

22 Word Open XML-Dateiformat

| | |
|--|----|
| Die Word-Dateiformate | 1 |
| Erstellung eines Open XML-Dokuments | 2 |
| Das OPC und dessen Bestandteile | 4 |
| Teile, Beziehungen und Inhaltstypen | 19 |
| Die programmtechnische Arbeit mit Open XML: eine Übersicht | 28 |
| Open XML-Dokumente & XSLT | 51 |
| Zusammenfassung | 54 |

In Office 2007 hat Microsoft neue Dateiformate für mehrere Office-Anwendungen eingeführt. Diese tragen die Bezeichnung »Office Open XML«, was oft auf »Open XML« oder »OOXML« abgekürzt wird. Das vorliegende Kapitel bietet einen Überblick zum neuen Dateiformat in Bezug auf Microsoft Word. Es baut dabei auf die Diskussion im Kapitel 21 auf. Falls Sie über keine XML-Vorkenntnisse verfügen, wäre es hilfreich, jenes Kapitel zuerst durchzulesen.

Die Word-Dateiformate

Bis einschließlich Word 2003 war die in Office 97 eingeführte binäre *.doc*-Datei das standardmäßige Dateiformat. Es stellt eine Momentaufnahme der von Word im Speicher gehaltenen Strukturen dar. Die Arbeit mit diesem Format gestaltet sich nicht einfach; Dritthersteller haben Schwierigkeiten aus einer Word-Datei Informationen zu lesen, geschweige denn Word-Dokumente zu erstellen, die erfolgreich in Word geöffnet werden können.

Hinweis Beginn

Lange Zeit war das binäre Dateiformat nicht öffentlich verfügbar. Der Entwickler musste bei Microsoft ein Gesuch stellen und ging gewisse Verpflichtungen ein. Seit Einführung von Office Open XML sind die Einzelheiten beider Formate offen zugänglich. Das binäre Format kann unter [http://msdn.microsoft.com/en-us/library/cc313105\(office.12\).aspx](http://msdn.microsoft.com/en-us/library/cc313105(office.12).aspx) heruntergeladen werden. Es gibt zudem ein Diskussionsforum auf MSDN unter http://social.msdn.microsoft.com/Forums/en-US/os_binaryfile/threads/.

Hinweis Ende

Wie im Kapitel 21 erwähnt, gibt es zwei Hauptgründe für den Wechsel zu einem XML-Dateiformat: verbesserte Wiederverwendung des Dokumentinhalts sowie erhöhte Zugänglichkeit. Ein mit XML strukturiertes Dokument kann auf mehrere Arten wiederverwendet werden. Zudem ist es für Dritthersteller um einiges einfacher geworden, Word-Dokument zu manipulieren bzw. herzustellen.

Die ersten Vorboten sahen wir in Office XP (2002) mit SpreadsheetML für Excel. In Office 2003 folgte WordProcessingML («Word 2003 XML») für Word. Obwohl beide Formate verbesserte Wiederverwendung sowie erhöhte Zugänglichkeit anboten, waren die XML-Vokabulare zu wenig umfangreich, um die alten Dateiformate zu ersetzen.

Microsoft wollte ein Dateiformat, welches

- das binäre als standardmäßiges Dateiformat ersetzt;
- die Konvertierung von Dokumenten im binären Format ohne Inhaltsverlust ermöglicht;
- robuster ist;
- von internationalen Standard-Organisationen anerkannt wird, beispielsweise ISO (Internationale Organisation für Normung).

Das Resultat ist das in Office 2007 eingeführte Office Open XML-Dateiformat.

Kasten Beginn

Office Open XML – das standardmäßige Dateiformat

In Office 2003 hat Microsoft entschieden, die bisherigen, binären Dateiformate als die standardmäßigen beizubehalten. Ihre Zeit (die über eine Dekade dauerte) ist mit der Veröffentlichung von Office 2007 zu Ende gegangen. Seither haben drei weitere Änderungen diesen Wechsel gefestigt:

- Die Standardisierung des Dateiformats durch ISO in 2008 (siehe auch den Kasten »Standardisierung der Dateiformate« auf Seite 10)
- Missbilligung des RTF-Dateiformats (bislang das für den Mensch lesbare Dateiformat für Word-Dokumente). Dieses Dateiformat wird nicht länger unterhalten oder ausgebaut.
- Einführung einer neuen, COM-basierten Konversionsschnittstelle für Textdateien (Word 2007 SP2). Diese benutzt Open XML statt RTF als das Zwischenformat und entfernt somit die letzte Abhängigkeit vom älteren Format.

Diese Open XML-Dateiformate sind die standardmäßigen Dateiformate für Office 2007, Mac Office 2008 und Office 2010. Zudem bietet Microsoft einen kostenlosen Satz Konvertierfilter zum Herunterladen an. Diese ermöglichen Benutzern von Word 2000, 2002 (XP), 2003, Mac Word v.X und Mac Word 2004 die Arbeit mit Open XML-Dokumenten. Damit können sie das neue Dateiformat öffnen, sowie binäre Dateien im neuen Dateiformat speichern.

Kasten Ende

Erstellung eines Open XML-Dokuments

Betrachten wir zuerst, wie ein Open XML Word-Dokument aufgebaut ist. Wir erstellen ein solches von Grund auf im Texteditor. Der erste Versuch ist eine Variation von Listing 21.17 aus Kapitel 21 (ohne Tabelle). Das Resultat aus Listing 22.1 ist ein leeres Dokument mit einer einzigen Absatzmarke. Es handelt sich hier um Word 2003 XML.

Listing 22.1: Inhalt von Bsp22_01.xml – Word 2003-WordProcessingML, das ein leeres Dokument definiert

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:wordDocument
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml">
  <w:body>
    <w:p/>
  </w:body>
</w:wordDocument>
```

Geben Sie diese Zeilen in einen Texteditor ein und speichern die Datei als **Bsp22_01.xml** ab. Das Resultat lässt sich in Word 2003, 2007 und 2010 als Word-Dokument öffnen. Das Dokument ist »leer«.

Wichtig Beginn

Denken Sie daran, dass die XML-Sprache auf die Großschreibung genau achtet!

Wichtig Ende

Das neuere Open XML hat mit seinem Vorgänger einige Gemeinsamkeiten. Viele der Elemente und Attribute bleiben gleich, wie beispielsweise `<w:p>` für einen Absatz (Paragraph), andere wurden jedoch geändert. Open XML benutzt beispielsweise `<w:document>` statt `<w:wordDocument>`.

Da Open XML ein neues XML-Vokabular eingeführt hat, benötigt es auch einen anderen XML-Namensraum URI.

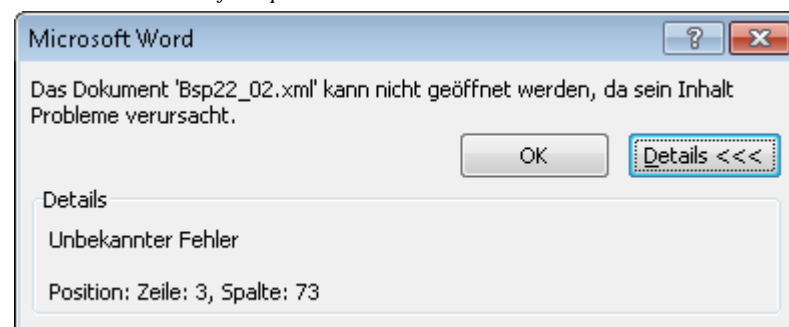
Probieren wir also zunächst, diese Änderungen im vorherigen Beispiel vorzunehmen (Listing 22.2). Wir speichern die Datei als **Bsp22_02.xml** und beobachten, wie sich das Dokument verhält.

Listing 22.2: Inhalt von Bsp22_02.xml – Ein leeres Dokument im WordProcessingML von Word 2003

```
<?xml version="1.0" encoding="UTF-8"?>
<w:document
  xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p/>
  </w:body>
</w:document>
```

Wird die Datei in Word 2007 oder 2010 geöffnet, erscheint eine Fehlermeldung: der Dateiinhalt wird nicht korrekt erkannt (Abbildung 22.1). Ob Word 2003 die Datei öffnen kann, kommt darauf an, ob das Kompatibilitätspack installiert ist. Bestenfalls öffnet Word die Datei als reine XML-Datei und zeigt die Elemente als Tags eines benutzerdefinierten Namensraums gehörend an.

Abbildung 22.1: Ein mit dem Namensraum für Open XML erstelltes XML-Dokument verursacht eine Fehlermeldung



Was fehlt?

Ein Open XML-Dokument besteht aus mehreren, in einem Paket (»Package«) zusammengefassten XML-Dateien (»Teilen«), statt einer einzelnen XML-Datei. Die Strukturen dieses Pakets müssen den Richtlinien und Regeln der »Open Packaging Conventions« (OPC) entsprechen.

Das OPC und dessen Bestandteile

Ein OPC-Paket enthält mehrere Teile (»Parts«). Einer dieser Teile enthält das Dokument-XML. Weitere Teile stellen Informationen über den Inhalt des Pakets oder wie die einzelnen Teile miteinander zusammenhängen bereit. Das XML für ein solches Paket ist in Listing 22.3 ersichtlich. Wenn Sie dieses Listing in einen Texteditor eingeben, die Datei als *Bsp22_03.xml* speichern und anschließend in Word 2007 oder 2010 öffnen, liegt ein »leeres« Dokument vor.

Listing 22.3: Inhalt von Bsp22_03.xml – Office Open XML für ein leeres Word 2007/2010-Dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<pkg:package
  xmlns:pkg="http://schemas.microsoft.com/office/2006/xmlPackage">

  <pkg:part
    pkg:name="/_rels/.rels"
    pkg:contentType="application/vnd.openxmlformats-package.relationships+xml">
    <pkg:xmlData>
      <Relationships
        xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
        <Relationship
          Id="rId1"
          Type=
"http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
          Target="bsp22/03.xml"/>
        </Relationships>
      </pkg:xmlData>
    </pkg:part>

    <pkg:part
      pkg:name="/bsp22/03.xml"
      pkg:contentType="application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml">
      <pkg:xmlData>
        <w:document
          xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
          <w:body>
            <w:p/>
          </w:body>
        </w:document>
      </pkg:xmlData>
    </pkg:part>
  </pkg:package>
```

Das »flache« OPC Format

Nehmen wir dieses Paket – das im so genannten »Flat OPC Format« erfasst ist – etwas genauer unter die Lupe.

Dieses Paket besteht aus zwei Teilen (Parts). Jeder Teil hat einen Namen (Name), der wie eine Pfadangabe aussieht. Der Zweite enthält das uns aus Listing 22.2 bekannte XML für ein Word-Dokument. Bezeichnen wir ihn als den Teil »Hauptdokument«. Der erste Teil definiert eine Beziehung (Relationship) zum zweiten Teil und hat einen Typ (Type) sowie ein Ziel (Target).

Der Typ der Beziehung ist `http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument` und ihr Ziel hat den gleichen Namen wie der Teil Hauptdokument (ohne das vorangehende »/«). Diese Tatsachen lassen darauf schließen, dass der erste Teil, die Beziehung, darauf hinweist, dass das Objekt `officeDocument` sich im Teil *bsp22/03.xml* (also im Teil Hauptdokument) befindet.

Bislang haben wir nichts gesehen, was darauf hindeutet, welche Art Dokument vorliegt. Die beschriebenen Informationen sind in allen Open XML-Dokumenten vorhanden. Was unterscheidet dieses von einem Excel- oder PowerPoint-Dokument? Darauf gibt es einige Hinweise:

- Der Teil Hauptdokument deklariert und setzt den Namensraum
`xmlns w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"`
- Zudem enthält jeder Teil ein Inhaltstyp (contentType). Für das zur Diskussion stehende Hauptdokument handelt es sich um

`application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml`

Diese URI legt fest, dass wir es mit einem Open XML Word-Dokument ohne Makros (*.docx*) zu tun haben. Dokumente mit Makros (*.docm*), sowie Vorlagen mit (*.dotm*) und ohne (*.dotx*) Makros haben eigene URIs für den Inhaltstyp.

Ein Open XML-Dokument umfasst meist mehr als nur zwei Teile. Fügen wir dem einfachen Dokument eine Kopfzeile zu und schauen was passiert.

Im Word 2003 XML wird die Informationen für die Kopfzeile innerhalb des Elements `wordDocument` verschachtelt. Im Open XML-Paket befindet sie sich in einem eigenen Teil, wie dies Listing 22.4 veranschaulicht.

Listing 22.4: Inhalt von Bsp22_04.xml – Ein Word 2007/2010 Open XML-Dokument mit einer Kopfzeile

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<pkg:package
  xmlns:pkg="http://schemas.microsoft.com/office/2006/xmlPackage">
  <pkg:part
    pkg:name="/_rels/.rels"
    pkg:contentType="application/vnd.openxmlformats-package.relationships+xml">
    <pkg:xmlData>
      <Relationships
        xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
        <Relationship Id="rId1"
          Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
          Target="word/document.xml" />
        </Relationships>
      </pkg:xmlData>
```

```

</pkg:part>
<pkg:part
  pkg:name="/word/document.xml"
  pkg:contentType="application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml">
  <pkg:xmlData>
    <w:document
      xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
      xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
      <w:body>
        <w:p />
        <w:sectPr>
          <w:headerReference w:type="default" r:id="rId1" />
        </w:sectPr>
      </w:body>
    </w:document>
  </pkg:xmlData>
</pkg:part>
<pkg:part
  pkg:name="/word/_rels/document.xml.rels"
  pkg:contentType="application/vnd.openxmlformats-package.relationships+xml">
  <pkg:xmlData>
    <Relationships
      xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
      <Relationship Id="rId1"
        Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/header"
        Target="header1.xml" />
    </Relationships>
  </pkg:xmlData>
</pkg:part>
<pkg:part
  pkg:name="/word/header1.xml"
  pkg:contentType="application/vnd.openxmlformats-officedocument.wordprocessingml.header+xml">
  <pkg:xmlData>
    <w:hdr xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
      <w:p>
        <w:r>
          <w:t>Kopfzeilentext</w:t>
        </w:r>
      </w:p>
    </w:hdr>
  </pkg:xmlData>
</pkg:part>
</pkg:package>

```

Dieses neue Paket hat vier, statt zwei Teile:

- Der erste Teil, /_rels/.rels, zeigt weiterhin auf den Teil Hauptdokument.
- Der zweite Teil ist immer noch das Hauptdokument, mit einigen Änderungen gegenüber zuvor:
 - Sein Name ist nun /word/document.xml.
 - Das Element <w:document> enthält einen Namensraum, der das

Namensraumpräfix `r` definiert.

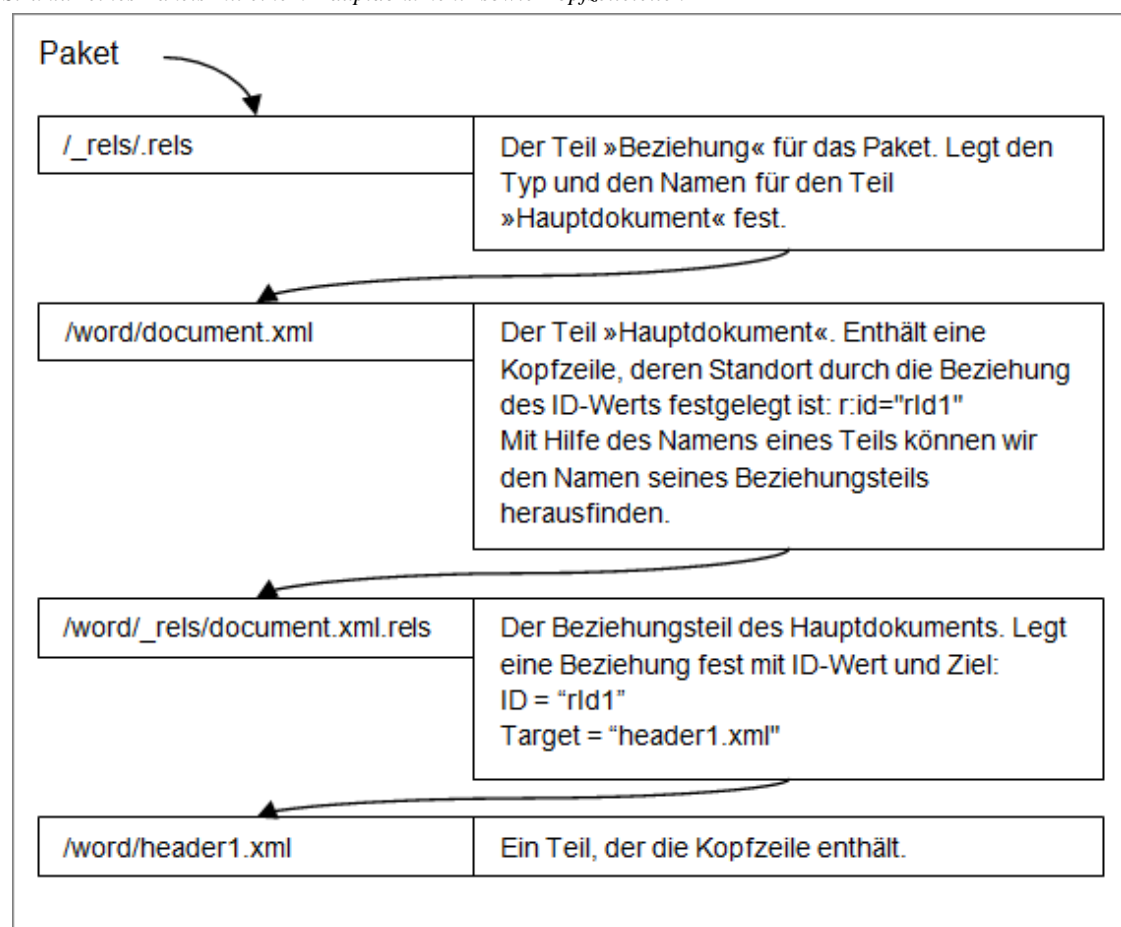
- Ein neues Element `<w:sectPr>` wurde zugefügt mit einem verschachtelten Element, das auf die Kopfzeile weist

