Die Kapitel- und Abschnitts-Inhaltsübersichten werden beim XML-Export automatisch erzeugt

Die automatische Generierung findet erst statt, wenn Sie die Buchdatei in XML speichern. Vorher können Sie den Prozess nicht in Gang setzen. Nach dem XML-Wiederimport können Sie das Ergebnis im FrameMaker überprüfen.

A.3 Grafik einbinden

Überblick

Zweck In diesem Abschnitt wird erläutert, was Sie bei der Verwendung von Grafiken in mit der dpml-Anwendung für Adobe FrameMaker erstellten Dokumenten beachten müssen.

Wenn Sie die hier angegebenen Regeln einhalten, haben Sie die Möglichkeit, hochwertige Grafiken in Ihren Dokumenten zu verwenden und die gleiche Qualität beim Publizieren in XML und im PDF für die Druckausgabe zu erhalten.

Inhalt Dieser Abschnitt enthält die folgenden Informationen:

CGM-Grafikdateien bereitstellen

Achtung!

Bei der Einbindung von Grafiken in dpml-Dokumente, dem anschließenden Export und Wiederimport von XML mit der dpml-Anwendung und beim Erzeugen der PDF-Dokumente kann es bei falsch gewählten Grafikformaten zu großen Qualitätsverlusten kommen.

Beachten Sie daher die folgenden Hinweise.

Grafik in XML-Anwendungen

Generell gibt es für FrameMaker viele geeignete Grafikformate, allen voran EPS (Encapsulated Postscript). Durch die Verwendung einer XML-Anwendung wie dpml werden diese Möglichkeit allerdings stark eingeschränkt. Das gilt insbesondere dann, wenn XSLT eingesetzt wird und die Grafiken deshalb gemeinsam mit den Textdaten in XML exportiert und wieder importiert werden müssen.

Der XML-Exportfilter für Grafiken versagt z.B. bei EPS-Dateien mit eingebetteter TIFF-Vorschau. Der Exportvorgang bricht dabei mit einer Fehlermeldung ab.

CGM – das richtige Format in XML-Anwendungen

Das CGM-Grafikformat (Computer Graphic Metafile) brachte bei den Tests des dpml-Teams Halle die besten Ergebnisse beim XML-Export. Dabei handelt es sich um ein standardisiertes Format, das ähnlich wie EPS-Dateien sowohl Vektor- als auch Pixelgrafik in hoher Qualität enthalten kann. Damit funktionierte der Export in allen bisherigen Tests problemlos. Selbst im FrameMaker hinzugefügte kleine Grafikelemente und Beschriftungen wurden problemlos mit exportiert.

CGM-Datei bereitstellen

Grafikdateien sollten Sie für die Verwendung in dpml im CGM-Format bereitstellen. Fordern Sie einzubettende Grafiken am besten gleich als CGM-Datei von Grafikern an. Wenn Sie selbst Grafiken zum Einbetten in dpml erstellen, exportieren Sie die CGM-Dateien aus den gängigen Vektorgrafikprogrammen, wie z.B. Adobe Illustrator, Freehand und Corel Draw.

Grafik-Element einfügen

Anleitung Führen Sie die folgenden Schritte durch, um eine CGM-Grafikdatei in ein dpml-Dokument einzufügen:

1. Navigieren Sie in der Strukturansicht im Dokument auf die gleiche Ebene in der Struktur, in der beispielsweise das einfache Absatz-Element liegt.
   ⇒ Das Element „Abbildung" ist im Elementkatalog wählbar.
2. Fügen Sie das Element „Abbildung" aus dem Elementkatalog ein.
   ⇒ Das Element „GRAPHIK" ist im Elementkatalog wählbar.
3. Fügen Sie das Element „GRAPHIK" aus dem Elementkatalog ein.
   ⇒ Der Dialog „Verankerter Rahmen" öffnet sich.
4. Klicken Sie auf die Schaltfläche „Neuer Rahmen".
   ⇒ Das Element „GRAPHIK" wird als verankerter Rahmen eingefügt.
5. Wählen Sie bei aktivem „GRAPHIK"-Element (Anfasser zum Skalieren sind sichtbar) aus dem Menü „Datei → Importieren → Datei..."
   ⇒ Der Dialog „Importieren" öffnet sich.
6. Navigieren Sie im Dialog „Importieren" zur CGM-Grafikdatei Ihrer Wahl und wählen Sie ganz unten im Dialog eine der beiden Optionen „Importieren durch Referenz" oder „Datei in Dokument kopieren".

   Importieren Sie die Datei durch Referenz, wird nur der Pfad zur CGM-Datei im Dokument gespeichert. Diese Verknüpfung funktioniert deshalb nicht mehr, wenn Sie die FrameMaker-Datei später verschieben.
   Kopieren Sie die Datei in das Dokument, wird die gesamte Grafikdatei im FrameMaker-Dokument eingebettet. Die Verknüpfung kann so nicht verloren gehen. Gleichzeitig nimmt die Dateigröße aber zu.
7. Klicken Sie auf die Schaltfläche „Importieren", um Ihre Auswahl zu bestätigen.
   ⇒ Die ausgewählte Grafik erscheint im Dokument.

Grafik anpassen und beschriften

Einmal in ein Grafikelement eingefügte CGM-Grafiken können Sie in dpml-Dokumenten genauso nachbearbeiten wie in unstrukturierten FrameMaker-Dokumenten.

Sämtliche Änderungen werden beim Exportieren nach XML übernommen, weil hierbei für jedes Grafikelement eine neue CGM-Datei erzeugt wird, die die Änderungen enthält.

Grafiken skalieren

Sie können den verankerten Rahmen der Grafik beliebig in der Höhe skalieren, wenn der Rahmen aktiv ist und die entsprechenden Anfasser angezeigt werden. Wenn Sie den Rahmen in der Breite so skalieren, dass die Grafik nicht im Fließtextbereich, sondern auch im Bereich der seitlichen Überschriften liegt, müssen Sie den Wert des Attributs „linkerRand" des übergeordneten Abbildungselement anpassen.

Achtung!

Eine falsche Einstellung des Attributs „linkerRand" des Elements „Abbildung" führt zu ungünstigen Abständen zu vor oder hinter der Abbildung liegenden Elementen und zu Problemen bei der Anzeige der wiederimportierten CGM-Grafiken.

Stellen Sie das Attribut „linkerRand" des Elements „Abbildung"

- auf „mitSeitenüberschriftsBereich" ein,
  - wenn die Grafik breiter als der Textbereich skaliert ist, also auch im Bereich der Seitenüberschriften liegt.
- auf „inSpalte" (Voreinstellung) ein,
  - wenn die Grafik maximal so breit wie die Fließtextspalte skaliert ist.

Beschriftungen und Grafikelemente einfügen

Sie können den CGM-Grafiken mit den FrameMaker-eigenen Grafikwerkzeugen Beschriftungen und kleine grafische Elemente hinzufügen. Das funktioniert genauso wie in unstrukturierten Dokumenten. Achten Sie lediglich darauf, die Beschriftungen und Grafikelemente nur innerhalb des verankerten Rahmens im Element „Abbildung" einzufügen, damit diese beim XML-Export mit gespeichert werden.

Weitere Informationen zur Grafikbearbeitung mit FrameMaker finden Sie in der FrameMaker-Hilfe unter „Graphiken".

A.4 Unstrukturierte Dokumente importieren

Überblick

Zweck In diesem Abschnitt finden Sie Anleitungen zum Import von unstrukturierten FrameMaker-Dokumenten in ein mit der dpml-Anwendung strukturiertes Dokument.