```
<w:headerReference w:type="default" r:id="rId1" />
```

- Der dritte Teil namens `/word/_rels/document.xml.rels` ist neu und definiert eine Beziehung `rId1` mit dem Ziel *header1.xml*.
- Der vierte Teil `/word/header1.xml` ist ebenfalls neu und enthält das Element `<w:hdr>` mit dem XML für einen einzigen Absatz mit Text für die Kopfzeile.

Das ergibt die in Abbildung 22.2 ersichtliche Verkettung von Teilen und Beziehungen.

Abbildung 22.2: Die Struktur eines Pakets mit einem Hauptdokument- sowie Kopfzeileteilen



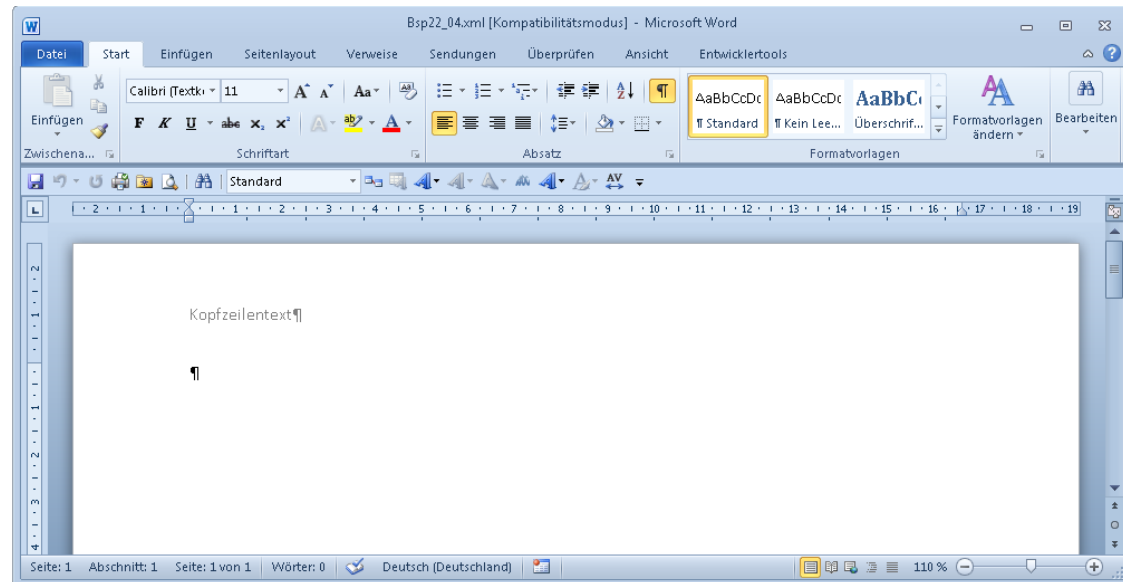
Hinweis Beginn

Falls Sie sich fragen, ob das mehrmalige Vorkommen des ID-Werts `rId1` innerhalb des Pakets zu Problemen führt, können wir Sie beruhigen. Paket-bezogene Beziehungen befinden sich immer im Beziehungsteil für das Paket, und solche für das Dokument im entsprechenden Teil für das Dokument.

Hinweis Ende

Wenn Sie dieses Beispiel im Texteditor eingeben, speichern und in Word 2007 oder Word 2010 öffnen sehen Sie ein Dokument mit dem Text »Kopfzeilentext« in der Kopfzeile (Abbildung 22.3).

Abbildung 22.3: Das Open XML-Dokument mit einer Kopfzeile



Das vollumfängliche OPC-Format

Bis hierher waren die Beispiele im flachen OPC XML-Format. Alle Informationen befinden sich in einer einzigen Datei; ein Element mit seinen Unterelementen definieren das Paket und seine Teile. Diese Sichtweise entspricht mehr oder weniger unserem gewöhnten »Alltag« und ist überschaubar, was den Einstieg in das Konzept des »echten« Open XML-Formats vereinfacht.

Laut der Open Package Convention besteht eine Datei im Open XML-Format richtigerweise aus einem Ordner, der als Paket dient, mit einer Datei für jeden Teil. Dieser Ordner muss dann in eine ZIP-Datei konvertiert und die Dateiendung vorzugsweise in (wie im Falls dieses Beispiels) *.docx* umbenannt werden.

Allzu weit davon entfernt sind wir mit dem letzten Beispiel nicht. Durch die Namen der Teile in Abbildung 22.2 zwingt sich die Struktur in Abbildung 22.4 förmlich auf.

Abbildung 22.4: Der Inhalt eines Open XML-Pakets, ausgelegt als Ordner und Dateien

| | |
|-----------------------|----------|
| Bsp22_04 | (Ordner) |
| └ _rels | (Ordner) |
| └└ .rels | (Datei) |
| └ word | (Ordner) |
| └└ _rels | (Ordner) |
| └└└ document.xml.rels | (Datei) |
| └└ document.xml | (Datei) |
| └└ header1.xml | (Datei) |

Sie können eine solche ZIP-Datei von Hand erstellen. Da aber kaum jemand

ernsthaft ein Open XML-Dokument auf diese Art erstellen wird, werden wir diese Struktur am Beispiel einer *docx*-Datei lediglich bestätigen.

1. Kopieren Sie die Datei *Kap22_04.docx* von der Buch-CD.
2. Führen Sie die Schritte 1 bis 4 im Kapitel 16, Abschnitt »Die Ribbon-Erweiterung in das Dokument einbinden« aus, um das Dokument in eine ZIP-Datei umzuwandeln. (Ersetzen Sie dabei in der Anleitung den Dateinamen *customUI* durch *Bsp22_04.docx*.)
3. Sie können die darin enthaltenen *.xml*- und *.rels*-Dateien im Texteditor öffnen und ihren Inhalt mit dem entsprechenden Teil in Listing 22.4 vergleichen. In *[word\header1.xml]*, beispielsweise, sollen Sie den folgenden XML-Code sehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<w:hdr xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:p>
    <w:r>
      <w:t>Kopfzeilentext</w:t>
    </w:r>
  </w:p>
</w:hdr>
```

Mit Ausnahme der XML-Deklaration in der ersten Zeile handelt es sich hier um das gleiche XML, das sich im Element *<pkgData>* von Listing 22.4 befindet.

4. Letztlich fällt eine zusätzliche Datei im Ordner der obersten Ebene auf: *[Content_Types].xml*. Diese enthält die Informationen des Inhaltstyps (*contentType*) aus jedem Teil des flachen OPC-XML-Pakets, mit einem zusätzlichen Eintrag, wie dies in Listing 22.5 ersichtlich ist.

Listing 22.5: Inhalt von [Content_Types].xml – Die Datei mit den Inhaltstypen im ZIP-Ordner Bsp22_04

```
<?xml version="1.0"?>
<Types
  xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
  <Default
    Extension="rels"
    ContentType="application/vnd.openxmlformats-package.relationships+xml"/>
  <Default
    Extension="xml"
    ContentType="application/xml"/>
  <Override
    PartName="/word/document.xml"
    ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml"/>
  <Override
    PartName="/word/header1.xml"
    ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.header+xml">
  </Types>
```

Somit liegt ein komplettes Open XML Word-Dokument vor. Es kann in Word 2007 oder Word 2010 und Word für Macintosh 2008 geöffnet werden, sowie in jeder weiteren Version von Word mit installiertem Konvertierungspack.

Kasten Beginn

Standardisierung der Dateiformate

Die »Open Packaging Conventions« (OPC) waren ursprünglich Teil der Office Open XML-Dateiformate, die im Jahr 2006 durch die ECMA als »ECMA-376« standardisiert wurden. (Dieser Standard ist ebenso als »ECMA-376 1st edition (2006)« bekannt.)

Eine neue Version des Standards wurde in 2008 durch die ISO als »ISO/IEC-29500:2008« abgesegnet. Eine aktualisierte Version des ECMA-Standards, das technologisch auf dem ISO-Standard ausgerichtet wurde, wurde als »ECMA-376 2nd edition (2008)« herausgegeben.

Die vier Hauptdokumente des ISO-Standards umfassen um die 7000 Seiten. Der Standard definiert »Übergangs« Bestimmungen, sodass Dokumente, die sich nach dem früheren Standard richten, als gültige »Übergangs«-Dokumente des ISO-Standards gelten. Vorgesehen ist, dass die Übergangsbestimmungen später entfernt werden.

Die Office 2007- sowie Mac 2008-Anwendungen (Word, Excel und PowerPoint) erstellen und erkennen Dokumente, das dem Standard »ECMA-376 1st edition (2006)« entsprechen.

Die Dokumente der Office 2010-Anwendungen sind eher auf der Linie des ISO-Standards.

Die ISO/IEC 29500:2008-Standarddokumente umfassen vier Hauptdokumente mit Anhängen, wie folgt:

- Part 1 (Fundamentals and Markup Language Reference)
Enthält Definitionen der Übereinstimmungen (»Conformance Definitions«) sowie Referenzmaterial, u.a. XML-Schemas für die XML-Dokument Markup Sprachen. Dieser Teil umfasst mehr als 5.500 Seiten.
- Part 2 (Open Packaging Convention)
(Wie in diesem Abschnitt vorgestellt.) Beschreibt das OPC, Kern-Eigenschaften und andere paketbezogene Punkte, wie die Handhabung digitaler Signaturen und Miniaturen. Es stellt die XML-Schemas für die OPC bereit. Dieser Teil umfasst mehr als 1000 Seiten.
- Part 3 (Markup Compatibility and Extensibility)
Beschreibt die Erweiterungsmöglichkeiten und umfasst ungefähr 40 Seiten.
- Part 4 (Transitional Migration Features)
Beschreibt Teile aus früheren Versionen, die aus Gründen der Rückwärtskompatibilität weiter unterstützt werden, wie VML (Grafiken). Es listet die Unterschiede zwischen »ISO/IEC 29500:2008« und »ECMA-376 1st edition« auf.

Diese Hauptdokumente für das ISO/IEC 29500:2008 Standard stehen unter <http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html> zur Verfügung.

Verwandte Standards inklusive einige Korrekturen für 2010 finden Sie unter http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54999. Diese sind jedoch *nicht* kostenlos.

Microsoft stellt Informationen zu den Office Open XML-Formate unter [http://msdn.microsoft.com/en-gb/library/cc313118\(offic.12\).aspx](http://msdn.microsoft.com/en-gb/library/cc313118(offic.12).aspx) bereit.

Vom besonderen Interesse ist das Microsoft-Dokument, das seine Erfüllung der Standards beschreibt: »[MS-OI29500]: Office Implementation Information for

ISO/IEC 29500 Standard Compliance«. Dieses Dokument können Sie unter [http://msdn.microsoft.com/en-us/library/ee908652\(office.12\).aspx](http://msdn.microsoft.com/en-us/library/ee908652(office.12).aspx) lesen.

Wo ist der passende Einstiegspunkt, um bei so vielen Informationen die Übersicht nicht zu verlieren? Wir empfehlen, »Section 8 – Overview« (Abschnitt 8 – Übersicht) der ISO/IEC 29500-1 »Fundamentals and Markup Language Reference« als Einstieg.

Anschließend ist es sinnvoll über das Inhaltsverzeichnis für die vier Hauptdokumente zu schweifen und Ausschau auf die Themen, die relevant für die gegenwärtig zu bewältigende Aufgabe sind, zu halten.

Obwohl diese Dokumentation zuerst überwältigend erscheint, enthält es eine überproportionale Menge an Erklärungstext und hilfreiche Beispiele. Sind Sie hauptsächlich an Word interessiert, reduziert sich der Fokus auf die Word-relevanten Teile.

Kasten Ende

Beispiel: Einen *customUI*-Teil in ein Dokument einbinden

Nun werden die bis hierher gewonnen Kenntnisse anhand eines einfachen Codebeispiels programmtechnisch umgesetzt. Im Kapitel 16 haben Sie erfahren, wie Schritt für Schritt eine Anpassung des Menübands durch Integration eines Ribbon XML-Teils im Dokumentpaket erfolgt. In diesem Abschnitt zeigen wir auf, wie dies programmtechnisch mit VBA vollzogen wird.

Das vorliegende Makro ist einfach und versucht nicht, alle möglichen Szenarien abzuhandeln. Es nimmt beispielsweise an, dass das Zieldokument weder einen *customUI.xml*-Teil noch eine entsprechende Beziehung enthält.

Was macht das Beispiel

Der VBA-Code dieses Beispiels führt folgende Handlungen aus, die den Schritten des Kapitels 16 entsprechen.

- packt die Datei *Bsp22_05rb.dotm* in einen Arbeitsordner aus,
- fügt die benötigten Beziehungen in den entsprechenden Teil für das Paket ein,
- kopiert *Bsp22_05ui.xml* in den korrekten Ordner der OPC-Struktur und nennt sie in *customUI.xml* um,
- packt den Ordner wieder in eine ZIP-Datei und ersetzt die ursprüngliche *.dotm*-Datei damit.

Diese Aufgaben werden ausgeführt, ohne das Word-Objektmodell zu bemühen; d.h. sie können genau so gut in Excel oder in einem klassischen VB-Projekt laufen. Die zwei Prozeduren, welche die Office Open XML-Datei aus- und einpackt werden in einem späteren Beispiel nochmals verwendet.

Hinweis Beginn

Eine ausführlichere Diskussion über die programmtechnische Arbeit mit dem Open XML-Format finden Sie weiter unten in diesem Kapitel.

Hinweis Ende

Das Beispiel ausführen

CD-ROM Beginn

Das Beispieldateien *Bsp22_05.docm*, *Bsp22_05rb.dotm*, und *Bsp22_05ui.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

CD-ROM Ende

1. Kopieren Sie alle vier Beispieldateien in einen Ordner auf Ihrer Festplatte mit der Pfadangabe *C:\Beispiele\Kap22*.
2. Öffnen Sie in Word die Datei *Bsp22_05.docm*.
3. Öffnen Sie den VB-Editor ((Alt)+(F11)) und blenden Sie das Modul Bsp22_05 ein.
4. Passen Sie – falls notwendig – die Pfadangaben an Ihr System an.
5. Führen Sie die Hauptprozedur aus.

Wurde die Prozedur erfolgreich ausgeführt, können Sie *Bsp22_05rb.dotm* in Word 2007 oder Word 2010 öffnen und die neue Registerkarte im Menüband sehen.

Es folgt eine Diskussion des Makro-Codes für die Bearbeitung des Open XML. Die Prozeduren der Hilfsmodule werden nicht behandelt.

Das Hauptmodul *Bsp22_05*

Am Anfang des Moduls Bsp22_05 sind **Konstanten** für folgende Informationen definiert (Listing 22.6):

- Pfadangaben und Dateinamen der Quelldateien sowie des Arbeitsordners. Diese können Ihren Bedürfnissen entsprechend angepasst werden.
- Die URI des Beziehungstyps für den Teil des Typs customUI
- Datename und Pfadangabe des Teils *customUI* innerhalb des Pakets.
- Eine ID und die URI des Namensraums für den Beziehungsteil XML
- Pfadangabe und Datename für den Teil Beziehungen des Pakets

Listing 22.6: Konstanten in Bsp22_05.docm

```
Option Explicit

Const sCustomUIFullName As String = _
    "c:\Beispiele\Kap22\Bsp22_05ui.xml"
Const sTargetDotmFullName As String = _
    "c:\Beispiele\Kap22\Bsp22_05rb.dotm"
Const sWorkPath As String = _
    "c:\Beispiele\Kap22\Bsp22_05"

Const CUSTOMUIREL As String = _
    "http://schemas.microsoft.com/office/2006/relationships/ui/extensibility"
Const CUSTOMUIPATH As String = _
    "/customUI/customUI.xml"
Const CUSTOMUIRELID As String = _
    "customUIRelId"
Const RELSNS As String = _
    "http://schemas.openxmlformats.org/package/2006/relationships"
Const RELPARTPATH As String = "_rels"
Const RELPARTEXTENSION As String = ".rels"
```

Hinweis Beginn

Bsp22_05ui.xml verweist auf die Schemas von Office 2007 und funktioniert somit für Word 2007 sowie für Word 2010. Falls Sie eine Datei erstellen möchten, welche die Menüband-Erweiterungen von Word 2010 nutzen soll, dann müssen die folgenden Anpassungen zum in Listing 22.6 enthaltenen Code vorgenommen werden.

```
Const sCustomUIFullName As String = _
    "c:\Beispiele\Kap22\Bsp22_05ui14.xml"
Const CUSTOMUIREL As String = _
    "http://schemas.microsoft.com/office/2007/relationships/ui/extensibility"
Const CUSTOMUIPATH As String = _
    "/customUI/customUI14.xml"
Const CUSTOMUIRELID As String = _
    "customUIRelId14"
```

Hinweis Ende

Die **Hauptprozedur** ist `addCustomUI` (Listing 22.7). Sie führt folgende Handlungen aus:

- Stellt sicher, dass die zwei Quelldateien vorliegen.
- Erstellt wenn nötig den Arbeitsordner.
- Packt das *.dotm* in den Arbeitsordner aus.
- Öffnet die Datei mit den Beziehungen für das Paket in einem `MSXML2.DOMDocument60` Objekt im Speicher.
- Fügt eine Beziehung für den Teil *customUI.xml* hinzu.
- Kopiert diesen Teil in das Paket.
- Packt den Ordnerinhalt wieder in eine ZIP-Datei und ersetzt die ursprüngliche *.dotm*.