Lesen Sie diesen Abschnitt und befolgen Sie die gegebenen Hinweise, wenn Sie Informationen aus einer FrameMaker-Dokumentation, die nicht mit der dpml-Anwendung erstellt wurde, importieren wollen.

Inhalt Dieser Abschnitt enthält die folgenden Informationen:

Kopieren und Einfügen

Texte aus unstrukturierten Dokumenten

Texte aus bereits vorhandenen unstrukturierten, mit Adobe FrameMaker erstellten Dokumentationen übernehmen Sie Absatz für Absatz in die dpml-Struktur, indem Sie diese über die Zwischenablage kopieren und ohne Formatierungen einfügen. Gehen Sie dazu in den folgenden Schritten vor:

1. Markieren Sie den Text, den Sie kopieren wollen im unstrukturierten Dokument und wählen Sie im Menü „Bearbeiten → Kopieren" oder drücken Sie die Tastenkombination „Strg+c"
   ⇒ Der ausgewählte Text ist in die Zwischenablage kopiert
2. Wechseln Sie in das mit dpml strukturierte Dokument

   Zwischen mehreren, in FrameMaker geöffneten Dokumenten können Sie mit der Tastenkombination „Strg+Tab" schnell umschalten
3. Erstellen Sie das Element in der dpml-Struktur, in das der kopierte Text eingefügt werden soll
4. Wählen Sie im Menü „Bearbeiten → Einfügen Spezial..."
   ⇒ Der Dialog „Inhalte einfügen" öffnet sich
5. Wählen Sie im Dialog „Inhalte einfügen" die Option „Einfügen" und unter „Als" die Einstellung „Text"
6. Klicken Sie auf die Schaltfläche „Einfügen", um die Einstellung zu bestätigen und den Text aus der Zwischenablage im Element zu platzieren
   ⇒ Der Text erscheint im strukturierten Dokument

   Wenn Sie den Text aus dem unstrukturierten Dokument nicht als Text mithilfe des Befehls „Einfügen Spezial...", sondern lediglich über „Einfügen" bzw. die Tastenkombination „Strg+v" aus der Zwischenablage platzieren, werden die Formatierungen aus dem unstrukturierten Dokument übernommen. Da dies meist zu ungewünschten Resultaten führt, sollten Sie Texte aus anderen FrameMaker-Dokumenten nur wie oben beschrieben in dpml-Dokumente einfügen.

Zeichenformate und Indexmarken

Achtung!

Wenn Sie Texte mit Zeichenformaten und/oder Indexmarken aus unstrukturierten Dokumenten in dpml-Dokumente kopieren, werden diese in der Struktur nicht übernommen.

Am deutlichsten sehen Sie das, wenn Sie die Texte aus der Zwischenablage mithilfe des Befehls „Einfügen Spezial…" als Text ohne Formatierungen einfügen, wie im letzten Thema beschrieben. Dieses Ergebnis bekommen Sie mit der dpml-Anwendung spätestens, wenn Sie die Datei als XML speichern und wieder importieren. Dabei werden keinerlei hart formatierte oder über Zeichenformate hervorgehobene Texte übernommen. Das gleiche gilt für die Indexmarken.

Sämtliche verwendete Zeichenformate und Indexmarken müssen Sie daher nach dem Kopieren und Einfügen eines Textes aus einem unstrukturierten in ein strukturiertes FrameMaker-Dokument mit Elementen der Struktur abbilden.

Weitere Informationen zu den mit dpml verfügbaren Elementen für Textbereichformatierungen finden Sie im Thema „Layout der Dokumente steuern – Elemente statt Zeichenformate" auf Seite A-20

Indexmarken-Element

Führen Sie die folgenden Schritte durch, um ein Indexmarken-Element in ein dpml-Dokument einzufügen:

1. Navigieren Sie im dpml-Dokument in ein Element, für das Sie aus dem Elementkatalog das Element „IndexMarke" einfügen können.

   Das „IndexMarke"-Element können Sie in den meisten Elementen einfügen, die auch Text enthalten, also zum Beispiel in Absätzen, Überschriften oder Hinweisen.

2. Fügen Sie das Element „IndexMarke" ein.
   ⇒ Der Dialog „Marke einfügen" öffnet sich.
3. Geben Sie den gewünschten Text für die Indexmarke ein.

Der Standardindex wird im dpml-Buch aus den Inhalten der Indexmarken-Elemente erzeugt. Mehr Informationen zu Indexmarken-Elementen finden Sie in der FrameMaker-Hilfe unter „Inhaltsverzeichnisse und Indizes → Hinzufügen von Indexmarken-Elementen in strukturierten Dokumenten".

Grafiken und Tabellen

Die Elemente für Grafiken und Tabellen sind in dpml so definiert worden, dass sie über Kopieren und Einfügen aus unstrukturierten Dokumenten übernommen werden können.

Grafiken kopieren und einfügen

Führen Sie die folgenden Schritte durch, um eine Grafik aus einem unstrukturierten Dokument zu kopieren und in einem dpml-Dokument einzufügen:

Prüfen Sie, dass alle Grafikelemente im Grafikrahmen, den Sie aus dem unstrukturierten Dokument kopieren wollen, im CGM-Format vorliegen. Nur beim Einsatz von CGM-Grafiken ist die vollständige Kompatibilität mit dpml beim XML-Eport und XML-Wiederimport garantiert.

1. Markieren Sie den gewünschten verankerten Grafikrahmen im unstrukturierten Dokument.
2. Wählen Sie im Menü „Bearbeiten → Kopieren" oder benutzen Sie die Tastenkombination „Strg+c".
   ⇒ Der verankerte Grafikrahmen wird mit seinem gesamten Inhalt in die Zwischenablage kopiert.
3. Wechseln Sie in das mit dpml strukturierte Dokument.

   Zwischen mehreren, in FrameMaker geöffneten Dokumenten können Sie mit der Tastenkombination „Strg+Tab" schnell umschalten.
4. Erstellen Sie ein „Abbildung"-Element in der dpml-Struktur.
5. Falls die Grafik, die eingefügt werden soll, breiter als die Textspalte für den Fließtext ist, ändern Sie den Wert des Attributs „linkerRand" des Elements „Abbildung" von der Voreinstellung „inSpalte" auf „mitSeitenüberschriftsBereich".
6. Wählen Sie im Menü „Bearbeiten → Einfügen" oder benutzen Sie die Tastenkombination „Strg+v".
   ⇒ Der verankerte Grafikrahmen wird mit seinem gesamten Inhalt aus der Zwischenablage im dpml-Dokument eingefügt.

Fortsetzung nächste Seite ...

... Fortsetzung Grafiken und Tabellen

Tabellen kopieren und einfügen

Führen Sie die folgenden Schritte durch, um eine Tabelle aus einem unstrukturierten Dokument zu kopieren und in einem dpml-Dokument einzufügen:

Die folgenden Anweisungen gelten nur für Tabellen aus unstrukturierten Dokumenten, die ausschließlich Text ohne Zeichenformate und keine zusätzliche Grafiken enthalten. Enthält die Tabelle Texte mit Zeichenformaten oder Grafiken, müssen Sie diese Strukturen nachträglich in dpml-Elementen abbilden, damit der XML-Export und -Wiederimport problemlos funktioniert.

1. Markieren Sie die gewünschte Tabelle inklusive aller Tabellenzellen im unstrukturierten Dokument.
2. Wählen Sie im Menü „Bearbeiten → Kopieren" oder benutzen Sie die Tastenkombination „Strg+c".
   ⇒ Die Tabelle wird mit allen Tabellenzellen in die Zwischenablage kopiert.
3. Wechseln Sie in das mit dpml strukturierte Dokument.