Der Arbeitsordner wird *nicht* entfernt, um Ihnen Gelegenheit zu geben, den Inhalt zu inspizieren.

Tipp Beginn

Es gibt verschiedene Möglichkeiten, über MSXML einem XML-Dokument im Speicher XML-Elemente und -Attribute hinzu zu fügen. Etwas fremd für den Entwickler, der hauptsächlich mit Office-Objektmodellen arbeitet, ist die Erstellung eines neuen Knotenpunkts sowie das Hinzufügen dieses Elements in das XML-Dokument. Zwei Schritte sind erforderlich:

```
Set objXMLNode = _
    objXMLDoc.createElement(tagDOMNodeType.NODE_ELEMENT, "Relationship", RELSNS)
...
objXMLDoc.DocumentElement.appendChild objXMLNode
```

Zuerst wird der Knotenpunkt erstellt. Danach werden seine Eigenschaften festgelegt. Erst dann wird der Knotenpunkt dem XML-Dokument angefügt.

Bei Office-Objektmodellen sind wir uns gewohnt, dass ein Objekt zuerst in das Dokument eingefügt und nachträglich bearbeitet wird. Bei der Arbeit mit MSXML ist es umgekehrt.

Tipp Ende

Listing 22.7: *Bsp22_05* – `addCustomUI`

```

Sub addCustomUI()

    Dim bContinue As Boolean
    Dim objFSO As Scripting.FileSystemObject
    Dim objXMLANode As MSXML2.IXMLDOMNode
    Dim objXMLNode As MSXML2.IXMLDOMNode
    Dim objXMLDoc As MSXML2.DOMDocument60
    Dim strPackageRelsFullName As String

    bContinue = True
    Set objFSO = New FileSystemObject
    If Not objFSO.FileExists(sCustomUIFullName) Then
        bContinue = logError("Die Datei mit dem XML für das Menüband konnte " & _
            "nicht gefunden oder nicht geöffnet werden.")
    End If

    If bContinue Then
        If Not objFSO.FileExists(sTargetDotmFullName) Then
            bContinue = logError("Das Zieldokument konnte nicht gefunden oder " & _
                "nicht geöffnet werden.")
        End If
    End If

    If bContinue Then
        If Not objFSO.FolderExists(sWorkPath) Then
            objFSO.CreateFolder sWorkPath
        End If
        If Not objFSO.FolderExists(sWorkPath) Then
            bContinue = logError("Der Arbeitspfad konnte nicht gefunden oder nicht " & _
                "erstellt werden.")
        End If
    End If

    If bContinue Then
        bContinue = logPack(unpack(sTargetDotmFullName, sWorkPath))
    End If

    If bContinue Then
        strPackageRelsFullName = _
            objFSO.BuildPath(sWorkPath, _
                objFSO.BuildPath(RELPARTPATH, RELPARTEXTENSION))
        If objFSO.FileExists(strPackageRelsFullName) Then
            Set objXMLDoc = New MSXML2.DOMDocument60
            objXMLDoc.validateOnParse = True
            If objXMLDoc.Load(strPackageRelsFullName) Then
                Set objXMLNode = _
                    objXMLDoc.createNode(tagDOMNodeType.NODE_ELEMENT, "Relationship", RELSNS)
                With objXMLNode
                    Set objXMLANode = _
                        objXMLDoc.createNode(tagDOMNodeType.NODE_ATTRIBUTE, "Type", "")
                    objXMLANode.NodeValue = CUSTOMUIREL
                    .Attributes.setNamedItem objXMLANode
                    Set objXMLANode = _

```

```

        objXMLDoc.createElement(tagDOMNodeType.NODE_ATTRIBUTE, "Target", "")
        objXMLNode.NodeValue = CUSTOMUIPATH
        .Attributes.setNamedItem objXMLNode
    Set objXMLANode = _
        objXMLDoc.createElement(tagDOMNodeType.NODE_ATTRIBUTE, "Id", "")
        objXMLNode.NodeValue = "customUIRelID"
        .Attributes.setNamedItem objXMLNode
    Set objXMLANode = Nothing
End With
objXMLDoc.DocumentElement.appendChild objXMLENode
Set objXMLENode = Nothing
objXMLDoc.Save strPackageRelsFullName
Else
    bContinue = logError("Der Beziehungsteil des Pakets konnte nicht geöffnet werden.")
End If
Set objXMLDoc = Nothing
Else
    bContinue = logError("Der Beziehungsteil des Pakets konnte nicht gefunden werden.")
End If
createPath _
    objFSO.GetParentFolderName( _
        objFSO.BuildPath(sWorkPath, CUSTOMUIPATH)), _
    objFSO
objFSO.CopyFile _
    sCustomUIFullName, _
    objFSO.BuildPath(sWorkPath, CUSTOMUIPATH), _
    overwritefiles:=True
bContinue = logPack(pack(sWorkPath, sTargetDotmFullName))
End If
End Sub

Function createPath( _
    strPath As String, _
    objFSO As scripting.FileSystemObject _
) As Boolean
    createPath = False
    If Not objFSO.FolderExists(strPath) Then
        If createPath(objFSO.GetParentFolderName(strPath), objFSO) Then
            createPath = True
            Call objFSO.createFolder(strPath)
        End If
    Else
        createPath = True
    End If
End Function

```

Das Modul **modPackUnpack**

Dieses Modul enthält Prozeduren für das aus- sowie einpacken eines Open XML-Pakets (ZIP-Datei).

Solche Werkzeuge muss der VBA-Entwickler selber erstellen, weil VBA dafür keine Methoden zur Verfügung stellt. Deren Entwicklung gestaltet sich schwierig, weil weder die Windows API noch sonst irgendwelche Microsoft COM-Objekte mit

Automatisierungsschnittstelle die Fähigkeit bieten, mit ZIP-Dateien arbeiten zu können.

Im Modul wird die einfache Dateiverwaltung mit dem Scripting File System-Objekt (FSO) ausgeführt. Um Dateien in und aus einem ZIP-Paket zu kopieren, bedarf es des `Shell32`-Objekts. Beide Objekte gehören zum Lieferumfang von Windows XP und später. Die vorliegende Methode hat jedoch einige Nachteile. Das Kopieren von Dateien in das ZIP-Paket mit dem `Shell32`-Objekt erfolgt asynchron. Es ist deshalb notwendig, eine Methode auszuarbeiten, die ermittelt, ab wann dieser Vorgang beendet wurde.

Es handelt sich hier also um ein Konzeptbeispiel, nicht um Produktionscode. Entsprechend haben wir einige Vorgänge der Klarheit halber vereinfacht:

- Der ganze Inhalt des Open XML-Dokuments wird ausgepackt, bearbeitet und wieder eingepackt. Das Resultat ersetzt dann das ursprüngliche Dokument.
- Wird ein Ordner erstellt, nimmt der Code an, dass der nächst höhere Ordner in der Hierarchie bereits existiert.

Um kommerzielle Anwendungen zu erstellen müsste der Code um einiges robuster programmiert werden.

Die Funktion **unpack**

Die Funktion `unpack` (Listing 22.8) packt die im Parameter `strPack` bestimmte Datei in den im Parameter `strFolder` angegebenen Ordner aus. Sie gibt einen der `pckError`-Enumwerte zurück. Diese Prozedur

- verifiziert, dass die in `strPack` angegebene Datei tatsächlich existiert,
- fügt derem Namen die Dateiendung **.zip** zu, nachdem eine allfällig vorhandene Datei gleichen Namens gelöscht wurde,
- löscht einen allfällig vorhandenen Ordner mit dem in `strFolder` angegebenen Namen und erstellt einen neuen,
- kopiert den Inhalt der ZIP-Datei in den neuen Ordner,
- entfernt die Dateiendung **zip** vom Dateinamen des Ursprungdokuments.

Tipp Beginn

Die Methode `Copyhere` des `Shell32`-Objekts hat ein Parameter `CopyOptions`. Diese legt das gewünschte Verhalten für Meldungen fest. Dies können Sie ihren Wünschen anpassen. Die folgenden Werten dürfen (wie für die in Kapitel 2 vorgestellte `MsgBox`-Funktion) in beliebiger Zusammenstellung zusammen addiert werden, um mehrere der Optionen zu aktivieren

| | |
|------|---|
| 4 | Kein »Progress«-Dialogfeld einblenden |
| 16 | Automatisch »Yes to all« antworten auf jedes eingeblendete Dialogfeld |
| 512 | Das Erstellen eines neuen Ordners nicht mit einer Meldung bestätigen |
| 1024 | Keine Meldung einblenden, falls ein Fehler vorkommt |

Tipp Ende

Listing 22.8: *Bsp22_05.docm: modPackUnpack – gemeinsame Strukturen sowie die Funktion unpack*

```
Option Explicit
Public Enum pckError
    pckErrorOther = -1
    pckErrorSuccess = 0
    pckErrorPackNotFound = 1
```



```
pckErrorFolderNotFound = 2
pckErrorItemNotFound = 3
End Enum

Declare Sub Sleep Lib "kernel32" _
    (ByVal dwMilliseconds As Long)

Function unpack( _
    strPack As String, _
    strFolder As String _
) As Integer

Const CopyOptions As Integer = 1556
Dim objFSO As Scripting.FileSystemObject
Dim objShell As Shell
Dim strTemp As String
On Error GoTo problem

Set objFSO = CreateObject("Scripting.FileSystemObject")
If objFSO.FileExists(strPack) Then
    strTemp = strPack & ".zip"
    If objFSO.FileExists(strTemp) Then
        objFSO.GetFile(strTemp).Delete
    End If
    Name strPack As strTemp

    If objFSO.FolderExists(strFolder) Then
        objFSO.GetFolder(strFolder).Delete
    End If
    objFSO.CreateFolder strFolder

    Set objShell = CreateObject("Shell.Application")
    objShell.Namespace(strFolder).CopyHere _
        objShell.Namespace(strTemp).Items, _
        CopyOptions
    Set objShell = Nothing

    Name strTemp As strPack
    unpack = pckError.pckErrorSuccess
Else
    unpack = pckError.pckErrorPackNotFound
End If
Set objFSO = Nothing
Exit Function

problem:
unpack = pckError.pckErrorOther
On Error Resume Next
If strTemp <> "" Then
    Name strTemp As strPack
End If
Set objShell = Nothing
Set objFSO = Nothing
```

```
Err.Clear
End Function
```

Die Funktion **pack**

Wie ihr Name verrät, packt die Funktion **pack** den Inhalt des im Parameter **strFolder** angegebenen Ordners in eine Datei mit dem im Parameter **strFile** festgelegten Namen ein. Sie gibt einen der **pckError** Enum-Werte zurück. Die Prozedur führt die folgenden Handlungen aus:

- Verifiziert, dass der in **strFolder** angegebene Ordner tatsächlich existiert.
- Fügt dem in **strDatei** angegebenen Namen die Endung **.zip** zu, und nachdem eine eventuell vorhandene Datei gleichen Namens gelöscht wurde, erstellt eine neue ZIP-Datei.
- Kopiert den Inhalt des Ordners **strFolder** in die ZIP-Datei und wartet, bis die Anzahl an Dateien im Ordner der obersten Ebene der Anzahl an Dateien im Ordner **strFolder** entspricht. Die Wartezeit wird auf jedem Fall nach Erreichen der durch **MaxWait** bestimmten Anzahl Millisekunden abgebrochen.
- Löscht eine eventuell vorhandene Datei mit dem in **strFile** angegebenen Namen und benennt die ZIP-Datei mit diesem Namen um.

Wichtig Beginn

Windows erkennt nicht jede Datei mit der Endung **.zip** als ZIP-Datei. Am Dateianfang muss eine entsprechende »Signatur« vorhanden sein. Hier wird dies mit den folgenden Codezeilen erzielt:

```
Open strTemp For Output As #1
Print #1, "PK" & Chr$(5) & Chr$(6) & String(18, 0)
Close #1
```

Wichtig Ende

Diese Funktion unterstützt ebenfalls den Parameter **CopyOptions**, welcher bereits im vorangegangenen Abschnitt vorgestellt wurde.

Listing 22.9: Bsp22_05.docm: modPackUnpack – die Funktion pack

```
Function pack( _
    strFolder As String, _
    strPack As String _
) As Integer

Const CopyOptions As Integer = 1556
Const EachWait As Long = 100 ' 100ms
Const MaxWait As Long = 10000 ' 10s
Dim lngWait As Long
Dim objShell As Shell
Dim objFSO As Scripting.FileSystemObject
Dim strTemp As String
On Error GoTo problem

Set objFSO = CreateObject("Scripting.FileSystemObject")
If objFSO.FolderExists(strFolder) Then
    strTemp = strPack & ".zip"
    If objFSO.FileExists(strTemp) Then
```

```

        objFSO.GetFile(strTemp).Delete
    End If
    Open strTemp For Output As #1
    Print #1, "PK" & Chr$(5) & Chr$(6) & String(18, 0)
    Close #1

    Set objShell = CreateObject("Shell.Application")
    objShell.Namespace(strTemp).CopyHere _
        objShell.Namespace(strFolder).Items, _
        CopyOptions

    On Error Resume Next
    lngWait = 0
    Do Until _
        ((objShell.Namespace(strTemp).Items.Count >= _
            objShell.Namespace(strFolder).Items.Count) _
        Or (lngWait > MaxWait))
        DoEvents
        Sleep EachWait
        lngWait = lngWait + EachWait
    Loop
    On Error GoTo 0
    Set objShell = Nothing

    If objFSO.FileExists(strPack) Then
        objFSO.GetFile(strPack).Delete
    End If
    Name strTemp As strPack
    pack = pckError.pckErrorSuccess
Else
    pack = pckError.pckErrorFolderNotFound
End If
Set objFSO = Nothing
Exit Function

problem:
pack = pckError.pckErrorOther
On Error Resume Next
If strTemp <> "" Then
    Name strTemp As strPack
End If
Set objShell = Nothing
Set objFSO = Nothing
Err.Clear
End Function

```

Teile, Beziehungen und Inhaltstypen

Durch die bisherigen Beispiele haben Sie einige Grundkenntnisse des Open XML-Formats gesammelt. Wie Sie sich vorstellen können, gestaltet sich der Aufbau eines »echten« Open XML-Dokuments meistens etwas komplexer. In diesem Abschnitt

stellen wir die Konzepte Teile, Inhaltstypen und Beziehungen eingehender vor. Somit haben Sie die nötigen Werkzeuge in der Hand, die Open XML-Dokumentation anzuwenden.

Teile und ihre Typen

Den Inhalt des Beispieldokuments *Bsp22_04.xml* im Abschnitt »Das »flache« OPC Format« ab Seite 5 bilden die beiden Teile Hauptdokument und Kopfzeile. Sogar ein neues, »leeres«, von Word erstelltes Dokument enthält einige Teile mehr, wie

```
/_rels/.rels  
/word/_rels/document.xml.rels  
/word/document.xml  
/word/theme/theme1.xml  
/word/settings.xml  
/word/fontTable.xml  
/word/webSettings.xml  
/docProps/app.xml  
/docProps/core.xml  
/word/styles.xml
```

Im Anbetracht dieser Liste ist anzunehmen, dass es viele verschiedene Teil-Typen gibt. Darin wird der Inhalt für Kopf- und Fußzeilen, Fußnoten, Kommentare, Grafiken usw. verwaltet.

Was den Open XML-Standard betrifft, stehen für Word-Dokumente eine große Anzahl Typen für Teile bereit. Diese können in drei verschiedenen Gruppen unterteilt werden:

- Word-spezifische (Tabelle 22.1)
- Allgemeine, die dem Open XML-Format aller Office Anwendungen gemeinsam sind (Textverarbeitung, Tabellenblätter, usw.) (Tabelle 22.2)
- Funktionalitätsspezifische, wie für Diagramme oder Zeichnungen (werden hier nicht aufgelistet)

Zudem verfügt Word über einige Typen, die nicht in den ISO/ECMA-Standards definiert sind. (Tabelle 22.3)

In diesen Tabellen gibt die Spalte »Anzahl« an, wieviele Teile eines bestimmten Typs sich im Paket eines Dokuments befinden dürfen. Einige Typen dürfen nur einmal vorkommen, während andere höchstens zweimal vorhanden sein können. Weitere Typen dürfen beliebig oft integriert werden. In diesem Fall werden die Teile fortlaufend nummeriert: *header1.xml*, *header2.xml* usw. Folgend einige Beispiele:

- Ein, und nur ein einziger Hauptdokumentteil ist vorgeschrieben (Anzahl = 1)
- 0 oder 1 Glossarteil ist erlaubt (Anzahl=0–1)
- 0, 1 oder 2 Kommentarteile (Anzahl=0–2). Einer dieser Teile enthält die Kommentare für das Hauptdokument. Der andere verwaltet Kommentare für das Glossar.
- 0 oder mehr Kopfzeilenteile dürfen vorhanden sein (Anzahl=mehrere)

Im Fall von Anzahl=0–2 gehört immer ein Teil dem Hauptdokument und der andere dem Glossar.