   Zwischen mehreren, in FrameMaker geöffneten Dokumenten können Sie mit der Tastenkombination „Strg+Tab" schnell umschalten.
4. Erstellen Sie ein „Tabellencontainer"-Element in der dpml-Struktur.
5. Wählen Sie im Menü „Bearbeiten → Einfügen" oder benutzen Sie die Tastenkombination „Strg+v".
   ⇒ Die Tabelle wird mit allen Tabellenzellen aus der Zwischenablage im dpml-Dokument eingefügt.

Sie können anstelle der ganzen Tabelle auch nur Spalten oder Zellen aus unstrukturierten FrameMaker-Tabellen kopieren und in dpml-Tabellen einfügen, indem Sie vor dem Kopieren einen kleineren Bereich der Tabelle markieren.

A.5 Strukturierte Dokumente publizieren

Überblick

Zweck Dieser Abschnitt enthält Informationen und Anleitungen zum Zusammenstellen und Publizieren der strukturierten Kapitel-Dateien mithilfe der dpml-Anwendung und der Buchfunktion von Adobe FrameMaker.

Inhalt Dieser Abschnitt enthält die folgenden Informationen:

Strukturierte Buchdatei anlegen

Buchverzeichnis anlegen

Legen Sie ein eigenes Dateiverzeichnis an, in dem Sie alle Kapitel des Buches ablegen. Geben Sie dem Verzeichnis einen für Ihr Projekt aussagefähigen Namen.

Berücksichtigen Sie bei der Benennung des Buchverzeichnisses, dass Sie neben diesem Ordner während des Publizierungsprozesses mit dpml noch zwei weitere anlegen: einen für die exportierten XML-Daten und einen für die aus den XML-Daten wieder importierten FrameMaker-Daten.

Buchtemplate in das Buchverzeichnis kopieren

Zu Ihrer dpml-Anwendung gehört ein Buchtemplate. Ein Buchtemplate ist die Vorlagedatei für ein Buch zur Verwendung mit der dpml-Anwendung. Das dpml-Buchtemplate ist eine strukturierte Buchdatei, die lediglich das Element Buch enthält.

Achtung!

Öffnen Sie das Buchtemplate niemals direkt, da Sie sonst das Template überschreiben können und den Zugriff anderer Redakteure verhindern, falls das Buchtemplate in einer Mehrbenutzerumgebung im Netzwerk verfügbar ist.

Kopieren Sie das Buchtemplate in Ihr Buchverzeichnis. Wenn Ihnen der Speicherort der Buchtemplate-Datei nicht bekannt ist, fragen Sie Ihren Systemadministrator.

Buchdatei vorbereiten

- Benennen Sie das in Ihrem Buchverzeichnis liegende Buchtemplate um.

  Wählen Sie einen für Ihr Projekt aussagefähigen Namen, so dass Sie die Datei später zuordnen können.

- Öffnen Sie die Buchdatei zur weiteren Bearbeitung.

  Schalten Sie den Elementkatalog und die Strukturansicht ein, falls diese noch nicht sichtbar sind.

- Geben Sie die noch fehlenden, benötigten Werte für die Attribute „Produkt", „ProduktTyp", „Hersteller", „DokumentTyp" und „Version" des Buch-Elements an.

  Weitere Informationen zu den Attributen finden Sie im Thema „Kapitel-Attribute angeben" auf Seite A-16, weitere Informationen zu Attributmanagern finden Sie im Thema „Dokumente mit dpml strukturieren – Attributemanager für optionale Attribute" auf Seite A-13.

Kapitel in das Buch aufnehmen

Dateien einfügen

Führen Sie die folgenden Schritte durch, um mit dpml strukturierte Kapiteldateien in ein dpml-Buch einzufügen:

1. Wählen Sie im Menü „Hinzufügen → Dateien..."
   oder klicken Sie auf die Schaltfläche im Buchfenster
   ⇒ Der Dialog „Dateien in Buch hinzufügen" öffnet sich
2. Wählen Sie im Dialog „Dateien in Buch hinzufügen" die Kapiteldateien aus, die Sie hinzufügen wollen

   Indem Sie die Steuerungs- oder die Hochstell-Taste beim Markieren mit der Maus gedrückt halten, können Sie mehrere Dateien zum Hinzufügen auswählen
3. Klicken Sie auf die Schaltfläche „Einfügen", um Ihre Auswahl zu bestätigen und den Dialog zu schließen
   ⇒ Die ausgewählten Dateien werden im Buch eingefügt

Buch aktualisieren

Nachdem Sie die Dateien in das Buch aufgenommen haben, werden diese in der Strukturansicht jeweils als „BUCHKOMPONENTE:" dargestellt. Diese Elemente scheinen in der Struktur nicht gültig zu sein. Lediglich am im Dokumentfenster angezeigten Dateinamen ist erkennbar, um welches Kapitel es sich handelt.

Führen Sie die folgenden Schritte durch, um die Strukturinformationen im dpml-Buch nach dem Hinzufügen von Kapiteldateien zu aktualisieren:

1. Wählen Sie im Menü „Bearbeiten → Buch aktualisieren..."
   oder klicken Sie auf die Schaltfläche im Buchfenster
   ⇒ Der Dialog „Buch aktualisieren" öffnet sich
2. Entfernen Sie alle gesetzten Haken in den Checkboxen vor den Optionen zur Aktualisierung

   Indem Sie keine der zusätzlichen Optionen wählen wird das Buch viel schneller aktualisiert.
3. Klicken Sie auf die Schaltfläche „Aktualisieren", um Ihre Auswahl zu bestätigen und den Dialog zu schließen
   ⇒ Nach kurzer Wartezeit ist das Buch aktualisiert; alle Kapitelelemente erscheinen in der Strukturansicht

   Sie können den Namen des Kapitels auch im Buchfenster an Stelle des Dateinamens einblenden. Klicken Sie dazu auf die Schaltfläche / .

Fortsetzung nächste Seite ...

... Fortsetzung Kapitel in das Buch aufnehmen

Buch weiterbearbeiten

Sie können das dpml-Buch nach dem Anlegen für verschiedene Aufgaben innerhalb Ihres Dokumentationsprojekts weiter bearbeiten:

- Sie können einer Buchdatei beliebig oft Kapiteldateien hinzufügen und das Buch aktualisieren
- Um die Reihenfolge der Kapitel im Buch zu verändern, können Sie diese entweder im Buchfenster oder in der Strukturansicht mit der Maus bewegen („Drag and Drop")
- Um Elemente im Buch zu löschen, markieren Sie diese einfach im Buchfenster oder in der Strukturansicht und drücken Sie die „Entf"-Taste auf der Tastatur
- In dpml-Büchern können Sie die Attribute aller Elemente in der Strukturansicht, z.B. von „Buch", „Kapitel" und „Anhang", zentral verwalten

In dpml-Büchern nehmen Sie auch die Einstellungen für die Absatz- und Seitennummerierung vor. Da diese Informationen aber beim XML-Export verloren gehen, setzen Sie die entsprechenden Werte am besten erst nach dem XML-Wiederimport.

Weitere Informationen zum Einstellen der Absatz- und Seitennummerierung im dpml-Buch finden Sie im Thema „Nummerierung einstellen" auf Seite A-43.

Automatisch generierte Listen

Veränderte Prozesse durch den Einsatz von XSLT

Die Vorgehensweise beim Generieren von Listen mithilfe eines dpml-Buches in der dpml-Anwendung unterscheidet sich grundlegend von der üblichen Vorgehensweise bei unstrukturierten FrameMaker-Buchdateien.

Die Listen werden nicht durch die im Buch integrierten Funktionen generiert, sondern beim XML-Export mithilfe eines XSLT-Stylesheets automatisch erzeugt. Der Standardindex bildet die Ausnahme von dieser Regel. Er wird erst nach dem XML-Wiederimport mit der klassischen Buchfunktion erzeugt.