Jeder Teiltyp akzeptiert ein oder mehrere URI für Inhaltstypen (contentType). In

Teilen, die XML enthalten, werden ein oder mehrere URI für Namensräume definiert oder zumindest ein URI für einen Wurzelnamensraum beinhalten.

Hinweis Beginn

Diese URIs führen wir im Text nicht auf, Sie werden jedoch viele der häufig vorkommenden URIs in *Kap22_CT.txt* und in *Kap22_NS.txt* auf der CD-ROM zum Buch im Ordner *\Beilagen\XMLPackages* finden.

Hinweis Ende

Tabelle 22.1: Word-spezifische Teiltypen

| Typname | Anzahl | Beschreibung |
|---|---------|--|
| Alternative Format Import | mehrere | Diese Teile sind für das Importieren von nicht-Open XML-Dokumenten vorgesehen. Sie unterstützen die Konvertierung des Inhalts in das neue Format für die Verwendung im Zieldokument. Beispiel: Ein HTML-Fragment wird in das Dokument eingefügt. Word wird es vermutlich in WordProcessingML umwandeln, und gleichzeitig die ursprüngliche Information in einem eigenen Teil speichern. |
| Comments (Kommentare) | 0–2 | Die Kommentare für alle Kommentar-Referenzen im Haupt- oder Glossar-Dokument. |
| Document Settings (Dokumenteinstellungen) | 0–2 | Verwaltet verschiedene Informationen, wie die allgemeinen Einstellungen für Fußnoten, oder Datenquelleninformationen für den Seriendruck. |
| Endnotes (Endnoten) | 0–2 | Die Endnoten für alle Endnote-Referenzen im Haupt- oder Glossar-Dokument. |
| Font Table (Schriftart-Tabelle) | 0–2 | Informationen zu den benutzten Schriftarten (wird beispielsweise für der Substitution einer Schriftart, wenn auf der Maschine nicht vorhanden ist, verwendet). |
| Footer (Fußzeile) | mehrere | Je ein Fußzeilenteil für jeden Fußzeilentyp (erste Seite, ungerade Seite, gerade Seite) pro Dokumentabschnitt. Je ein Satz für das Hauptdokument sowie das Glossar-Dokument. |
| Footnotes (Fußnoten) | 0–2 | Alle Fußnoten des Haupt- oder des Glossar-Dokuments. |
| Glossary Document (Glossar-Dokument) | 0–1 | Ein Speicherplatz für im Hauptdokument referenzierte Dokumentfragmente. |
| Header (Kopfzeile) | mehrere | Je ein Kopfzeilenteil für jeden Kopfzeilentyp (erste Seite, ungerade Seite, gerade Seite) pro Dokumentabschnitt. Je ein Satz für das Hauptdokument sowie das Glossar-Dokument. |
| Main Document (Hauptdokument) | 1 | Der Textkörper des Word-Dokuments oder der Word-Dokumentvorlage. |
| Numbering Definitions (Definitionen der Nummerierungen) | 0–2 | Eine Definition für jedes benutzte Nummerierungs- bzw. Aufzählungsschema. |
| Style Definitions (Definitionen der Formatvorlagen) | 0–2 | Definitionen für benutzterdefinierte und Word-eigene Formatvorlagen. |
| Web Settings (Einstellungen für Internet-Dokumente) | 0–2 | Einstellungen für Webseitendokumente, wie beispielsweise Einstellungen für ein Frameset-Objekt |

Tabelle 22.2: Gemeinsame Office-Teiltypen

| Typname | Anzahl | Beschreibung |
|--|---------|--|
| Additional Characteristics (erweiterte Eigenschaften) | mehrere | Speicher für erweiterte Informationen (beispielsweise die Höchstzahl Spalten oder Zeilen eines Kalkulationsblatts.) |
| Audio | mehrere | Ein Teil, der Audio-Information speichert |
| Bibliography (Bibliographie) | 0–1 | Ein Teil, der Bibliographie-Informationen speichert |
| Content (Inhalt) | mehrere | Enthält jedes unterstützte XML-Vokabular. Aus den Standards ist zu entnehmen, dass dieser Teiltyp für Objekte wie OMath-XML ist. |
| Custom XML Data Storage (Speicher für Benutzerdefiniertes XML) | mehrere | Beliebiges XML. In Word werden diese als »Custom XML Parts« bezeichnet. |
| Custom XML Data Storage Properties (Eigenschaften des Custom XML Data Storage) | mehrere | Informationen zum Inhalt/Speicher eines Custom XML Storageteils. Meistens 1 pro Custom XML Storageteil. |
| Digital Signature Origin (Ursprung der digitalen Signatur) | 0–1 | Ein leerer Teil, der als Anfangspunkt für die Aufspürung von Informationen zu einer digitalen Signatur dient. |
| Digital Signature XML Signature | 0–1 | Ein Teil für die digitale Signatur des Pakets |
| Embedded Control Persistence (Speicher für Daten von eingebetteten Steuerelemente) | mehrere | Für ActiveX-Steuerelemente, die Daten im XML-Format speichern. Ein Teil pro Steuerelement mit diesem Bedürfnis. |
| Embedded Object (Eingebettetes Steuerelement) | mehrere | Ein beliebiges, eingebettetes OLE-Objekt. |
| Embedded Package (Eingebettetes Paket) | mehrere | Ein beliebiges Open XML-Paket, das im Dokument eingebettet wurde. Beispiel: eine <i>xlsx</i> -Excel-Arbeitsmappe. Das gesamte Paket, im ZIP-Format, wird im Word-Dokument-Paket gespeichert. |
| File Properties, Extended (Erweiterte Dokumenteigenschaften) | 0–1 | Erweiterte Eigenschaften sind Eigenschaften die spezifisch sind für den Typ des Open XML-Dokuments. Diese umfassen beispielsweise bei einem Word-Dokument den Namen der verbundenen Dokumentvorlage, statistische Informationen wie die Anzahl Seiten, Zeichen, Wörter und Absätze usw.. |
| File Properties, Core (Kern-Dokumenteigenschaften) | 0–1 | Kern Eigenschaften sind Eigenschaften, die allen Office-Anwendungen gemeinsam sind. Dazu gehören Author (Autor), Title (Titel) usw. |
| File Properties, Custom (Benutzerdefinierte Dokumenteigenschaften) | 0–1 | Benutzerdefinierte Eigenschaften |
| Font | mehrere | Ein Teil für jede Schriftart, die im Paket <i>eingebettet</i> |

| | | |
|---|---------|---|
| (Schriftart) | | ist. (Normalerweise ist dies nicht der Fall, die Schriftarten werden dynamisch geladen. Wird eine TrueType Schriftart benutzt, kann sie mit dem Dokument verteilt werden, indem die Information im Dokument eingebettet ist. Die Option dafür befindet sich in der Registerkarte <i>Speichern</i> der Word-Optionen. Die Dateigröße wird dadurch erheblich erhöht.) |
| Image (Grafik) | mehrere | Ein Teil pro Grafik |
| Printer Settings (Druckereinstellungen) | mehrere | Je nach Einstellungen könnte das Paket einen solchen Teil pro Dokumentabschnitt enthalten. |
| Thumbnail (Miniatur) | 0–1 | Ein Thumbnail des Dokuments |
| Video | mehrere | Eine Videodatei wie AVI oder MPEG |

Tabelle 22.3: Zusätzliche Microsoft Teil-Typen

| Typname | Anzahl | Beschreibung |
|--|---------|---|
| Attached Toolbars (Angehängte Symbolleisten) | 0–1 | Ein Teil mit binären Daten für die Unterstützung von benutzerdefinierten Symbolleisten. |
| Customizations (Anpassungen) | 0–2 | Tastenkombinationen und Anpassungen von Symbolleisten |
| CustomUI | 0–1 | Anpassungen des Menübands |
| CustomUI2 | 0–1 | Anpassungen des Menübands sowie der Backstage-Ansicht |
| Embedded Control Persistence Binary Data (Binäre Daten eines eingebetteten Steuerelements) | mehrere | Unterstützt (ActiveX) Steuerelemente, die, anstelle von XML, binäre Daten speichern müssen. Ein Teil pro Steuerelement, das binäre Daten speichert. |
| Mail Merge Recipient Data (Daten über Seriendruckempfänger) | 0–1 | Enthält die Listen der eingebundenen bzw. ausgeschlossenen Seriendruckempfänger. Der Teiltyp ist in der Dokumentation des Standards erwähnt, jedoch nicht dokumentiert. |
| Quick Access Toolbar Customizations (Anpassungen zur Symbolleiste für den Schnellzugriff) | 0–1 | Anpassungen zur Symbolleiste für den Schnellzugriff. |
| Styles With Effects (Formatvorlagen mit Effekten) | 0–2 | Speichert eine Kopie des Teils <i>Styles</i> (wird mit der Word 2010-Funktionalität für Effekterweiterungen benutzt) |

Mehr zum Thema Beziehungen

In diesem Abschnitt werden die Typen für Quellen und Ziele einer Beziehung behandelt. Wir beschreiben zudem die Unterschiede zwischen impliziten und expliziten Beziehungen. Letztlich werden erlaubte Beziehungen, Eigenschaften und das Handhaben der Beziehungen für Custom XML Parts vorgestellt.

Quellen und Ziele

Jede »Relationship« im Open XML umschreibt eine Beziehung zwischen zwei

Objekten: einer Quelle und einem Ziel. Die Quelle kann das Paket sein, oder ein Teil innerhalb des Pakets. Das Ziel darf ein Teil im Paket sein (eine »interne Quelle«) oder könnte auf etwas außerhalb des Pakets zeigen (eine »externe Quelle«).

Wir sahen in *Bsp22_04* beispielsweise Folgendes:

- Wie das Paket eine Beziehung definiert, die als Ziel den Teil Hauptdokument hat.
- Falls im Dokument eine Kopfzeile vorkommt, wird diese Beziehung im Teil Hauptdokument definiert, die als Ziel den Teil Kopfzeile (»header«) hat.

Jede Quelle, die eine Beziehung definieren muss, hat einen Teil Beziehungen (Relationships). Dieser enthält alle Beziehungen für den Teil. Umgekehrt ist das nicht der Fall: ein Ziel führt keine Informationen mit, die Auskunft darüber gibt, welcher Teil seine Quelle ist. Es ist also allgemein gesehen einfacher, die Beziehungen eines Pakets von Quelle zum Ziel zu verfolgen, als herauszufinden, welcher Teil die Quelle eines Zielteils ist.

Das Auffinden eines Beziehungsteils gestaltet sich relativ einfach, weil sein Name auf dem des Quellteil basiert. Hat ein Teil den Namen */abc/def.ghi*, heißt der damit verbundenen Beziehungsteil */abc/_rels/def.ghi.rels*.

Wenn wir annehmen, das Paket selber den Namen */* hat, ist logischerweise der Name des Teils mit allen Beziehungen der obersten Ebene */_rels/.rels*.

Es müsste also möglich sein, durch alle Teile des Pakets zu navigieren in dem

1. der Beziehungsteil */_rels/.rels* geöffnet wird und
2. für jede interne Beziehung
 - den Zielnamen lesen
 - den Namen für den Beziehungsteil daraus zusammenstellen
 - falls der Beziehungsteil vorhanden ist, ihn öffnen und Schritt (2) ausführen

Bitte beachten Sie, dass dies alles ausgeführt wird, ohne einen Zielteil zu öffnen. Nur sein Beziehungsteil wird angesprochen (sofern vorhanden).

Die Mehrzahl der Beziehungstypen haben interne Ziele (Teile innerhalb des Pakets). Aus diesem Grund befinden sich die meisten Zieltypen in Tabelle 22.1 bis Tabelle 22.3. Weitere Beziehungstypen für externe Ziele sind in Tabelle 22.4 aufgelistet. Einige Beziehungstypen werden für interne sowie externe Beziehungen eingesetzt, wie etwa Audioteilen (bzw. -dateien), Hyperlinks, Grafiken oder Videoteilen (bzw. -dateien).

Tabelle 22.4: Externe Zieltypen

| Zieltyp | Bemerkungen |
|--|---|
| Attached Template (Angehängte Dokumentvorlage) | Darf auch im Teil <i>settings</i> definiert sein |
| FrameSet | Darf auch im Teil WebSettings definiert sein |
| SubDocument (Filialdokument) | Falls es sich um ein Master-Dokument handelt, werden die Beziehungen für die Subdokumente im Beziehungsteil des Hauptdokuments definiert. |
| Mail Merge Data Source (Seriendruckdatenquelle) | Darf auch im Teil <i>settings</i> definiert sein |
| Mail Merge Header Source (Seriendrucksteuersatz) | Darf auch im Teil <i>settings</i> definiert sein |
| XSL Transformation | Wird eine Transformation beim Speichern automatisch |

| |
|---|
| ausgeführt, wird sie im Teil <i>settings</i> definiert. |
|---|

Implizite und explizite Beziehungen

Es gibt zwei Arten Beziehung zwischen Quell- und Zielteilen: die explizite und die implizite.

Ist die Beziehung *explizit*, enthält das XML des Quellteils den Namen eines `r:id`-Werts, wie `rId5`. Der *settings*-Teil könnte beispielsweise eine Beziehung zu einer Seriendruckdatenquelle wie folgt definieren:

```
<w:mailMerge>
. . .
<w:dataSource r:id="rId5"/>
. . .
</w:mailMerge>
```

Im Beziehungsteil des Teil *settings* wird die Beziehung aufgesucht, deren ID `rId5` ist. Daraus werden den Beziehungstyp und das Ziel ermittelt.

Ist die Beziehung *implizit*, enthält der Quellteil kein `r:id`-Attribut für das Ziel. Die Referenz für eine Fußnote im Hauptdokumentteil wird beispielsweise durch eine »gewöhnliche« Id-Nummer für den Zielteil definiert:

```
<w:footnoteReference w:id="5"/>
```

Da alle Fußnotentexte in einem einzigen Teil aufgelistet werden, wäre es überflüssig, `r:id="rId3"` in jeder `footnoteReference`-Element einzugeben. (Und es kommt auch nie vor.)

Folgende Fragen stellen sich

1. Ist eine Beziehung zwischen der Quelle Hauptdokumentteil und dem Ziel Fußnotenteil wirklich notwendig?
Ja, weil diese den Namen des Fußnotenteils bereitstellt.
2. Enthält der Hauptdokumentteil kein Element mit einem Attribut für einen Id-Wert der Referenz (wie `r:id="rId3"`), wie kann die Beziehung im Beziehungsteil nachgeschlagen werden?

Ein Beziehungselement (`Relationship`) mit einem `Type`-Attribut wird gesucht, wie beispielsweise für Fußnoten:

```
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/footnotes"
```

Erlaubte Beziehungstypen und Typen-URIs

Die Open XML-Standards definieren, welche Arten von Beziehungen erlaubt, welche fakultativ und welche für eine bestimmte Kombination von Quellteil- und Zieltypen erforderlich sind. Erlaubte Beziehungen sind in Tabelle 22.5 aufgelistet.

Tabelle 22.5: Erlaubte Beziehungen

| Quelle(n) | Implizit/ Explizit | Ziel(e) |
|---|-----------------------|---|
| Package (Paket) | E | Main Document (erforderlich) Digital Signature Origin Properties (Core, Custom, Extended) |
| Comments (Kommentare) Endnotes, Footnotes | E | Alternative Format Import Content Various diagram parts |

| | | |
|---|---|--|
| (End-, Fußnoten) Header, Footer (Kopf-, Fußzeile) Glossary Document (Glossardokument) Main Document (Hauptdokument) | | Embedded Control Persistence Embedded Object Embedded Package Hyperlink Image Video |
| Glossary Document (Glossar- Document) Main Document (Hauptdokument) | I | Comments Document Settings Endnotes Font Table Footnotes Numbering Definitions Style Definitions Web Settings |
| Glossary Document (Glossar- Document) Main Document (Hauptdokument) | E | Footer Header Printer Settings |
| Main Document (Hauptdokument) | I | Additional Characteristics Bibliography Custom XML Data Glossary Document Theme Thumbnail |
| Main Document (Hauptdokument) | E | SubDocument |
| Font Table (Schriftart-Tabelle) | E | Font |
| Numbering Definitions (Definitionen für die Nummerierung) | E | Image |
| Settings (Einstellungen) | E | Document Template Mail Merge Data Source Mail Merge Header Source Recipient Data XSL Transformation |
| Web Settings (Einstellungen für Webseiten) | E | FrameSet |
| Additional Characteristics (zusätzliche Eigenschaften) Bibliography (Bibliographie) Custom XML Data (Benutzerdefinierte XML-Daten) | E | Custom XML Data Properties |
| Digital Signature Origin (Ursprung der digitalen Signatur) | E | Digital Signature XML-Signatur |
| Embedded Object (eingebettetes Objekt) Embedded Package (eingebettetes Paket) | E | Hyperlink |

Keine anderen Teiltypen können die Quelle einer Beziehung bilden (wie Alternative Format Import, Style Definitions, Audio, Custom XML Data Properties, Digital Signature XML Signature, Embedded Content Persistence, Properties (Core, Custom, Extended), Font, Image, Printer Settings, Thumbnail, Video, Hyperlink).