Inhaltsverzeichnis erzeugen

Führen Sie die folgenden Schritte durch, um das Inhaltsverzeichnis beim XML-Export automatisch erzeugen zu lassen:

1. Navigieren Sie in der Strukturansicht an die mit einem roten Viereck als ungültig markierte Stelle unterhalb des Elements „Buch" vor dem ersten Kapitelelement.
   ⇒ Das Element „Inhaltsverzeichnis" ist im Elementkatalog wählbar.
2. Fügen Sie das „Inhaltsverzeichnis"-Element aus dem Elementkatalog ein.
   ⇒ Anstelle des roten Vierecks erscheint das Element „Inhaltsverzeichnis" im Strukturbaum.

   Wenn nach dem Einfügen des „Inhaltsverzeichnis"-Elements das rote Viereck unterhalb des Buchelements weiterhin sichtbar ist, muss in Ihrer Variante der dpml-Struktur vor dem Inhaltsverzeichnis zusätzlich ein Element für die Versionsnachweisliste eingefügt werden.
3. Fügen Sie keine Inhalte in das Element „Inhaltsverzeichnis" ein; geben Sie auch keine Werte für seine Attribute an.

   Die Inhalte und Attributwerte werden beim XML-Export der Buchdatei automatisch erzeugt.

In gleicher Weise wie das Inhaltsverzeichnis können weitere automatisch generierte Listen in die dpml-Struktur aufgenommen werden. Dazu gehören z.B. das Abbildungs- und Tabellenverzeichnis oder die Versionsnachweisliste.

Die verfügbaren Listen hängen von der Version der dpml-Anwendung ab, die Sie benutzen.

XML publizieren und öffnen

Struktur prüfen

Bevor Sie das gesamte Buch in XML-Daten exportieren, muss die Struktur auf Konsistenz geprüft werden. Fehlende oder falsch verschachtelte Elemente führen sonst beim XML-Export zu Fehlermeldungen und zum Abbruch.

Führen Sie die folgenden Schritte durch, um die Struktur des dpml-Buches auf Gültigkeit zu prüfen:

1. Wählen Sie im Menü „Element → Prüfen".
   ⇒ Der Dialog „Elementprüfung" öffnet sich.
2. Wählen Sie im Dialog „Elementprüfung" die Option „gesamtes Buch" und klicken Sie auf die Schaltfläche „Starten".
   ⇒ Die Prüfung der Elementstruktur wird gestartet. Dabei wird auch die Struktur der im Buch eingefügten Dateien auf Gültigkeit geprüft.
3. Wenn die Elementprüfung Fehler in der Buch-Struktur entdeckt, müssen Sie diese durch Anpassen beheben.

   Wird ein Fehler in der Struktur gefunden, dann springt der Cursor automatisch an die entsprechende Stelle im Dokument. Gleichzeitig wird im Dialog „Elementprüfung" eine Fehlerbeschreibung ausgegeben. Während Sie den Fehler beheben, können Sie den Dialog „Elementprüfung" geöffnet lassen. Hilfreich bei der Fehlerkorrektur ist, dass Fehler auch in der Strukturansicht durch rote Farbe gekennzeichnet werden und Sie im Elementkatalog die gültigen Elemente sehen können.

   Benutzen Sie keine der „ignorieren"-Optionen im „Elementprüfung" Dialog, lassen Sie auch keine Spezialfälle mit der entsprechenden Schaltfläche zu, sondern beheben Sie die Fehler sofort. Klicken Sie danach im Dialog „Elementprüfung" auf die Schaltfläche „Starten", um den Prüfvorgang fortzusetzen.

   Der Prüfvorgang wird immer von der aktuellen Cursorposition aus gestartet. Wenn das Ende des Dokuments erreicht ist, wird er am Anfang fortgesetzt.

Fortsetzung nächste Seite ...

... Fortsetzung XML publizieren und öffnen

4. Wird ein Fehler in der Struktur einer eingefügten Buchdatei gefunden, müssen Sie die entsprechende Datei öffnen und im Menü „Element → Prüfen" wählen, um eine Elementprüfung zu starten und den genauen Fehler zu beheben.

   Wird ein Fehler in einer im Buch eingefügten Datei gefunden, wird das entsprechende Element im Buchfenster und in der Strukturansicht markiert. Gleichzeitig wird im Dialog „Elementprüfung" die Meldung „Fehler bei Gültigkeitsprüfung in Datei" ausgegeben.
   Klicken Sie die eingefügte Datei im Buchfenster doppelt, um Sie direkt aus dem Buch zu öffnen.

   Setzen Sie nach der Elementprüfung in einer im Buch eingefügten Datei die Elementprüfung im Buch fort oder starten Sie diese erneut.

5. Wenn keine Fehler in der Struktur gefunden werden, erscheint im Dialog „Elementprüfung" die Meldung „Buch ist gültig".
   Schließen Sie in diesem Fall den Dialog „Elementprüfung".
   ⇒ Die Prüfung der Elementstruktur ist abgeschlossen.

Die Prüfung der Elementstruktur des Buches kann sehr lange dauern, weil sämtliche Dateien, die dem Buch hinzugefügt wurden, beim Prüfen durch FrameMaker geöffnet und danach wieder geschlossen werden.

Öffnen Sie alle im Buch eingefügten Dateien im FrameMaker, bevor Sie die Elementprüfung in der Buchdatei starten, um den Prüfprozess zu beschleunigen.

Buch als XML sichern

Führen Sie folgende Schritte durch, um Ihr dpml-Buch mit allen enthaltenen Dateien als XML-Daten zu speichern:

Das Sichern des dpml-Buches als XML dauert sehr lange, weil während des Exports auch die Inhaltsverzeichnisse generiert werden und dabei die im Buch enthaltenen Dateien durch FrameMaker mehrmals geöffnet und wieder geschlossen werden.
Öffnen Sie alle im Buch eingefügten Dateien im FrameMaker, bevor Sie das Buch als XML-Daten speichern, um den Vorgang etwas zu beschleunigen.

Fortsetzung nächste Seite ...

... Fortsetzung XML publizieren und öffnen

1. Wählen Sie im Menü „Datei → Buch sichern unter..."
   ⇒ Der Dialog „Buch speichern" öffnet sich.
2. Wählen Sie mit dem Dropdownmenü „Dateityp" im Dialog „Buch speichern" die Einstellung „XML (*.xml)".
   ⇒ Als Dateityp ist XML festgelegt. Im Fenster zur Auswahl des Speicherorts werden nur noch XML-Dateien angezeigt.
3. Wählen Sie im Fenster zur Auswahl des Speicherorts den Ordner für die XML-Daten in Ihrem Projekt aus.

   Legen Sie im Fensters zur Auswahl des Speicherorts einen neuen Ordner für die XML-Daten an, falls dieser in Ihrem Projekt noch nicht existiert.
4. Geben Sie im Kombinationsfeld „Dateiname" der Datei einen aussagefähigen Namen; achten Sie auf die richtige Dateiendung (.xml).
5. Klicken Sie auf die Schaltfläche „Speichern", um Ihre Einstellungen zu bestätigen und den XML-Export zu starten.
   ⇒ Der Dialog „Buch speichern" schließt sich.
   ⇒ Möglicherweise öffnet sich an dieser Stelle der Dialog „Strukturierte Anwendung verwenden".
   Wählen Sie in diesem Fall im Dropdown-Listenfeld die Einstellung „DPML" aus und klicken Sie auf die Schaltfläche „Fortfahren".
   ⇒ Der XML-Export wird gestartet.

   Beim Speichern in XML werden die FrameMaker-Daten in XML-Daten umgewandelt. Gleichzeitig werden die Listen, wie das Inhaltsverzeichnis oder die Inhaltsübersichten am Anfang von Kapiteln, generiert. Sämtliche in den FrameMaker-Dateien enthaltenen Grafiken werden in einzelne Dateien umgewandelt und im Ordner für die XML-Daten gespeichert. Der gesamte Prozess kann eine Weile dauern. Haben Sie Geduld.
6. Schließen Sie nach dem Speichern der XML-Daten das dpml-Buch in FrameMaker.

Im XML-Ordner Ihres Projekts werden alle Texte und Grafiken „FrameMaker-unabhängig" gespeichert. Bei der Archivierung des Projekts dürfen Sie diesen Ordner nicht vergessen. Durch XML haben Sie zukunftssichere, von zukünftig eingesetzten Programmen oder Programmversionen unabhängige Daten.

Fortsetzung nächste Seite ...

... Fortsetzung XML publizieren und öffnen

XML-Datei öffnen

Nachdem Sie die XML-Daten generiert und das dpml-Buch geschlossen haben, müssen Sie die XML-Daten in FrameMaker importieren, damit Sie die Druckversion der Dokumentation inklusive aller generierten Listen als PDF publizieren können.

Führen Sie die folgenden Schritte durch, um die XML-Daten in FrameMaker zu importieren:

1. Wählen Sie im Menü „Datei → Öffnen..."
   ⇒ Der Dialog „Dokument öffnen" öffnet sich.
2. Navigieren Sie im Dialog „Dokument öffnen" zum XML-Ordner Ihres Projekts; markieren Sie die XML-Datei, die Sie beim Speichern des dpml-Buches angelegt haben.
3. Klicken Sie auf die Schaltfläche „Öffnen".
   ⇒ Möglicherweise öffnet sich an dieser Stelle der Dialog „Strukturierte Anwendung verwenden".
   Wählen Sie in diesem Fall im Dropdown-Listenfeld die Einstellung „DPML" aus und klicken Sie auf die Schaltfläche „Fortfahren".
   ⇒ Der Dialog „Buch sichern" öffnet sich.
4. Wählen Sie im Dialog „Buch sichern" den Ordner für das aus den XML-Daten erzeugte Buch zum Erstellen des PDF.

   Legen Sie im Fensters zur Auswahl des Speicherorts einen neuen Ordner für das aus den XML-Daten erzeugte Buch an, falls dieser in Ihrem Projekt noch nicht existiert.
5. Geben Sie im Kombinationsfeld „Dateiname" der Datei einen aussagefähigen Namen. Achten Sie auf die richtige Dateiendung (.book).
6. Klicken Sie auf die Schaltfläche „Speichern", um Ihre Einstellungen zu bestätigen und den XML-Import zu starten
   ⇒ Der Dialog „Buch speichern" schließt sich. Der XML-Import wird gestartet.

   Beim Öffnen von XML werden die XML-Daten wieder in FrameMaker-Daten umgewandelt. Dabei wird ein Buch mit den dazugehörenden Kapiteldateien erzeugt und im Ordner Ihrer Wahl gespeichert. Die beim XML-Export generierten Listen werden berücksichtigt und alle Querverweise aktualisiert. Die Grafikdateien, die beim XML-Export erzeugt wurden, werden in die FrameMaker-Dokumente eingebettetet. Der gesamte Prozess kann eine Weile dauern. Haben Sie Geduld.

Nummerierung einstellen

Achtung!

Die Nummerierungseinstellungen für Kapitel im Buch müssen bei Absätzen und Seitennummern richtig gesetzt sein, damit sie im fertigen Dokument entsprechend übernommen werden.

Es gibt momentan keine Möglichkeit, diese Einstellungen in XML zu speichern. Sie müssen daher nach dem XML-Import neu gesetzt werden; auch dann, wenn Sie im dpml-Buch vor dem XML-Export bereits richtig eingestellt waren.

Absatznummerierung einstellen

Führen Sie die folgenden Schritte durch, um die Einstellung für die Absatznummerierung in allen Kapiteln des nach dem XML-Import generierten Buches zu korrigieren:

> Öffnen Sie alle Dateien des generierten Buches, um das Übernehmen der Einstellung für die Absatznummerierung zu beschleunigen.

1. Markieren Sie im Buchfenster alle Kapitel des Buches, außer dem ersten.

   > Klicken Sie dazu das zweite Kapitel des Buches an, halten Sie die Umschalt-Taste gedrückt und klicken Sie das letzte Kapitel im Buch an.

2. Wählen Sie im Menü „Format → Dokument → Nummerierung…" oder klicken Sie die Kapitel mit der rechten Maustaste an und wählen Sie im sich öffnenden Kontextmenü den Befehl „Nummerierung".
   ⇒ Der Dialog „Einstellungen für Nummerierung" öffnet sich.
3. Wählen Sie unter Absatz die Option „Nummerierung von vorigem Absatz im Buch weiterführen".
4. Klicken Sie auf die Schaltfläche „Zuweisen".
   ⇒ Der Dialog „Einstellungen für Nummerierung" wird geschlossen. Die Einstellung für die Absatznummerierung wird für alle markierten Kapitel des Buches übernommen.

   > Beim Übernehmen der Einstellung der Absatznummerierung wird diese nicht nur in der Buchdatei, sondern auch in den einzelnen Kapiteldateien gespeichert. Dieser Prozess kann eine Weile dauern. Haben Sie Geduld.

Fortsetzung nächste Seite …

... Fortsetzung Nummerierung einstellen

Seitennummerierung einstellen

Führen Sie die folgenden Schritte durch, um die Einstellung für die Seitennummerierung in allen Kapiteln des nach dem XML-Import generierten Buches zu korrigieren:

> Halten Sie alle Dateien des generierten Buches geöffnet, um das Übernehmen der Einstellung für die Seitennummerierung zu beschleunigen.

1. Markieren Sie im Buchfenster alle Kapitel des Buches, inklusive dem ersten.

   > Klicken Sie dazu das erste Kapitel des Buches an, halten Sie die Umschalt-Taste gedrückt und klicken Sie das letzte Kapitel im Buch an.

2. Wählen Sie im Menü „Format → Dokument → Nummerierung…" oder klicken Sie die Kapitel mit der rechten Maustaste an und wählen Sie im sich öffnenden Kontextmenü den Befehl „Nummerierung".
   ⇒ Der Dialog „Einstellungen für Nummerierung" öffnet sich.
3. Wählen Sie unter Seite die Option „Erste Seite Nr.: 1 → Format: Arabisch (14)".
4. Klicken Sie auf die Schaltfläche „Zuweisen".
   ⇒ Der Dialog „Einstellungen für Nummerierung" wird geschlossen. Die Einstellung für die Seitennummerierung wird für alle markierten Kapitel des Buches übernommen.

   > Beim Übernehmen der Einstellung der Seitennummerierung wird diese nicht nur in der Buchdatei, sondern auch in den einzelnen Kapiteldateien gespeichert. Dieser Prozess kann eine Weile dauern. Haben Sie Geduld.

Die Einstellung von Kapitelnummern entfällt in der dpml-Anwendung, weil diese mit der Absatznummerierung in Verbindung mit Attributen gelöst wurde.

Die immer bei „1" beginnende Seitennummerierung der Kapitel ermöglicht, dass sich neue Versionen eines Kapitels innerhalb einer gedruckten Dokumentation leicht einpflegen lassen, ohne dass die gesamte Dokumentation neu erstellt werden muss.

Standardindex erzeugen

Der Standardindex wird auf „klassische" Weise mithilfe der FrameMaker-Buchfunktion erzeugt. Diese Liste ist unstrukturiert und kann daher nicht in XML-Daten umgewandelt werden.