Beziehungen zu Teilen für Custom XML Parts (Daten)

Der Tabelle 22.5 ist zu entnehmen, dass nur ein Hauptdokumentteil ein Ziel des Typs »Custom XML (Data) Part« enthalten kann. Im Kapitel 7 wurde veranschaulicht, wie ein Custom XML Part mit einem Inhaltssteuerelement verknüpft wird. Inhaltssteuerelemente dürfen in Kopf- und Fußzeilen eingefügt werden. Eigentlich würden wir meinen, ein in der Kopfzeile eingefügtes Objekt müsste eine Referenz auf einen Teil haben, der außerhalb der Kopfzeile liegt. Folglich müsste ein Beziehungsteil für die Kopfzeile vorliegen, worin eine Beziehung zu finden ist, die einen Custom XML Part als Ziel hat. Aber laut Tabelle 22.5 ist dies nicht erlaubt. Wie wird es dann gemacht?

Datenverbindungen für Inhaltssteuerelemente sind ein Spezialfall. Kommt ein Inhaltssteuerelement vor, in irgendeinem Teil, der diese Objekte unterstützt (also Kopf-, Fußzeile, Fuß-, Endnote oder Hauptdokument), hat das Hauptdokument einen Beziehungsteil, der die Beziehung zum Custom XML Part definiert.

Der Teil, der das Inhaltssteuerelement mit Datenverknüpfung enthält, benutzt den Namensraum, einen Xpath-Ausdruck sowie ein StoreItemID-Wert, um den Custom XML Part zu referenzieren. Folgenden Codezeilen dienen als Beispiel:

```
w:prefixMappings="xmlns:ns0='http://www.beispiele.com/kap22'"
w:xpath="/ns0:Adresse[1]/ns0:Ort[1]"
w:storeItemID="{55AF091B-3C7A-41E3-B477-F2FDAA23CFDA}"
```

Der Namensraum und die Xpath-Angabe können in weiteren Custom XML Parts desselben Pakets vorkommen und sind somit nicht unbedingt eindeutig. Der StoreItemID-Wert ist der einzige, der garantiert eindeutig ist. Dieser ist jedoch im getrennten Custom XML Properties-Teil des Custom XML Parts gespeichert und ist daher keine direkte Referenzierung.

Hinweis Beginn

Weiteres zu Custom XML Parts lesen Sie im Abschnitt »Dokumenteigenschaften, Custom XML Parts und Inhaltssteuerelemente« ab Seite 32.

Hinweis Ende

Standards für die Benennungen von Teilen

Im Beispiel *Bsp22_03.xml* haben wir `/bsp22/03.xml` als der Name des Hauptdokumentteils eingesetzt. Würden Sie dieses Beispiel in Word öffnen, unter einem anderen Namen abspeichern und danach das XML des Dateiinhalts anschauen, würden Sie sehen, dass Word einen anderen Namen eingesetzt hat. Word benutzt für diesen Teil den im Open XML-Standard vorgesehenen Standardnamen `/word/document.xml`.

Word hat eine interne Liste von Ordner- und Dateinamen für jeden Teiltyp. Wie Sie jedoch gesehen haben, bleibt ein Dokument mit anderen Namen gültig. Es ist nicht auszuschließen, dass Ihr Code es einmal mit einem solchen Dokument zu tun haben wird, weshalb er entsprechend programmiert werden soll. Das Beispiel im Abschnitt »Beispiel: Seriendruck-Datenquelleninformationen abfragen und entfernen« ab Seite 32 bietet einige Hinweise, wie die Teile und Beziehungen eines Pakets durchzuschleifen ist, um festzustellen, wo sich etwas befindet und wie es in Wirklichkeit heißt.

Tipp Beginn

Um herauszufinden, welche Standardnamen Word benutzt, ist es am einfachsten, ein Beispiel in Words Benutzerschnittstelle zu erstellen, dieses zu speichern und anschließend die Ordnerstruktur zu analysieren.

Tipp Ende**Wichtig Beginn**

Eine wichtige Regel gilt es zu beachten. Ist das Ziel in einer Beziehung eine relative URL innerhalb des Pakets, ist diese Pfadangabe relativ zum Standort der *Quelle der Beziehung* und nicht relativ zur Datei *.rels* worin sich die Beziehung befindet.

Wichtig Ende

Die programmtechnische Arbeit mit Open XML: eine Übersicht

Vor der Einführung des Word 2003 XML-Dateiformats war die Automatisierung des Objektmodells die einzige zuverlässige Methode, Word-Dokumente zu erstellen oder zu bearbeiten. Ab Word 97 wurde dies mit jeder COM-fähigen Programmiersprache möglich, sei sie VBA, VB6, Delphi oder eine .NET-Sprache (über die Schnittstelle der »Interop Assemblies«).

Dank den XML-Dateiformaten von Word 2003 und später ist es nun möglich, Word-Dokumente ohne installierte Word-Anwendung zu erstellen und zu bearbeiten.

Nachfolgend stellen wir die Vor- und Nachteile der zwei Methoden gegenüber.

Das XML-Dateiformat

Einige Gründe sprechen für die direkte Manipulation einer Dokument-XML-Datei:

- Die Dokumente können durch eine beliebige Programmiersprache, (vorausgesetzt, sie verfügt über die notwendigen Werkzeuge) auf einem beliebigen Betriebssystem bearbeitet werden. XML-Dateien sind reine Textdateien. Open XML-Dateien sind gewöhnliche ZIP-Behälter.
- Die Word-Anwendung muss dabei nicht installiert sein.
- Zudem erfolgt die Ausführung der Bearbeitung schneller, als über das Objektmodell.

Diese Überlegungen sprechen für die direkte Bearbeitung des XML für Dokumente, die auf einem Server erstellt oder bearbeitet werden müssen. Die Word-Anwendung wurde nicht für die Ausführung in einer Server-Umgebung konzipiert. Wird sie so automatisiert, entstehen Probleme, sobald Word eine Antwort vom Benutzer verlangt. Ein Web-Server könnte beispielsweise Berichte erstellen, die auf Daten von einem SQL-Server basieren.

Das Word-Objektmodell

Auch die Arbeit mit dem Objektmodell hat ihre Vorteile, wie etwa

- Office-Anwendungen stellen einige Dienstleistungen zur Verfügung, die ohne

die Anwendung nur mit großer Mühe angeboten werden können. Ein wichtiges Beispiel ist das Ausdrucken eines Dokuments.

Weder Windows noch Crystal Reports oder sonst ein Entwicklertool bietet die Möglichkeit an, ein Word-Dokument ohne Hinzuziehen seiner Anwendung auszudrucken. Der Entwickler müsste mühsam jede Einzelheit der Word-Dokumentstruktur und -paginierung kennen und herausfinden, wie dies für die Ausgabe auf jedem möglichen Drucker zu codieren ist. Das Open XML des Dokuments müsste in Detail abgearbeitet und die Abbildung für den Ausdruck zusammengesetzt werden. Dieses Vorhaben wäre extrem kostspielig und zeitintensiv.

- Auch die direkte Bearbeitung eines Dokuments verlangt eine vertiefte Kenntnis der Word- sowie Open XML-Strukturen. Wird ein Absatz beispielsweise samt seiner Objekte durch die Word-Anwendung gelöscht, stellt Word sicher, dass alle Beziehungen und Referenzen nachgeführt werden. Diese Handlungen müssen bei der Bearbeitung mit Open XML explizit durch den Programmiercode ausgeführt werden.

Enthält der gelöschte Bereich beispielsweise eine Referenz zu einer Fußnote, muss den entsprechenden Teil geöffnet und auch diese Fußnote gelöscht werden.

Das Löschen einer Grafik gestaltet sich noch komplexer, da die Grafik unter Umständen sonst wo im Dokument eingesetzt werden könnte. Wenn nicht, muss die Beziehung und evtl. sein Teil gelöscht werden. Zudem müssten weitere Referenzen geprüft und angepasst werden.

Das Zusammenspiel der zwei Methoden

Obwohl die Bearbeitung eines Dokuments über Automatisierung der Word-Anwendung in eine Open XML-Datei resultiert, es ist nicht möglich, über Automatisierung die Paket- und XML-Strukturen genau zu bestimmen. Das Objektmodell bietet eine Dienstleistung an, die für die ausgeführten Handlungen Open XML schreibt – wie Word es für gut hält.

Als Beispiel, nehmen wir ein Logo, das im Paket gespeichert und später für den Gebrauch in Kopf- und Fußzeilen verwendet werden soll. Mit Open XML kann der Entwickler genau festlegen, wo die Grafik zu speichern und wie sie in den verschiedenen Teilen einzubinden und zu referenzieren ist.

Über das Objektmodell ist diese Feinabstimmung nicht möglich. Word entscheidet beim Einfügen, wo im Paket die Grafik gespeichert und wie sie in das Dokument eingebunden wird. Vielleicht wird es die Grafikdaten direkt in die Zeile mit dem XML der Fußzeile schreiben. Auch möglich wäre, dass Word die Grafik in einem getrennten Teil speichert, eine Beziehung erstellt, und in die Fußzeile eine Referenz dazu einfügt. Der Entwickler kann das nicht beeinflussen.

Nur eine eingehende Analyse des Projekts kann entscheiden, welche Methode besser zum Ziel führt.

Was *nicht* möglich ist, ist die gleichzeitige Bearbeitung eines Dokuments durch das Objektmodell, sowie durch Manipulieren des Open XML-Pakets. Die Datei wird beim Öffnen von Windows gesperrt, sodass nur der eine Prozess damit arbeiten kann. Falls Sie ein Dokument auf beide Arten bearbeiten wollen, müssen diese Handlungen nacheinander erfolgen. Das Dokument muss geschlossen und von Windows freigegeben werden, bevor die Bearbeitung mit der anderen Methode erfolgen kann.

Zugang zum Open XML durch das Objektmodell

Das Word-Objektmodell bietet einige Methoden, die den Zugriff auf den XML-Inhalt eines Dokuments sowie auf gewisse Teile ermöglichen:

- Custom XML Parts können zugefügt, bearbeitet und mit Inhaltssteuerelementen verknüpft werden (im Kapitel 7 beschrieben)
- Das Word 2003 XML für die aktuelle Markierung oder eines Dokumentbereichs kann einer Zeichenkette zugewiesen werden (die Eigenschaft XML des Range- bzw. Selection-Objekts)
- Das Open XML für die aktuelle Markierung oder eines Dokumentbereichs kann einer Zeichenkette zugewiesen werden (die Eigenschaft WordOpenXML des Range- bzw. Selection-Objekts)
- Der Inhalt eines bestimmten Bereichs kann als eine Datei jedes unterstützten Dateiformats (wie Word 2003 XML, flaches OPC oder Open XML) gespeichert werden (die Methode ExportFragment des Range-Objekts)
- Der Inhalt einer in einem unterstützten Format gespeicherten Datei kann importiert werden (die Methode ImportFragment des Range-Objekts)
- Inhalt in der Form von WordProcessingXML oder Open XML kann als Zeichenkette in ein Dokument eingefügt werden (die Methode InsertXML des Range- bzw. Selection-Objekts)

Diese Methode setzt voraus, dass der Bereich oder die Markierung eine zulässige Stelle ist. Die Methode verursacht eine Fehlermeldung, wenn beispielsweise der Zielbereich eine Grafik ist.

Beim Importieren oder Exportieren von XML in ein bzw. aus einem in Word geöffneten Dokument spielt es keine Rolle, ob das Dokument selber als *.doc*, *.docx*, *.rtf* oder überhaupt auf der Festplatte gespeichert wurde. Word hält eine Repräsentation des Dokuments im Speicher bereit, und benutzt diese Informationen für Umwandlung der Information und deren Einfügen bzw. für das Bereitstellen des XML.

Dabei behandelt es sich nicht um ein kleines XML-Fragment, wie `<w:t>abc</w:t>`, sondern um das benötigte XML für ein gültiges Word-Dokument, das als selbständige Datei in Word geöffnet werden könnte. Im Fall von Word-Open XML bedeutet dies, dass es in der Form des flachen OPC-Formats vorhanden ist (siehe den Abschnitt »Das »flache« OPC Format« ab Seite 5).

Des Weiteren verweigert Word das Einfügen über `InsertXML`, wenn das XML mit einem BOM (»Byte Order Mark«) anfängt. (Ein BOM trägt Informationen über Unicode.) Word kann eine Datei mit einem BOM problemlos öffnen, es meckert nur, wenn es in der Zeichenkette vorhanden ist, die über `InsertFile` in das Dokument eingefügt werden soll.

Werkzeuge für die direkte Bearbeitung eines Open XML-Pakets

Da Open XML-Pakete aus einem ZIP-Behälter mit reinen Textdateien bestehen, können sie mit beliebigen Programmiersprachen oder Systemen bearbeitet werden, die diese einfache Voraussetzungen unterstützen. Wie für die Arbeit mit XML, vereinfacht ein gutes Werkzeug die Aufgabe erheblich. Eine Programmiersprache

oder Bibliothek/Add-On soll die folgenden Voraussetzungen erfüllen. Sie muss ...

- mit ZIP-Dateien und deren Inhalt arbeiten können. Noch vorteilhafter ist die Fähigkeit, mit jeder Art Paket und dessen Teilen arbeiten zu können. Sie muss den standardmäßigen Umgang mit Ordnern und Dateien unterstützen, wie das Kopieren, Öffnen und Löschen.
- mit XML-Dateien arbeiten können. Welche Funktionalitäten benötigt werden, hängt jeweils vom einzelnen Projekt ab. Auf jedem Fall müssen XML-Dokumente geladen und gespeichert werden können, sowie
 - das Erstellen, Löschen und Navigieren einer Hierarchie von XML-Knotenpunkten, die Zählung von Knotenpunkten, sowie das Lesen deren Daten
 - die Verwendung von Xpath, um einzelne sowie Sammlungen von Knotenpunkten anzusprechen
 - den Einsatz von XSLT unterstützen, um das XML zu transformieren.

Für das Windows-Betriebssystem stellt .NET Framework die vollumfänglichste Umgebung für solche Arbeiten zur Verfügung. Es umfasst Bereiche für die Arbeit mit XML sowie (ab Version 3.0) mit Paketen. Die Klassen unter `System.IO.Packaging`, beispielsweise, bieten Werkzeuge für die allgemeine Arbeit mit ZIP-Dateien ab Windows XP SP3.

Zudem hat Microsoft ein Open XML SDK entwickelt, das eine strengere Typisierung des Zugangs zu Open XML-Paketen bereitstellt.

Viele Informationsquellen stehen für die Arbeit mit Open XML in .NET Framework zur Verfügung. (Was auch der Grund ist, warum die Beispiele in diesem Kapitel VBA sind – diese Dokumentation fehlt.)

Hinweis Beginn

Die Dokumentation für das Open XML Format SDK 2.0 (»Software Development Kit«) für Microsoft .NET Framework 3.5 in Zusammenarbeit mit Office 2007 sowie Office 2010 steht auf <http://msdn.microsoft.com/en-us/library/bb448854.aspx> bereit.

Die Dokumentation für das Open XML Format SDK 1.0 für Microsoft .NET Framework 3.0 in Zusammenarbeit mit Office 2007 steht auf [http://msdn.microsoft.com/en-us/library/bb448854\(office.12\).aspx](http://msdn.microsoft.com/en-us/library/bb448854(office.12).aspx) bereit.

Der Namensraum `System.IO.Packaging`, welcher mit ZIP-Dateien arbeitet ist Bestandteil von Microsoft .NET Framework 3.0 und später. Dokumentation für die Funktionalität von .NET Framework 4.0 sowie Links für frühere Versionen finden Sie auf <http://msdn.microsoft.com/en-us/library/system.io.packaging.aspx>.

Diskussionsforen finden Sie auf <http://www.OpenXMLDeveloper.org> sowie <http://social.msdn.microsoft.com/Forums/en-US/oxmlsdk/threads>.

Hinweis Ende

Word-VBA und die Arbeit mit Open XML

Nur weil VBA ein Teil von Word (und Office) ist, bedeutet das nicht, dass es auf der Arbeit mit dem eigenen Objektmodell begrenzt ist. VBA basiert auf der klassischen Visual Basic und ist eine allgemein einsetzbare, prozedurale Programmiersprache. Es kann mit Dateien, Ordnern und sogar der Windows API umgehen, wie dies in mehreren Kapiteln dieses Buchs beschrieben ist. Damit kann auch mit Open XML-Dateien gearbeitet werden.

Einzigster Stolperstein ist die Tatsache, dass Microsoft keine entsprechenden Werkzeuge in der Sprache eingebaut hat. Die benötigte Bibliothek muss der Entwickler in das Projekt einbinden und allenfalls ergänzen.

Hinweis Beginn

Wie ein Verweis auf eine Bibliothek gesetzt wird, wurde in Kapitel 9 beschrieben.

Hinweis Ende

Für die Arbeit mit XML stellt Microsoft die Bibliothek MSXML zur Verfügung. Diese bietet Funktionalität für das Laden, Speichern und Navigieren von XML-Dokumenten und unterstützt Xpath Version 1.0 sowie XSLT Version 1.0. Die Beispiele in diesem Kapitel wurden mit MSXML Version 6.0 erstellt.

Leider gibt es kein Gegenstück für die Arbeit mit ZIP-Paketen. In Microsoft-Produkten wird ein solches Werkzeug einzig in .NET Framework angeboten. Das Betriebssystem Windows 7 wird mit einer Packaging API ausgeliefert, welches Windows Vista ebenfalls zur Verfügung steht, wenn das »Platform Update for Windows Vista« installiert wurde. Leider stellt diese COM-Bibliothek keine Automatisierungsschnittstelle zur Verfügung, die einen Einsatz mittels VBA ermöglichen würde.

Deshalb machen die VBA-Beispiele in diesem Kapitel vom Windows Scripting-Objekt und Windows Shell-Objekt Gebrauch, um

- eine Open XML-ZIP-Paket zu öffnen, analog wie eine *.docx*-Datei im Windows-Explorer als ZIP-Datei geöffnet werden kann
- einen Ordner mit seinen Dateien in eine Open XML-ZIP-Paket (*.docx*) zu packen

Hinweis Beginn

Falls Sie vorhaben, intensiv mit dem Open XML-Format zu programmieren, wäre es empfehlenswert, in bessere Werkzeuge für die Arbeit mit ZIP-Dateien zu investieren. Verschiedene Hersteller von ZIP-Software für den Benutzer verkaufen APIs für ihre Anwendungen, wie beispielsweise WinZip.

Hinweis Ende

Die Beispiele in den folgenden Abschnitten veranschaulichen, wie mit VBA Open XML-Dateien direkt bearbeitet werden können.

Beispiel: Seriendruck-Datenquelleninformationen abfragen und entfernen

Ein Seriendruckhauptdokument ist im Prinzip eine Vorlage, woraus mehrfache Kopien erstellt werden, mit variierendem Inhalt aus einer Datenquelle. Das klassische Beispiel ist ein Standardbrief, der an mehrere Empfänger gesandt wird.

Viele Organisationen benutzen die Seriendruckfunktionalität für die verschiedensten Aufgaben. Mit der Zeit sammelt sich eine Menge Seriendruckhauptdokumente an, in Benutzer- sowie freigegebenen, geteilten Ordner.

Ein großes Problem für die Verwaltung dieser Dokumente stellt die Verbindung zur Datenquelle dar. Die genauen Verbindungsinformationen stehen nur dann zur Verfügung, wann das Dokument erfolgreich geöffnet werden kann. Ist die Datenquelle nicht verfügbar, werden die Datenquelleninformationen beim Öffnen des Dokuments durch Word entfernt.

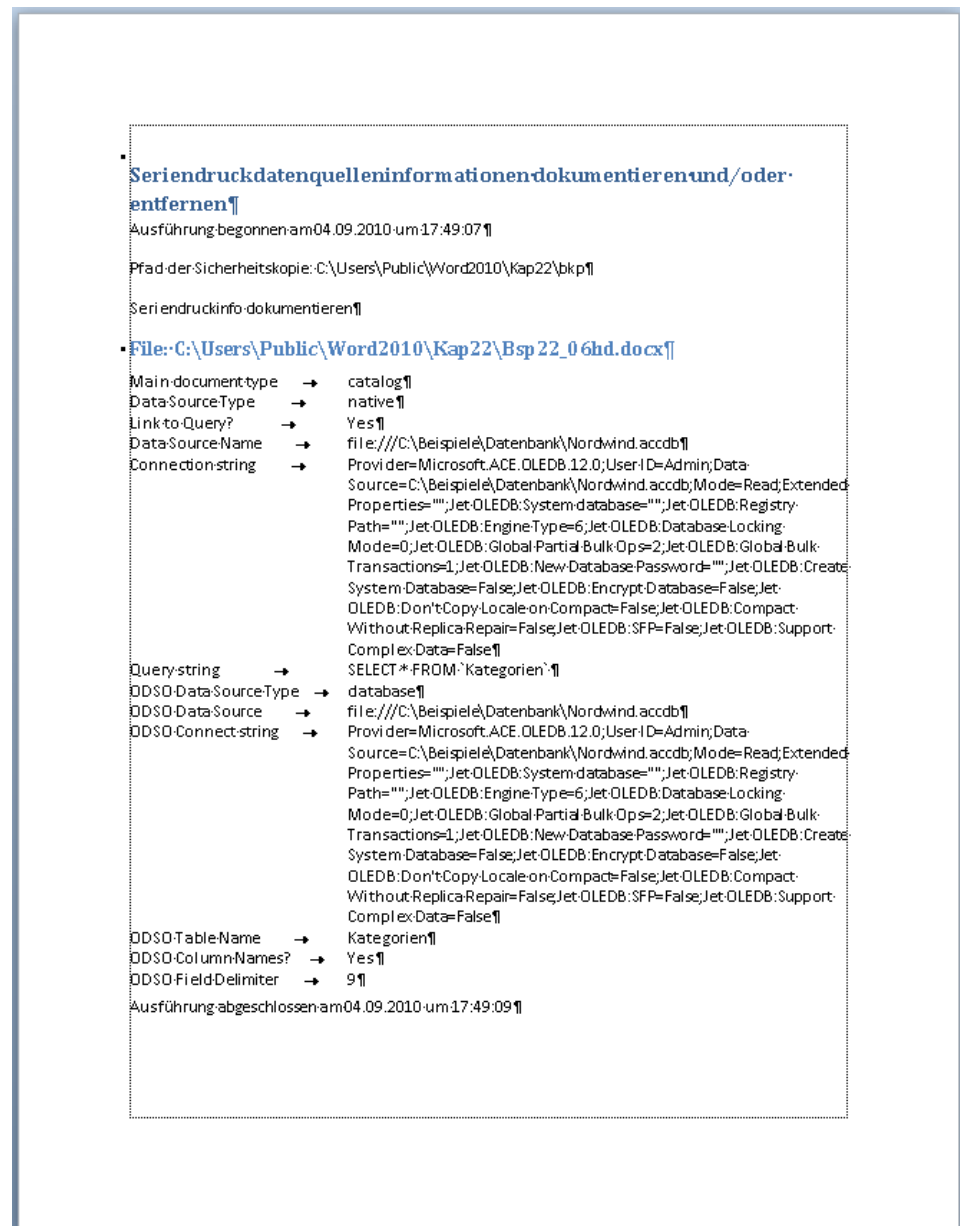
Des Weiteren wird die Datenquelle automatisch aus dem

Seriendruckhauptdokument entfernt, wenn es programmtechnisch (statt durch den Benutzer) geöffnet wird. Der Code muss die gleiche Datenquelle wieder anhängen, und dafür braucht es ebenfalls diese Informationen.

Das vorliegende Beispiel zeigt eine mögliche Lösung auf, indem es die Datenquelleninformationen aus dem Open XML-Dokument liest und dokumentiert (Abbildung 22.6). Es zeigt auf

- die Verfolgung einer Verkettung von Beziehungen und Teilen innerhalb eines Pakets
- das extrahieren von Daten aus einem Paket
- das Entfernen von Teilen und die Aktualisierung zwischen Beziehungsteilen und dem Inhalt von *[Content_Types].xml*.

Abbildung 22.5: Bericht über die Seriendruckdatenquelleninformationen im Seriendruckhauptdokument



Wozu dient das Beispiel?

Neben der Dokumentation ermöglicht dieses Beispiel das Entfernen der Datenquelle aus dem Seriendruckhauptdokument. Der Code führt folgende Handlungen aus. Er

- macht eine Sicherheitskopie der ausgewählten *.docx*-Datei und speichert sie in einem dafür bestimmten Ordner
- packt diese *.docx*-Datei in einen Unterordner dieses Ordners aus
- navigiert durch die Strukturen der Teile und Beziehungen, bis er den Teil *settings* aufspürt, worin der Hauptteil der Datenquelleninformationen sich

befindet

- falls diese Informationen gefunden wurden
 - sucht er den entsprechenden Teil mit den Beziehungen für den Teil *settings.xml*. Dieser könnte mehrere Beziehungen für den Seriendruck enthalten, die auf Teile für die externe Datenquelle und eventuell für eine Steuersatzdatei zeigen. Möglich ist auch eine Beziehung auf einen internen Teil *recipientData* (enthält Informationen über welche Empfänger der Benutzer ausgegrenzt oder eingebunden hat).
 - kann fakultativ die Dokumentation in das gegenwärtig aktives Dokument ausgeben
 - kann fakultativ die Seriendruckinformationen aus dem Teil *settings*, aus den entsprechenden Beziehungen sowie aus dem Teil *recipientData* entfernen. Falls im Teil mit den Beziehungen für den Teil *settings* keine Beziehungen mehr vorliegen, wird auch den Teil für die Beziehungen gelöscht. Der Teil *[Content_Types].xml* wird entsprechend angepasst.
- Packt das geänderte Dokument wieder ein und ersetzt das ursprüngliche Dokument.

Das Beispiel ausführen

CD-ROM Beginn

Das Beispieldokument *Bsp22_06.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*. Sie brauchen zudem ein Seriendruckhauptdokument. Sie können ein eigenes Dokument oder die Datei *Bsp22_06hd.docx* im gleichen Ordner auf der CD-ROM verwenden. Dieses ist mit der Datenbank *Nordwind2007.accdb* als Datenquelle verbunden, die sich im Ordner *C:\Beispiele\Datenbank* befinden soll, wenn das Dokument ohne Verlust der Datenquelle geöffnet werden soll.

CD-ROM Ende

1. Öffnen Sie das Dokument *Bsp22_06.docm*.
2. Öffnen Sie den VBA-Editor ((Alt)+(F 11)) und wechseln zum Modul *Bsp22_06*. (Dieses Modul ist der Einstiegspunkt für die Funktionalität und wird hier nicht wiedergegeben.)
3. Passen Sie die Pfadangaben in der Prozedur für Ihr System an und führen sie aus. Ein Log sowie die Dokumentation zum Seriendruck wird in das aktuelle Dokument (*Bsp22_06.docm*) geschrieben. Falls Sie das Beispiel nochmals von Vorne ausführen möchten, können Sie diesen Text einfach löschen.

Die weiteren Module, die im Folgenden vorgestellt werden, sind:

- *Bsp22_06_Main* enthält Prozeduren für
 - die Erstellung der Sicherheitskopie der angegebenen Datei
 - das Auspacken dieser Datei
 - das Navigieren und das Bearbeiten der Datei
- *clsContentTypesPart* ist ein Klassenmodul mit Funktionen für die Verwaltung des Teils *[Content_Types].xml*.
- *modLog* enthält einige Hilfsfunktionen und -methoden für das Loggen von Informationen und Fehlern und wird hier nicht detailliert aufgeführt.

- modPackUnpack: beachten Sie dazu die Beschreibungen im Abschnitt »Beispiel: Einen *customUI*-Teil in ein Dokument einbinden« ab Seite 11.

Das Hauptmodul: Bsp22_06

Deklaration der Konstanten

Dieser Abschnitt hält die URIs für die Beziehungsteile, sowie Text für Logs und Fehlermeldungen fest.

Zudem befinden sich Speicherort und Dateiname des Beziehungsteils für das Paket hier.

Letztlich wird eine globale Variable deklariert, der ein ContentTypes-Objekt zugewiesen wird. Dieses wird durch mehrere Prozeduren eingesetzt.

Listing 22.10: Hauptmodul in Bsp22_06.docm – die Konstante

```
Option Explicit

Const PACKAGEDESC As String = _
    "Package"
Const DATASOURCEREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/mailMergeSource"
Const DATASOURCEDESC As String = _
    "Data Source"
Const ODSODATASOURCEDESC As String = _
    "Data Source (ODSO)"
Const HEADERSOURCEREL As String = _

"http://schemas.openxmlformats.org/officeDocument/2006/relationships/mailMergeHeaderSource"
Const HEADERSOURCEDESC As String = _
    "Header Source"
Const MAINDOCREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
Const MAINDOCDESC As String = _
    "Main Document Part"
Const RECIPIENTDATAREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/recipientData"
Const RECIPIENTDATADESC As String = _
    "Recipient Data Part"
Const SETTINGSREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings"
Const SETTINGSDESC As String = _
    "Settings Part"

Const RELPARTPATH As String = "_rels"
Const RELPARTEXTENSION As String = ".rels"
Dim objCTP As clsContentTypesPart
```

Die Prozedur *Process1OpenXMLFile*

Diese Prozedur hat vier Arbeitsgänge:

1. Sicherstellen, dass die Datei der Parameter `StrFullName` und die Pfadangabe der Parameter `strBackupPath` existieren und, dass die Sicherheitskopie nicht schon vorhanden ist.
2. Eine Sicherheitskopie der vorhandenen Datei erstellen und diese in einen Arbeitsordner auspacken.
2. Den Namen des Hauptdokumentteils ermitteln, den Namen seines Beziehungsteils bestimmen und den Teil *settings* mit den Seriendruckinformationen öffnen. Danach wird die Prozedur `processSettingsPart` aufgerufen, um einen Bericht zu erstellen bzw. die Seriendruckinformationen zu löschen, in Abhängigkeit der booleschen Parameter `bDocument` und `bRemove`.
3. Packt den Arbeitsordner ein und ersetzt die ursprüngliche Datei.

Falls Sie den Inhalt des Arbeitsordners anschauen möchten, kommentieren Sie die folgende Codezeile aus:

```
objFSO.DeleteFolder strPackagePath, True
```

Ein einzelnes `FileSystemObject` `objFSO` wird erstellt und an alle Unterprozeduren übergeben. Mit diesem Objekt wird die allgemeine Dateiverwaltung ermöglicht, wie das Erstellen von Ordnerstrukturen oder die Prüfung von Dateien und Ordnern auf deren Vorhandensein.

Listing 22.11: Hauptmodul in Bsp22_06.docm – Hauptprozedur

```
Sub Process1OpenXMLFile( _
    strFullName As String, _
    strBackupPath As String, _
    bDocument As Boolean, _
    bRemove As Boolean _)

    Dim bContinue As Boolean
    Dim objFSO As Scripting.FileSystemObject
    Dim objXMLDoc As MSXML2.DOMDocument
    Dim strExtension As String
    Dim strPartFullName As String
    Dim strMainPartFullName As String
    Dim strPackagePath As String
    Dim strSettingsPartFullName As String
    Dim strTargetFullName As String
    Dim strFileName As String

    ' Phase 1

    bContinue = logHeader(strFullName)

    Set objFSO = New FileSystemObject
    If Not objFSO.FileExists(strFullName) Then
        bContinue = logError("Die Datei konnte nicht gefunden oder nicht geöffnet werden.")
    End If

    If bContinue Then
        strExtension = objFSO.GetExtensionName(strFullName)
        Select Case UCase(strExtension)
            Case "DOCX", "DOCX", "DOTM", "DOTX"
```

```

        ,
        Case Else
            bContinue = logError( _
                "Die Dateierweiterung muß .docm, .docx, .dotm oder .dotx sein.")
        End Select
    End If

    If bContinue Then
        If Not objFSO.FolderExists(strBackupPath) Then
            bContinue = logError( _
                "Konnte den Ordner '" & _
                strBackupPath & "' für die Sicherheitskopie nicht finden.")
        End If
    End If

    If bContinue Then
        strTargetFullName = _
            objFSO.BuildPath( _
                strBackupPath, _
                objFSO.GetFileName(strFullName))
        If objFSO.FileExists(strTargetFullName) Then
            bContinue = logError( _
                "Die Datei ist schon im Ordner für die Sicherheitskopie vorhanden. " & _
                "Die Ausführung für diese Datei wurde abgebrochen.")
        End If
    End If

    ' Phase 2

    If bContinue Then
        objFSO.CopyFile _
            Source:=strFullName, _
            Destination:=strTargetFullName, _
            overwritefiles:=True
        strPackagePath = _
            objFSO.BuildPath( _
                strBackupPath, _
                objFSO.GetBaseName(strFullName))
        bContinue = _
            logPack(modPackUnpack.unpack(strFullName, strPackagePath))
    End If

    ' Phase 3

    If bContinue Then
        Set objCTP = New clsContentTypesPart
        objCTP.openPart strPackagePath
        bContinue = _
            getSingleTarget( _
                strPackagePath, _
                PACKAGEDESC, _
                "[@Type='" & MAINDOCREL & "']", _
                MAINDOCDESC, _

```

```

        strMainPartFullName, _
        objFSO, _
        BuildTargetFullName:=True, _
        RelsPartShouldExist:=True, _
        RelShouldExist:=True, _
        RemoveRel:=False)
    If bContinue Then
        bContinue = _
        getSingleTarget( _
            strMainPartFullName, _
            MAINDOCDESC, _
            "[@Type='" & SETTINGSREL & "']", _
            SETTINGSDESC, _
            strSettingsPartFullName, _
            objFSO, _
            BuildTargetFullName:=True, _
            RelsPartShouldExist:=False, _
            RelShouldExist:=False, _
            RemoveRel:=False)
    If bContinue Then
        If strSettingsPartFullName <> "" Then
            processSettingsPart _
                strSettingsPartFullName, _
                bDocument, _
                bRemove, _
                objFSO
        End If
    End If
    objCTP.closePart
End If

' Phase 4

If bContinue Then
    bContinue = logPack(modPackUnpack.pack(strPackagePath, strFullName))
End If

If objFSO.FolderExists(strPackagePath) Then
    objFSO.DeleteFolder strPackagePath, True
End If

Set objFSO = Nothing
End Sub

```

Dokumenteigenschaften, Custom XML Parts und Inhaltssteuerelemente

Dieser Abschnitt bietet eine Übersicht der Benutzung und der Speicherorte von Dokumenteigenschaften in Word Open XML-Dokumenten an. Custom XML Parts und ihre Verwaltung in Open XML-Paketen werden zudem näher vorgestellt.