Der Standardindex wird deshalb, wenn benötigt, erst nach dem XML-Export und dem anschließenden XML-Import generiert.

Standardindex hinzufügen

Führen Sie die folgenden Schritte durch, um Ihrer Dokumentation einen Standardindex hinzuzufügen:

1. Markieren Sie im Buchfenster das letzte Kapitel.

   Bei Korrekturversionen liegt hinter dem Standardindex eventuell noch ein Kommentarverzeichnis. Markieren Sie in diesem Fall das Kapitel vor dem Kommentarverzeichnis.

2. Wählen Sie im Menü „Hinzufügen → Standardindex..."
   ⇒ Der Dialog „Einstellungen für Standardindex" öffnet sich.
3. Klicken Sie auf die Schaltfläche „Einfügen", um die Voreinstellungen für den Standardindex zu bestätigen.
   ⇒ Der Standardindex wird generiert; in der Strukturansicht erscheint er als „BUCHKOMPONENTE:"

   Weitere Informationen zu den Einstellungsmöglichkeiten für Indizes finden Sie in der FrameMaker-Hilfe unter dem Punkt „Inhaltsverzeichnisse und Indizes".
4. Markieren Sie das generierte Element „BUCHKOMPONENTE:" in der Strukturansicht.
5. Wählen Sie im Elementkatalog das Element „Standardindex" und klicken Sie auf die Schaltfläche „Eingliedern".
   ⇒ Das Element „BUCHKOMPONENTE:" wird in das Element „Standardindex" eingegliedert und dadurch in der Struktur gültig.

   Der Standardindex ist unstrukturiert. Sein Status als generierte Liste lässt sich nicht in XML-Daten übernehmen. Dadurch, dass Sie den Index eingliedern, könnten Sie, falls Sie weitreichende Änderungen vornehmen, das Buch trotzdem erneut als XML-Datei speichern. Das Element „Standardindex" wird in diesem Fall beim Export geleert. Beim erneuten XML-Import wird es nicht in die FrameMaker-Struktur übernommen, so dass Sie den Index neu erzeugen müssen.

... Fortsetzung PDF publizieren

PDF publizieren

Als letzten Schritt im dpml-Publishingprozess mit Adobe FrameMaker erstellen Sie das PDF.

Mit der PDF-Datei wird die Dokumentation ausgedruckt. Außerdem ist es gut für den Austausch von Korrekturversionen und als Format für die Druckerei geeignet. Als Onlinedateiformat mit Hypertextfunktionen (z.B. PDF-Lesezeichen) können Sie es zusätzlich als interaktives Benutzerhandbuch einsetzen.

Buch als PDF speichern

Führen Sie die folgenden Schritte durch, um das Buch als PDF zu speichern:

Erstellen Sie das PDF-Dokument erst, nachdem Sie alle anderen Schritte im dpml-Publishingprozess durchgeführt haben. Damit alle zur Dokumentation gehörenden Listen vorhanden sind, müssen Sie das Buch als XML gespeichert und die gespeicherten XML-Daten wieder geöffnet haben.

1. Wählen Sie im Menü „Datei → Buch sichern unter..."
   ⇒ Der Dialog „Buch speichern" öffnet sich.
2. Wählen Sie mit dem Dropdownmenü „Dateityp" im Dialog „Buch speichern" die Einstellung „PDF... (*.pdf)".
   ⇒ Als Dateityp ist PDF festgelegt. Im Fenster zur Auswahl des Speicherorts werden nur noch PDF-Dateien angezeigt.
3. Wählen Sie den Speicherort für die PDF-Datei.
4. Geben Sie im Kombinationsfeld „Dateiname" der Datei einen aussagefähigen Namen; achten Sie auf die richtige Dateiendung (.pdf).
5. Klicken Sie auf die Schaltfläche „Speichern", um Ihre Einstellungen zu bestätigen und die Generierung des PDFs zu starten.
   ⇒ Der Dialog „Buch speichern" schließt sich und der Dialog „PDF-Einstellungen für ausgewählte Dateien" öffnet sich.

Fortsetzung nächste Seite ...

... Fortsetzung PDF publizieren

6. Legen Sie die Einstellungen für das PDF fest. Achten Sie bei der Verwendung von Lesezeichen darauf, dass Sie als Quelle „Elemente" auswählen und dort nur die benötigten Elemente als Lesezeichen übernehmen.

   Weitere Informationen zu den Möglichkeiten der PDF-Generierung finden Sie in der FrameMaker-Hilfe unter „HTML- und Adobe PDF-Konvertierung".

7. Klicken Sie auf die Schaltfläche „Zuweisen", um Ihre Einstellungen zu bestätigen.
   ⇒ Das PDF wird mit den gewählten Einstellungen erzeugt.

Normalerweise müssen Sie die PDF-Einstellungen in Adobe FrameMaker nicht verändern, da diese im Template für die dpml-Anwendung auf geeignete Werte eingestellt wurden.