Dokumenteigenschaften speichern kleine Informationseinheiten über das Dokument.

- Ihr Inhalt kann oft mittels einer *DocProperty*-Feldfunktion innerhalb des Dokumenttexts angezeigt werden (beispielsweise der Name des Autors in einer Kopf- oder Fußzeile).
- Sie können gelesen und teilweise beschrieben werden, ohne das Objektmodell zu bemühen.
- Sie werden oft für die Dokumentverwaltung benutzt.

Dokumenteigenschaften vor der Ära Open XML

Vor Office 2007 gab es zwei Arten von Dokumenteigenschaften, die Office-eigenen und die benutzerdefinierten Eigenschaften.

Ungefähr 30 Office-eigene Eigenschaften werden bereit gestellt, wie Autor, Titel oder die Anzahl Wörter im Dokument. Einige davon wurden durch Word verwaltet. Andere konnten durch die Benutzerschnittstelle bzw. das Objektmodell festgelegt und bearbeitet werden. Der Inhalt vieler Eigenschaften kann wie erwähnt über *DocProperty*-Feldfunktionen im Dokument angezeigt werden.

Benutzerdefinierte Dokumenteigenschaften werden durch den Benutzer oder den Entwickler erstellt. Ihren Inhalt konnte ebenfalls mit *DocProperty*-Feldfunktionen angezeigt aber nicht aktualisiert werden. Ihnen wird ein bestimmter Datentyp (wie Zeichenkette, Zahl, Datum) zugewiesen. Der Wert der Eigenschaft konnte über das Objektmodell ausgelesen und beschrieben werden.

Hinweis Beginn

Die Anzeige einer Feldfunktion muß meistens ausdrücklich aktualisiert werden, um sicher zu stellen, dass sie die korrekte Information widerspiegelt.

Hinweis Ende

Dokumenteigenschaften bei Open XML

Ab Office 2007, gleichzeitig mit der Einführung des Open XML-Dateiformats, gibt es sechs Objekte, die als »Eigenschaften« betrachtet werden können. Im Gegensatz zum alten binären Dateiformat, ist es relativ einfach, mit der Hilfe von XML-Funktionalität wie Xpath, ein beliebiges Stück XML aus einem Dokument zu lesen. Diese Tatsache hat die Grenzen ein wenig verwischt. Jede Art von Objekt wird ein wenig anders gehandhabt, manchmal werden sogar verschiedene Eigenschaften der gleichen Art anders gehandhabt. Diese Unterschiede sorgen für etwas Verwirrung.

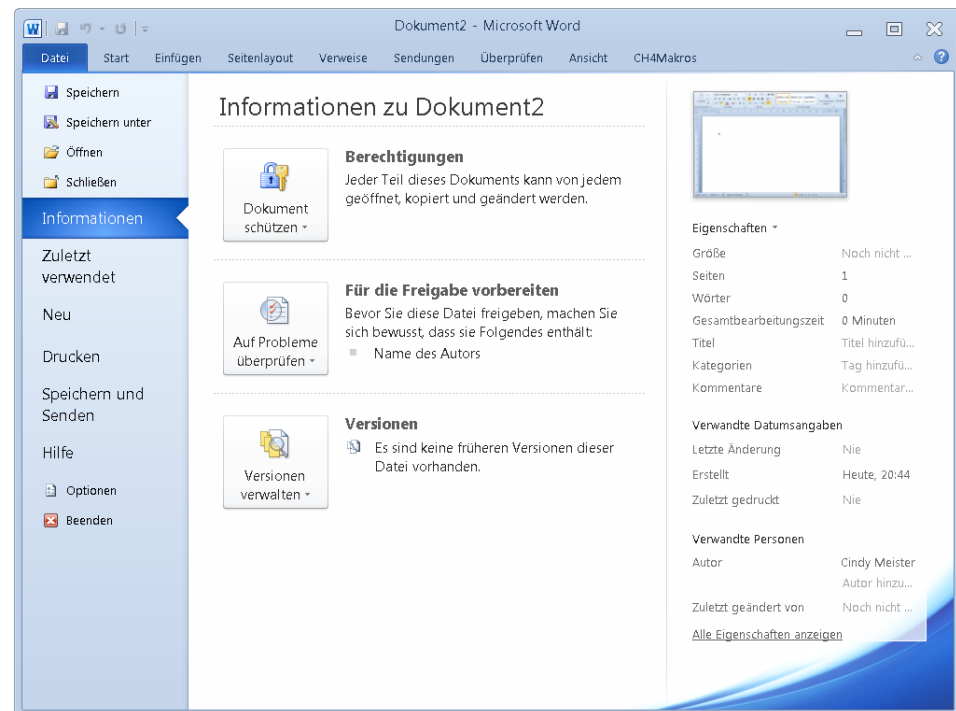
Die sechs Objektarten sind:

- Kerneigenschaften
- Erweiterte Eigenschaften
- Benutzerdefinierte Eigenschaften
- Deckblatteigenschaften
- »Server«-Eigenschaften
- Entwicklerdefinierte Eigenschaften, die in einem Custom XML Part gespeichert werden

Die Benutzerschnittstelle ermöglicht die Einsicht in und Bearbeitung von einigen Eigenschaftsarten. Office 2010 stellt dafür den Eigenschaftenbereich in der Registerkarte *Datei/Informationen* der Backstage-Ansicht zur Verfügung (Abbildung 22.6). In Office 2007 muß der Dokumentbereich über *Office-*

Schaltfläche/Vorbereiten/Dokumenteigenschaften eingeblendet werden. Das Dialogfeld mit weiteren Eigenschaften wird durch den Menübefehl *Dokumenteigenschaften/Erweiterte Eigenschaften* eingeblendet. (Dieser Bereich sowie das Dialogfeld stehen auch in Word 2010 zur Verfügung und werden über das Dropdownmenü *Eigenschaften* in *Datei/Informationen* eingeblendet.)

Abbildung 22.6: Der Eigenschaftenbereich, rechts; im oberen Teil ist der Dokumentbereich im Dokumentfenster sichtbar



Über *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften* ist es möglich, Inhaltssteuerelemente einzufügen, die mit einer Dokumenteigenschaft verknüpft sind.

Hinweis Beginn

Mehr zum Thema Schnellbausteine sowie eine Abbildung des Menüs sind im Kapitel 6 beschrieben.

Hinweis Ende

Kerneigenschaften

Diese entsprechen zum größten Teil den Office-eigenen Eigenschaften früherer Versionen. Sie sind im Open XML-Standard definiert. Wenn vorhanden, sind sie im Teil `/docProps/core.xml` gespeichert. (Word erstellt diesen Teil auch beim Speichern eines leeren Dokuments.)

Inhaltssteuerelemente für sieben dieser Kern-Eigenschaften stehen für das Anzeigen und Aktualisierung ihrer Inhalt zur Verfügung: Autor, Betreff, Kategorie, Kommentare, Schlüsselwörter, Status, sowie Titel. Dies sind die einzigen Dokumenteigenschaften die standardmäßig im Dokumentbereich angezeigt werden und über den Menübefehl *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften*

als verknüpftes Inhaltssteuerelement eingefügt werden können.

Erweiterte Eigenschaften

Diese Eigenschaften sind spezifisch für ein Open XML-Dokument. Words erweiterte Eigenschaften sind nicht die gleichen wie jene von Excel oder PowerPoint. Diese Eigenschaften sind ebenso im Open XML-Standard definiert. Wenn vorhanden, werden erweiterte Eigenschaften im Teil `/docProps/app.xml` gespeichert. (Word erstellt diesen Teil auch beim Speichern eines leeren Dokuments.)

Inhaltssteuerelemente stehen zur Verfügung für das Anzeigen und Aktualisierung von zwei dieser erweiterten Eigenschaften: Manager und Firma. Dies sind die einzigen erweiterten Eigenschaften, die über den Menübefehl *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften* als verknüpftes Inhaltssteuerelement eingefügt werden können.

Allgemein betrachtet, entsprechen die Kern- und die erweiterten Dokumenteigenschaften den Word-eigenen Eigenschaften aus früheren Versionen.

Benutzerdefinierte Eigenschaften

Die Benutzerdefinierten Eigenschaften entsprechen denjenigen aus früheren Versionen. Der Teil »Custom Document Properties« ist im Open XML-Standard definiert, es gibt jedoch keine Angaben über die Anzahl oder Namen der Eigenschaften, die darin gespeichert werden können. Wenn vorhanden werden sie in `/docProps/custom.xml` gespeichert. Diesen Teil wird erst bei Bedarf von Word erstellt.

Es ist nicht möglich, benutzerdefinierte Eigenschaften mit einem Inhaltssteuerelement zu verknüpfen; es gibt dafür auch keinen Eintrag im Menü *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften*. Der Inhalt kann jedoch mit einer *DocProperty*-Feldfunktion angezeigt werden.

Deckblatteigenschaften

Word 2007 führte das Konzept und die Schaltfläche *Deckblatt* ein. Damit wird am Dokumentanfang ein Deckblatt eingefügt, bestückt mit Feldern für die Eingabe von Informationen wie »Titel«, »Datum der Veröffentlichung« und »Kurzbeschreibung«. Im Ganzen sind es sechs deckblattspezifische Eigenschaften: Kurzfassung, Firmenadresse, Firmen-E-Mail-Adresse, Firmenfaxnummer, Firmentelefonnummer, und Veröffentlichungsdatum. Verknüpfte Inhaltssteuerelemente stehen für alle diese Eigenschaften im Menü *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften* zur Verfügung.

Word speichert deren Inhalt in einem gewöhnlichen Custom XML Part.

Ein solcher Teil dient als Beispiel für einen »wohl geformten« Custom XML Part. Word benutzt dafür bestimmte Schemas und arbeitet bewusst mit dem Teil.

»Server«-Eigenschaften

Diese Eigenschaften dienen der Dokumentverwaltung in einer Server-Umgebung, wie SharePoint. Sie sind im Open XML-Standard *nicht* definiert.

Beim Öffnen eines auf SharePoint gespeicherten Dokuments erstellt oder aktualisiert SharePoint drei sich im Dokument befindende Custom XML Parts. Einer dieser Teile enthält den aktuellen Wert der Eigenschaften, ein anderer die Schema-

Informationen. Die Eigenschaften erscheinen meistens in der Liste *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften*. Es ist auch möglich, diese Eigenschaften in einem Sonderteil »Document-Eigenschaften – Server« des Dokumentbereichs zu integrieren. Sie werden in Word 2010 auch im Eigenschaftenbereich der Backstage-Ansicht angezeigt, außer sie sind Teil einer Dropdownliste. In diesem Fall erscheint ein Link, welcher sie im Dokumentbereich einblendet.

»Server«-, wie Deckblatt-Eigenschaften, werden in einem wohl geformten Custom XML Part gespeichert.

Erwähnenswert ist, dass

- neben den Kern-, erweiterten und Deckblatt-Eigenschaften diese Eigenschaften automatisch durch Word in der Eigenschaftenlisten, dem Dokumentbereich und dem Eigenschaftenbereich der Backstage-Ansicht eingebunden werden
- mit InfoPath den Dokumentbericht angepasst werden kann
- für diese Eigenschaftsart erweiterte Funktionalität zur Verfügung steht. Die Eigenschaft kann beispielsweise als Dropdownliste bezeichnet werden. In diesem Fall übernimmt die Liste ihren Inhalt aus einem mit der Eigenschaft verknüpften Schema. Sie wird im Dokumentbereich und als Inhaltssteuerelement als Dropdownliste dargestellt.

Für eine eingehendere Diskussion dieser Eigenschaften müssen Sie in der SharePoint-Dokumentation nachschlagen.

Entwicklerdefinierte Eigenschaften

Es handelt sich hier um keine Eigenschaften im herkömmlichen Sinne. Da sie aber ohne Einsatz des Objektmodells gelesen und geschrieben und zudem in einem mit einem Inhaltssteuerelement verknüpften Custom XML Part gespeichert werden, betrachten wir sie als »Eigenschaften«. Word führt sie in keiner Eigenschaftenliste und keinem -bereich auf und verknüpft sie nicht automatisch mit einem Inhaltssteuerelement. Diese Handlungen müssen alle durch den Entwickler vollzogen werden.

Kern-, erweiterte und Deckblatt-Eigenschaften vs. Entwicklerdefinierte Eigenschaften

Word besitzt einen Mechanismus, um Inhaltssteuerelemente mit allen dieser Eigenschaftstypen zu verknüpfen. Da dieser mit den Word-Objekten CustomXMLParts und CustomXMLPart arbeitet, muss Word die Speicherorte von Kern- und erweiterten Eigenschaften als Custom XML Parts behandeln.

Wenn Sie durch die im Word-Dokument vorhandenen CustomXMLParts schleifen, werden auch die Kern-, erweiterten und Deckblattteile aufgelistet. Dies wird durch Ausführung der Prozedur in Listing 22.12 veranschaulicht, wenn ActiveDocument ein leeres Dokument ist.

Listing 22.12: Den Inhalt jedes CustomXMLParts des aktuellen Dokuments ausgeben.

```
Sub enumCustomXML()  
    Dim objCXPM As Office.CustomXMLPrefixMappings  
    Dim objCXPs As Office.CustomXMLParts  
    Dim objCXP As Office.CustomXMLPart  
    For Each objCXP In ActiveDocument.CustomXMLParts  
        Debug.Print objCXP.XML  
    End For  
End Sub
```

Next
End Sub

Im Direktfenster ((Strg)+(G)) werden Zeilen aufgelistet, die wie folgt anfangen, wobei »Properties« die erweiterten Eigenschaften bedeuten:

```
<cp:coreProperties
<Properties
<CoverPageProperties
```

Beim Betrachten des XML für die Kern- und erweiterten Eigenschaften fällt auf, dass das von Listing 22.12 ausgegebene XML nicht genau übereinstimmt mit dem Inhalt der Paketeile *core.xml* bzw. *app.xml*. Das über das Objektmodell zur Verfügung gestellte XML enthält lediglich die Elemente, die mit einem Inhaltssteuerelement verknüpft werden können.

Im Vergleich wird das XML, das aus einem »echten« Custom XML Part stammt, vollumfänglich ausgegeben.

Sicherlich ein Grund dafür ist, dass ein Inhaltssteuerelement nicht mit einem Wert verknüpft werden soll, den Word selber verwaltet (beispielsweise das Erstellungsdatum oder die Anzahl Seiten). Dieser Mechanismus ist jedoch nicht hundertprozentig konsequent.

CDROM Beginn

Auf der CD-ROM zum Buch finden Sie im Ordner *\Beilagen\XMLPackages* die Excel-Arbeitsmappe *wordeigenschaften.xlsx*. Diese listet verschiedene Informationen für die Kern-, erweiterten und Deckblatteigenschaften auf, mit den für die Verknüpfung benötigten XPath-Angaben.

CDROM Ende

Beispiel: Custom XML Parts ersetzen ohne das Word-Objektmodell

Im Kapitel 7 haben Sie erfahren, wie Standardbriefe mit Daten aus einer Datenbank erstellt werden. Ein Beispiel veranschaulicht die programmmäßige Verknüpfung von Inhaltssteuerelementen mit einem CustomXMLPart und die Übertragung der Daten in diesen Teil – alles mit dem Word-Objektmodell und MSXML.

In diesem Abschnitt zeigen wir eine alternative Methode auf, die sich *nicht* auf das Objektmodell von Word stützt. Stattdessen wird das Open XML-Dokument ausgepackt, die Daten aus der Nordwind2007.accdb-Datenbank direkt in den CustomXMLPart geschrieben, und das Dokument wieder eingepackt. Obwohl der Code sich in einem VBA-Modul befindet und in Word läuft, könnte dieser Vorgang genauso gut in Access oder einer Server-Anwendung, ohne Vorhandensein der Word-Anwendung, ausgeführt werden.

Abbildung 22.7: Das Dokument mit Inhaltssteuerelementen, verknüpft mit einem Custom XML Part

| | |
|---|---------------------------|
| Northwind GmbH Abteilung Verkauf | Tacoma, den 22. Juni 2010 |
| <p>Around the Horn Thomas Hardy 120 Hanover Sq. WA1 1DP London Großbritannien</p> | |
| <p>Betreff: Betreff</p> | |
| <p>Sehr geehrtes Fräulein Thomas Hardy</p> | |
| <p><u>Den Briefinhalt hier eingeben.</u></p> | |
| <p>Wählen Sie eine Grußformel aus.</p> | |
| <p>Andrew Fuller</p> | |

Der Code in *Bsp22_07.docm* führt folgende Handlungen aus:

- Sammelt die Daten für alle Absender und Kunden aus der Nordwind2007.acddb-Datenbank
- Blendet ein Dialogfeld ein, in welchem der Benutzer den Absender und Empfänger sowie die bevorzugte Sprache für die Grußzeilen auswählt.