B Die DPML-DTD

An dieser Stelle wird die DTD für die dpML-Anwendung in Ihrem aktuellen Stand wiedergegeben. Für verschiedene Anpassungen sind in der DTD im Detail Veränderungen möglich. Das verwendete Tabellenmodell baut auf dem CALS-Tabellenmodell bzw. dem XML Exchange Table Model der OASIS auf und ist an den entsprechenden Stellen in der DTD durch Kommentare kenntlich gemacht.

```
<?xml version="1.0" encoding="UTF-8"?>

<!ENTITY % meta-attributes '
           docutype      CDATA    #REQUIRED
           version       NMTOKEN  #REQUIRED
           product       CDATA    #REQUIRED
           prodtype      CDATA    #REQUIRED
           manufacturer  CDATA    #REQUIRED'>

<!ENTITY % crossref-attributes '
           fmidref       IDREF    #REQUIRED
           format        CDATA    #IMPLIED
           srcfile       CDATA    #IMPLIED'>

<!ENTITY % safetysymbols '
           general|electricity|battery|explosive|flammable|
           toxic|corrosive|falling|slipping|tripping|
           crushing|crushing_foot|hand|blade|
           belt_hand_entangle|chain_hand_entangle|
           gear_hand_entangle|rollers_hand_entangle|
           hot_surface_din|hot_surface_ansi|suspended_loads|
           tipover|laser|light|magnet|radio_frequency|
           ground_conveyor|automatic_startup|biohazard|
           cold_burns|conveyor_in_railtrack|
           explosive_atmosphere|gas_cylinder|harmful_substance|
           lifting_heavy_object|humidity|inhalation|
           radioactive|oxidizing|ozone|pollution|
           overpressure|industrial_robot|tipping_rolling|
           shaft|entanglement|uv_light|vacuum|
           arc_flash_hand_ansi|arc_flash_person_ansi|
           falling_ansi|corrosive_ansi|explosion_ansi|
           slipping_ansi|tripping_ansi'>

<!ENTITY % infosymbols 'info|material|note|reference|tool|view'>

<!ENTITY % meta-elements 'docutype|manufacturer|product|prodtype'>
```

```
<!ENTITY % inline-elements 'emphasis|emphasisbold|
                            superscript|subscript'>

<!-- yesorno - ENTITY-declaration is for CALS Table Model -->

<!ENTITY % yesorno 'NMTOKEN'> <!-- no if zero(s),
                                   yes if any other value -->

<!ELEMENT book (frontpage,verver?,toc,chapter+,
                appendix?,index?,loa?,loc?)>
<!ATTLIST book
          %meta-attributes;>

<!ELEMENT frontpage EMPTY>
<!ATTLIST frontpage
          %meta-attributes;>

<!-- element verver: contains version-verification-table,
                     generated by XSLT -->
<!ELEMENT verver ((ververhead, table)?)>
<!ATTLIST verver
          %meta-attributes;>

<!ELEMENT ververhead EMPTY>

<!-- element toc: contains table of contents as a list of
                  crossreferences, generated by XSLT -->
<!ELEMENT toc (tochead?,tocentry*)>
<!ATTLIST toc
          %meta-attributes;>

<!ELEMENT tochead EMPTY>

<!ELEMENT tocentry (tocref?)>
<!ATTLIST tocentry
          tocentrytype (chapterlevel|sectionlevel|topiclevel|
                        appendixlevel|appendixtopiclevel|index)
                        "topiclevel">

<!ELEMENT tocref EMPTY >
<!ATTLIST tocref
          %crossref-attributes;>

<!ELEMENT chapter (head,chapterovw,(section+|topic+),emptypage?)>
<!ATTLIST chapter
          %meta-attributes;>

<!ELEMENT emptypage EMPTY>

<!-- element chapterovw: chapter-overview -->
```

```
<!ELEMENT chapterovw (chapterovwhead,chapterobj,chaptertoc)>

<!ELEMENT chapterovwhead EMPTY>

<!-- element chapterobj: chapter-object -->
<!ELEMENT chapterobj (chapterobjhead,chapterobjbody)>

<!ELEMENT chapterobjhead EMPTY>

<!ELEMENT chapterobjbody (para+)>

<!-- element chaptertoc: contains table of contents for chapter
                         as a list of crossreferences,
                         generated by XSLT -->
<!ELEMENT chaptertoc (chaptertochead?,chaptertocintro?,
                           chaptertocitem*)>

<!ELEMENT chaptertochead EMPTY>

<!ELEMENT chaptertocintro EMPTY>

<!ELEMENT chaptertocitem (chaptertocref)>

<!ELEMENT chaptertocref EMPTY>
<!ATTLIST chaptertocref
          %meta-attributes;>

<!ELEMENT section (head,sectionovw,topic+)>

<!-- element sectionovw: section-overview -->
<!ELEMENT sectionovw (sectionovwhead,sectionobj,sectiontoc)>

<!ELEMENT sectionovwhead EMPTY>

<!-- element sectionobj: section-object -->
<!ELEMENT sectionobj (sectionobjhead,sectionobjbody)>

<!ELEMENT sectionobjhead EMPTY>

<!ELEMENT sectionobjbody (para+)>

<!-- element sectiontoc: contains table of contents for section
                         as a list of crossreferences,
                         generated by XSLT -->
<!ELEMENT sectiontoc (sectiontochead?,sectiontocintro?,
                      sectiontocitem*)>

<!ELEMENT sectiontochead EMPTY>

<!ELEMENT sectiontocintro EMPTY>
```

```
<!ELEMENT sectiontocitem (sectiontocref)>

<!ELEMENT sectiontocref EMPTY>
<!ATTLIST sectiontocref
          %meta-attributes;>

<!ELEMENT topic (head,(safinst|tip)*,block+)>
<!ATTLIST topic
          pagetemplate (nobreak|break) "nobreak">

<!ELEMENT block (head, (para|safinst|tip|smallref|
                 figure|table|list|comment)+)>
<!ATTLIST block
          newpage (false|true) "false">

<!ELEMENT para (#PCDATA|%inline-elements;|indexmarker|crossref|
                %meta-elements;)*>
<!ATTLIST para
          correctspacebelow (false|true)  "false"
          pagetemplate (undefined|breakcontinued|continued) "undefined"
          newpage (false|true) "false">

<!ELEMENT head (#PCDATA|superscript|subscript|indexmarker|
                %meta-elements;)*>
<!ATTLIST head
          fmid    ID    #IMPLIED>

<!ELEMENT safinst (symbol,signalize,safinstpara+) >
<!ATTLIST safinst
          symbol (%safetysymbols;)  "general"
          correctspacebelow (false|true) "false"
          pagetemplate (undefined|breakcontinued|continued) "undefined"
          newpage (false|true) "false">

<!ELEMENT signalize (#PCDATA)>

<!ELEMENT safinstpara (#PCDATA|%inline-elements;|indexmarker|crossref|
          %meta-elements;)*>

<!ELEMENT tip (symbol,tippara+)>
<!ATTLIST tip
          symbol (%infosymbols;) "info"
          correctspacebelow (false|true) "false"
          pagetemplate (undefined|breakcontinued|continued) "undefined"
          newpage (false|true) "false">

<!ELEMENT tippara (#PCDATA|%inline-elements;|indexmarker|crossref|
          %meta-elements;)*>
```

```
<!ELEMENT symbol EMPTY>

<!ELEMENT list ((safinstsmall|tipsmall)*,item,(item | sublist)*)>
<!ATTLIST list
          type (numbered|bulleted) "numbered">

<!ELEMENT sublist ((safinstsmall|tipsmall)*,item+)>
<!ATTLIST sublist
          type (numbered|bulleted) "numbered">

<!ELEMENT item ((itempara|safinstsmall|tipsmall),
                (itempara|safinstsmall|tipsmall|figure|table|comment)*)>
<!ATTLIST item
          newpage (false|true) "false"
          pagetemplate (undefined|breakcontinued|continued) "undefined">

<!ELEMENT itempara (#PCDATA|%inline-elements;|indexmarker|crossref|
                    %meta-elements;)* >
<!ATTLIST itempara
          correctspacebelow (false|true) "false">

<!ELEMENT safinstsmall (symbolsmall,signalizesmall)>
<!ATTLIST safinstsmall
          symbol (%safetysymbols;) "general">
          correctspacebelow (false|true) "false">

<!ELEMENT signalizesmall (#PCDATA)>

<!ELEMENT tipsmall (symbolsmall,tiptextsmall)>
<!ATTLIST tipsmall
          symbol (%infosymbols;) "info"
          correctspacebelow (false|true) "false">

<!ELEMENT tiptextsmall (#PCDATA|%inline-elements;|indexmarker|crossref|
                        %meta-elements;)*>

<!-- element smallref: for highlighting references,
                       similar to a tipsmall with @symbol="reference"
                       but can occur outside of lists -->
<!ELEMENT smallref (smallreftext)>

<!ELEMENT smallreftext (#PCDATA|%inline-elements;|indexmarker|crossref|
                        %meta-elements;)*>

<!ELEMENT symbolsmall EMPTY>

<!ELEMENT figure (graphicanchor,graphic,caption?)>
<!ATTLIST figure
          leftborder (incolumn|includesideheadrange) "incolumn"
          newpage (false|true) "false"
```

```
          pagetemplate (undefined|breakcontinued|continued) "undefined"
          fmid    ID     #IMPLIED>

<!ELEMENT graphicanchor EMPTY>

<!ELEMENT graphic EMPTY>
<!ATTLIST graphic
          file       CDATA      #IMPLIED
          dpi        NMTOKEN    #IMPLIED
          rasterdpi  NMTOKEN   #IMPLIED
          alt        CDATA      #IMPLIED
          impsize    CDATA      #IMPLIED
          impby      (ref|copy) #IMPLIED
          sideways   NMTOKEN    #IMPLIED
          impang     CDATA      #IMPLIED
          xoffset    CDATA      #IMPLIED
          yoffset    CDATA      #IMPLIED
          position   NMTOKEN    #IMPLIED
          align      NMTOKEN    #IMPLIED
          cropped    NMTOKEN    #IMPLIED
          float      NMTOKEN    #IMPLIED
          width      CDATA      #IMPLIED
          height     CDATA      #IMPLIED
          angle      CDATA      #IMPLIED
          bloffset   CDATA      #IMPLIED
          nsoffset   CDATA      #IMPLIED>

<!ELEMENT caption (#PCDATA|%inline-elements;|indexmarker|
                   %meta-elements;)*>

<!-- =====  Element and attribute declarations for OASIS CALS-Tables
based on XML Exchange Table Model, modified to fit in Application,
all Entities but %yesorno; are removed for easy readability =====  -->

<!ELEMENT table (title?,tgroup+,caption?)>
<!ATTLIST table
          frame (top|bottom|topbot|all|sides|none) #IMPLIED
          colsep      %yesorno; #IMPLIED
          rowsep      %yesorno; #IMPLIED
          pgwide      %yesorno; #IMPLIED
          tabstyle    NMTOKEN   #IMPLIED
          orient (port|land) #IMPLIED
          align (left|right|center|justify|char) #IMPLIED
          charoff     NMTOKEN   #IMPLIED
          char        CDATA     #IMPLIED
          fmid        CDATA     #IMPLIED
          pagetemplate (undefined|breakcontinued|continued) "undefined"
          leftborder (incolumn|includesideheadrange) "incolumn">

<!ELEMENT tgroup (colspec*,spanspec*,thead?,tfoot?,tbody)>
```

```
<!ATTLIST tgroup
          cols         NMTOKEN   #REQUIRED
          colsep       %yesorno; #IMPLIED
          rowsep       %yesorno; #IMPLIED
          align (left|right|center|justify|char) #IMPLIED
          char         CDATA     #IMPLIED
          charoff      NMTOKEN   #IMPLIED
          tgroupstyle NMTOKEN   #IMPLIED>

<!ELEMENT colspec EMPTY >
<!ATTLIST colspec
          colnum       NMTOKEN   #IMPLIED
          colname      NMTOKEN   #IMPLIED
          colwidth     CDATA     #IMPLIED
          colsep       %yesorno; #IMPLIED
          rowsep       %yesorno; #IMPLIED
          align (left|right|center|justify|char) #IMPLIED
          char         CDATA     #IMPLIED
          charoff      NMTOKEN   #IMPLIED>

<!ELEMENT spanspec EMPTY>
<!ATTLIST spanspec
          spanname     NMTOKEN   #REQUIRED
          namest       NMTOKEN   #REQUIRED
          nameend      NMTOKEN   #REQUIRED
          colsep       %yesorno; #IMPLIED
          rowsep       %yesorno; #IMPLIED
          align (left|right|center|justify|char) #IMPLIED
          char         CDATA     #IMPLIED
          charoff      NMTOKEN   #IMPLIED>

<!ELEMENT thead (colspec*,row+)>
<!ATTLIST thead
          valign (top|middle|bottom) "bottom">

<!ELEMENT tfoot (colspec*,row+)>
<!ATTLIST tfoot
          valign (top|middle|bottom) "top">

<!ELEMENT tbody (row+)>
<!ATTLIST tbody
          valign (top|middle|bottom) "top">

<!ELEMENT row (entry+)>
<!ATTLIST row
          rowsep %yesorno; #IMPLIED
          valign (top|middle|bottom) "top">

<!ELEMENT entry (#PCDATA|%inline-elements;|indexmarker|crossref|
                 %meta-elements;|tablepara|tipsmall|safinstsmall|
```

```
                graphic|tablesymbol|list)*>
<!ATTLIST entry
          colname     NMTOKEN   #IMPLIED
          namest      NMTOKEN   #IMPLIED
          nameend     NMTOKEN   #IMPLIED
          morerows    NMTOKEN   #IMPLIED
          colsep      %yesorno; #IMPLIED
          rowsep      %yesorno; #IMPLIED
          align (left|right|center|justify|char) #IMPLIED
          char        CDATA     #IMPLIED
          charoff     NMTOKEN   #IMPLIED
          valign (top|middle|bottom) #IMPLIED
          rotate      %yesorno; #IMPLIED
          spanname    NMTOKEN   #IMPLIED>

<!ELEMENT title (#PCDATA|superscript|subscript|indexmarker|
                %meta-elements;)*>

<!-- === End of element and attribute declarations for CALS Tables ==
======== that are based on OASIS XML Exchange Table Model ======= -->

<!ELEMENT tablepara (#PCDATA|%inline-elements;|indexmarker|crossref|
                     %meta-elements;)*>

<!ELEMENT tablesymbol EMPTY>
<!ATTLIST tablesymbol
          symbol (%safetysymbols;|%infosymbols) "general"
          size (big|small) "big">

<!ELEMENT comment (#PCDATA|%inline-elements;)*>
<!ATTLIST comment
          fmid   ID   #IMPLIED>

<!ELEMENT emphasis (#PCDATA|superscript|subscript)*>

<!ELEMENT emphasisbold (#PCDATA|superscript|subscript)*>

<!ELEMENT superscript (#PCDATA|emphasis|emphasisbold)*>

<!ELEMENT subscript (#PCDATA|emphasis|emphasisbold)*>

<!ELEMENT crossref EMPTY>
<!ATTLIST crossref
          %crossref-attributes;>

<!ELEMENT indexmarker EMPTY>
<!ATTLIST indexmarker
          type       CDATA    #IMPLIED
          text       CDATA    #IMPLIED>
```

```
<!ELEMENT docutype EMPTY>

<!ELEMENT manufacturer EMPTY>

<!ELEMENT product EMPTY>

<!-- element prodtype: product-type -->
<!ELEMENT prodtype EMPTY>

<!ELEMENT appendix (appendixhead,appendixovw?,appendixtopic+,
                    emptypage?)>
<!ATTLIST appendix
          %meta-attributes;>

<!ELEMENT appendixhead EMPTY>
<!ATTLIST appendixhead
          fmid    ID    #IMPLIED>

<!-- element appendixovw: appendix-overview -->
<!ELEMENT appendixovw (appendixovwhead,appendixobj,appendixtoc)>

<!ELEMENT appendixovwhead EMPTY>

<!-- element appendixobj: appendix-object -->
<!ELEMENT appendixobj (appendixobjhead,appendixobjbody)>

<!ELEMENT appendixobjhead EMPTY>

<!ELEMENT appendixobjbody (para+)>

<!-- element appendixtoc: contains table of contents for appendix
                          as a list of crossreferences,
                          generated by XSLT -->
<!ELEMENT appendixtoc (appendixtochead?,appendixtocintro?,
                       appendixtocitem*)>

<!ELEMENT appendixtochead EMPTY>

<!ELEMENT appendixtocintro EMPTY>

<!ELEMENT appendixtocitem (appendixtocref)>

<!ELEMENT appendixtocref EMPTY>
<!ATTLIST appendixtocref
          %crossref-attributes;>

<!ELEMENT appendixtopic (head,(para|safinst|tip|
                         figure|table|list|comment|block)*)>
<!ATTLIST appendixtopic
          pagetemplate (nobreak|break) "nobreak">
```

```
<!-- element loa: list-of-abbreviations as table -->
<!ELEMENT loa (loahead,table,emptypage?)>
<!ATTLIST loa
          %meta-attributes;>

<!ELEMENT loahead EMPTY >
<!ATTLIST loahead
          fmid    ID    #IMPLIED>

<!ELEMENT index ANY>

<!-- element loc: list-of-comments for read proofs
                  as a list of crossreferences,
                  generated by XSLT -->
<!ELEMENT loc (lochead?,locitem*)>
<!ATTLIST loc
          %meta-attributes;>

<!ELEMENT lochead EMPTY>

<!ELEMENT locitem (#PCDATA|compageref)*>

<!ELEMENT compageref EMPTY >
<!ATTLIST compageref
          %crossref-attributes;>
```