- Packt den Standardbrief (*Bsp22_08.docm*) aus
- Schreibt die Daten für Absender und Kunden in den passenden Custom XML-Teil des Briefs
- Packt den Ordnerinhalt in ein neues Dokument (*Bsp22_09.docm*) ein
- Öffnet das erstellte Dokument, um die Demo fortzuführen

Beim Öffnen von *Bsp22_09.docm* wird ein Makro ausgeführt, dass die Listen der Inhaltssteuerelemente für Begrüßungen mit Auswahloptionen in der gewählten Sprache füllt. (Es wäre durchaus möglich gewesen, diese Listen bei der Bearbeitung des XMLs zu füllen, wir wollen aber auch die andere Methode aufzeigen.)

Das Beispiel ausführen

CD-ROM Beginn

Für dieses Beispiel benötigen Sie die Beispieldokumente *Bsp22_07.docm* und *Bsp22_08.docm*. Beide befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*. Es setzt zudem die Datenbank *Nordwind2007.accdb* voraus.

CD-ROM Ende

Um das Beispiel vorzubereiten müssen Sie

- *Bsp22_07.docm* öffnen und den VB-Editor starten ((Alt)+(F11)).
- Das Modul *Bsp22_07_Datenbank* einblenden und die Pfadangaben in den Konstanten *sBriefDatei*, *sTargetBriefDatei* und *sArbeitPfad* Ihrem System entsprechend anpassen.
- Die Prozedur *Brief_Vorbereiten* im Modul *bsp22_07* ausführen.
- Im eingblendeten Dialogfeld einen Absender, einen Kunden und eine Sprache auswählen, dann auf **OK** klicken.

Der Code führt die beschriebenen Handlungen aus und öffnet am Schluss das Dokument *Bsp22_09.docm*.

Das Hauptmodul: Bsp22_07.docm

Dieses Modul enthält einige Prozeduren, die in Listing 22.13 ersichtlich sind:

- *Brief_Vorbereiten* blendet das Dialogfeld ein und ruft beim Klicken der Schaltfläche **OK** die Prozedur *BriefInfoErstellen* auf.
- Diese ist ähnlich wie in den bisherigen Beispielen aufgebaut: sie stellt sicher, dass die benötigten Beispieldateien existieren. In diesem Fall wissen wir, dass der Custom XML Part bereits im Dokument vorhanden ist.
- Weiter ersetzt die Prozedur die Daten in den Elementen des XML. (Alternativ könnte das XML im Speicher zusammengestellt und den vorhandenen Teil ersetzen.)

Die unterstützenden Prozeduren sind denen im Beispiel *Bsp07_03_CC.dotm* ähnlich und sind hier nicht aufgeführt.

Listing 22.13: Hauptmodul (Auszug) von Bsp22_07.docm

```
Option Explicit

Public Const sBriefDatei As String = "c:\Beispiele\Kap22\Bsp22_08.docm"
Public Const sTargetBriefDatei As String = "c:\Beispiele\Kap22\Bsp22_09.docm"
```

```
Public Const sArbeitPfad As String = "c:\Beispiele\Kap22\Bsp22_07"
Public Const sBriefXMLDatei As String = "/customXml/item2.xml"

Public Sub Brief_Vorbereiten()
    Dim frm As frmBsp22_07

    Set frm = New frmBsp22_07
    frm.Show
    If frm.Tag = "OK" Then
        BriefInfoErstellen frm
    End If
    Unload frm
    Set frm = Nothing
    Word.Documents.Open sTargetBriefDatei
End Sub

Private Sub BriefInfoErstellen( _
    frm As frmBsp22_07 _
)

    Dim bContinue As Boolean
    Dim objFSO As Scripting.FileSystemObject
    Dim objXMLDoc As MSXML2.DOMDocument60
    Dim objRS As ADODB.Recordset
    Dim strSQL As String
    Dim strBriefXmlDatei As String

    bContinue = True
    Set objFSO = New FileSystemObject
    If bContinue Then
        If Not objFSO.FileExists(sBriefDatei) Then
            bContinue = logError("Das Quelldokument konnte nicht gefunden " & _
                                "oder nicht geöffnet werden.")
        End If
    End If

    If bContinue Then
        If Not objFSO.FolderExists(sArbeitPfad) Then
            objFSO.CreateFolder sArbeitPfad
        End If
        If Not objFSO.FolderExists(sArbeitPfad) Then
            bContinue = logError("Der Pfad zum Arbeitsordner konnte nicht gefunden " & _
                                "oder nicht erstellen werden.")
        End If
    End If

    If bContinue Then
        bContinue = logPack(unpack(sBriefDatei, sArbeitPfad))
    End If

    If bContinue Then
        strBriefXmlDatei = _
```

```

objFSO.BuildPath(sArbeitPfad, sBriefXMLDatei)
If objFSO.FileExists(strBriefXmlDatei) Then
    Set objXMLDoc = New MSXML2.DOMDocument60
    With objXMLDoc
        .validateOnParse = True
        If .Load(strBriefXmlDatei) Then
            'Die Absender-Informationen ermitteln und zuweisen
            With .SelectSingleNode("//absender").Attributes
                .getNamedItem("name").Text = frm.cboAbsender.Text
                .getNamedItem("id").Text = frm.cboAbsender.Value
            strSQL = _
                " SELECT Ort" & _
                " FROM   Personal" & _
                " WHERE  [Personal-Nr] = " & frm.cboAbsender.Value
            Set objRS = ADODatenHolen(strSQL)
            .getNamedItem("ort").Text = RSvalue(objRS, "Ort")
            Set objRS = Nothing
        End With

        'Die Kunden-Informationen ermitteln und zuweisen
        strSQL = _
            " SELECT *" & _
            " FROM   Kunden" & _
            " WHERE  [Kunden-Code] = '" & _
            frm.cboKunde.Value & "'"
        Set objRS = ADODatenHolen(strSQL)
        .SelectSingleNode("//kunde/kunden-nr").Text = _
            frm.cboKunde.Value
        .SelectSingleNode("//kunde/firma").Text = _
            RSvalue(objRS, "Firma")
        .SelectSingleNode("//kunde/kontakt").Text = _
            RSvalue(objRS, "Kontaktperson")
        .SelectSingleNode("//kunde/strasse").Text = _
            RSvalue(objRS, "Straße")
        .SelectSingleNode("//kunde/ort").Text = _
            RSvalue(objRS, "Ort")
        .SelectSingleNode("//kunde/plz").Text = _
            RSvalue(objRS, "PLZ")
        .SelectSingleNode("//kunde/land").Text = _
            RSvalue(objRS, "Land")

        .SelectSingleNode("//bestellung/sprache").Text = _
            AusgewählteSprache(frm)

        'Das vom Benutzer angegebene Datum in das XML-Dokument schreiben
        .SelectSingleNode("//datum").Text = Format(Date, "d. MMMM yyyy")

        .Save strBriefXmlDatei
    Else
        bContinue = logError("Could not open the expected Custom XML Part")
    End If
End With
Set objXMLDoc = Nothing

```



```

Else
    bContinue = logError("Der erwartete Custom XML Part konnte nicht geöffnet werden.")
End If
bContinue = logPack(pack(sArbeitPfad, sTargetBriefDatei))
End If
End Sub

```

Das Hauptmodul Bsp22_08/09.docm

Beim Öffnen des neuen Dokuments *Bsp22_09.docm* wird das Ereignis `Document_Open` angestoßen (Listing 22.14).

Listing 22.14: Das Ereignis `Document_Open`

```

Private Sub Document_Open()
    Brief_Vorbereiten
End Sub

```

Dieses ruft die Prozedur `BriefVorbereiten` auf, die folgende Aufgaben durchführt:

- Code für die Sprache aus dem Custom XML Part einlesen.
- Anhand des Codes Zusammenstellung einer Liste von Begrüßungsformeln in der gewählten Sprache. Diese Information ist, im Gegensatz zum Beispiel des Kapitel 7, innerhalb des Dokuments, in weiteren Custom XML Parts gespeichert.
- Die Inhaltssteuerelemente, wie im Beispiel des Kapitels 7, gruppieren und sperren.

Als der »Besitzer« (Entwickler und Ersteller des Dokuments und der Custom XML Parts) kennen wir die ID-Werte der Custom XML Parts. Wir können diese, anstelle des Namensraumes einsetzen, um die Custom XML Parts direkt anzusprechen. Die Konstante mit diesen ID-Werten sind am Anfang der Prozedur in Listing 22.15 ersichtlich.

Listing 22.15: *Bsp22_08/09 – Main Module*

```

Option Explicit

Const sBriefXmlId As String = "{AF41273F-F24B-4C4A-BA90-186D44611319}"
Const sAnredeXmlId As String = "{163C49DF-ED09-4F91-A5B6-D4A4158F07C4}"
Const sGruessXmlId As String = "{63B0593E-64F5-43F8-96D8-0A49080A59CA}"

Public Sub Brief_Vorbereiten()
    Dim cc As Word.ContentControl
    Dim ccGruppe As Word.ContentControl
    Dim doc As Word.Document
    Dim objCXP As office.CustomXMLPart
    Dim rng As Word.Range
    Dim strSprache As String

    Set doc = ActiveDocument
    Set objCXP = doc.CustomXMLParts.SelectByID(sBriefXmlId)
    strSprache = objCXP.SelectSingleNode("/bestellung/sprache").Text
    Set objCXP = Nothing
    'Inhaltssteuerelemente fertig vorbereiten
    ListeFüllen doc, "Anrede", "anrede", sAnredeXmlId, strSprache
    ListeFüllen doc, "Grußformel", "begrueßung", sGruessXmlId, strSprache

```

```

'Die Inhaltssteuerelemente gegen das Löschen schützen
For Each cc In doc.ContentControls
    If cc.Title <> "Unterschriftsgrafik" Then
        cc.LockContentControl = True
    End If
    If cc.Title = "Beschreibung" Or cc.Title = "Artikel-Nr" Then
        cc.LockContents = True
    End If
Next

'Die Inhaltssteuerelemente gruppieren, um das Dokument zu schützen
Set rng = doc.Content
rng.MoveEnd Unit:=wdCharacter, Count:=-1
rng.Select
Set ccGruppe = doc.ContentControls.Add( _
    Type:=wdContentControlGroup, _
    Range:=Selection.Range)
ccGruppe.LockContentControl = False
doc.Bookmarks("BriefInhalt").Range.ContentControls(1).Range.Select
Set ccGruppe = Nothing
Set rng = Nothing
Set doc = Nothing
End Sub

Private Sub ListeFüllen( _
    doc As Word.Document, _
    strCCTitle As String, _
    strElement As String, _
    strXmlID As String, _
    strSprache As String)

Dim cc As Word.ContentControl
Dim objCXP As office.CustomXMLPart
Dim objNodes As office.CustomXMLNodes
Dim objNode As office.CustomXMLNode
Set objCXP = doc.CustomXMLParts.SelectByID(strXmlID)
Set objNodes = objCXP.SelectNodes("//" & strSprache & "/" & strElement)
For Each cc In doc.SelectContentControlsByTitle(strCCTitle)
    cc.DropDownListEntries.Clear
    For Each objNode In objNodes
        cc.DropDownListEntries.Add objNode.Text
    Next
Next
Set objNode = Nothing
Set objNodes = Nothing
Set objCXP = Nothing
End Sub

```

Open XML-Dokumente & XSLT

Die XSLT (Transformationen) wurde im Kapitel 21 kurz vorgestellt. Es handelt sich um eine kräftige, nicht prozedurale Sprache, die den Inhalt eines XML-Dokuments in eine Vielzahl andersartige Dokumentformate umwandeln kann. Oft wird XSLT dazu benutzt, um ein XML-Quelldokument in eine HTML Webseite umzuwandeln. Enthält das XML-Dokument mehrere Einträge einer bestimmten Art (anders ausgedrückt, einer Liste gleichartiger Elemente), können diese beispielsweise in eine HTML-Tabelle ausgegeben werden. Vereinfacht gesehen besteht das XSLT aus zwei Umwandlungsmustern:

- Muster 1 erstellt das HTML für die gesamte Webseite, inklusiv die Struktur-Elemente für die Tabelle und ihre Kopfzeile.
- Muster 2 generiert je eine Tabellenzeile pro Eintrag einer Liste von gleichartigen Elementen.

Eine Anpassung dieser Muster ermöglicht die Erstellung eines Word Open XML-Dokument. In diesem Fall

- Muster 1 erstellt das WordProcessingML für das Word-Dokument, unter anderem mit den Element-Strukturen für eine Tabelle und ihre Kopfzeile.
- Muster 2 generiert in WordProcessingML eine Tabellenreihe pro Eintrag einer Liste von gleichartigen Elementen.

Wenn Sie jemals mit der Seriendruckfunktionalität gearbeitet haben, wird Ihnen diese Idee bekannt vorkommen. XSLT ist in der Tat eine Methode, ein 1:n Resultat zu erzielen, wie beim Seriendruck. Sie ist vor allem nützlich auf einem Server-System, wo die XML-Daten aus einer SQL-Datenquelle stammen.

Überlegungen zur Arbeit mit XSLT

Es müssen einige Punkte beachtet werden, wenn Sie Open XML-Dokument mit XSLT erstellen möchten:

- Sie müssen zuerst die XSLT-Sprache beherrschen. Falls Sie ausschließlich mit XML-Werkzeuge von Microsoft arbeiten, bedeutet dies XSLT 1.0 sowie Xpath 1.0. Diese haben gegenüber XSLT 2.0 und Xpath 2.0 eine begrenzte Funktionalität.
- Sie müssen eingehende Kenntnisse des WordProcessingML haben.
- Die auszuführende Handlung soll genau analysiert werden, um die optimale Vorgehensweise zu ermitteln. Falls die Aufgabe daraus besteht, Text in das Hauptdokument einzufügen, wäre es sinnvoll, diesen Teil aus das Paket zu lösen, dessen XML zu transformieren und den Teil im Paket anschließend zu ersetzen.

Muss zusätzlich der Inhalt weiterer Teile geändert werden (beispielsweise Kopf- und Fußzeilen), muss jeder Teil für sich transformiert werden. Oder das Paket könnte in flache OPC-Format umwandelt werden, dieses transformiert, und das Resultat wieder in ein standardisiertes Paket zurückgewandelt werden.

- Das Erstellen und die Formatierung jedes kleinen Teils des neuen Inhalts muß überlegt werden. Nehmen wir als Beispiel die Einbeziehung eines Datums in der Fußzeile. Wird es durch XSLT eingefügt? Kann das XSLT 1.0 überhaupt? Oder wäre es besser, das Datum durch eine Feldfunktion anzugeben? Wenn ja, kann garantiert werden, dass die Feldfunktion aktualisiert wird, sodass das Ergebnis

sichtbar ist und ausgedruckt wird? Wie steht es mit der Formatierung des Datums? Wird die Transformation auf das Format TT.MM.JJJJ bestehen, oder kann man es den Einstellungen des Benutzersystems überlassen? Falls man das Letztere machen möchte, wie ist vorzugehen? Vielleicht wäre eine Feldfunktion doch besser geeignet?

- Sie müssen wissen, was das Resultat enthalten soll. Falls Sie beispielsweise 1.000 Datensätze in ein Dokument ausgeben möchten, mit einem Abschnitt pro Datensatz, braucht es auch 1.000 verschiedene Kopf- und Fußzeilen? Vielleicht schon, wenn jede andere Informationen enthalten soll.
- Es muß sichergestellt werden, dass die zur Verfügung stehenden Werkzeuge fähig sind, mit der Datenmenge und der Komplexität der Aufgabe umzugehen. Sie können damit vielleicht ein Dokument mit 1.000 Abschnitten generieren, aber was ist, wenn es 1.000.000 Datensätze sind?
- Die benötigten Ressourcen für den Unterhalt einer auf XSLT basierenden Lösung müssen in Betracht gezogen werden.

Eine letzte Bemerkung

Das Erstellen eines Word-Dokuments ist nur eine der möglichen Aufgaben, wofür sich XSLT eignet. Jedes Mal, wenn Sie XML generieren oder ändern müssen, stellt sich die Frage, wie am besten vorzugehen ist:

- Ist die Aufgabe am besten über die Programmierschnittstelle eines DOMDocument zu lösen?
- Oder wäre es besser, ein XSLT zu schneiden, um das benötigte XML zu generieren?

Die einfachere oder robustere Methode ergibt sich vom Fall zu Fall. Ebenfalls müssen die Kenntnisse, Erfahrung und Fähigkeiten des Entwicklers in Betracht gezogen werden. Jemand, der sich mit XSLT sehr gut auskennt, aber von der anzuwendenden Programmiersprache wenig Kenntnisse hat, wird beispielsweise die Aufgabe mittels XSLT schneller und zuverlässiger lösen können.

Beispiel

CDROM Beginn

Die Dateien für das Beispiel befinden sich im *Kap22_XSLT.zip* auf der CD-ROM im Ordner *\Beispiele\Kap22*. Alle Einzelheiten zum Beispiel können Sie dem englischen Text des Dokuments *Kap22_XSLT.doc* entnehmen.


CDROM Ende

Aus Platzgründen können wir kein Codebeispiel für den Einsatz von XSLT mit Open XML ins Buch miteinbeziehen. Es steht jedoch eines auf der CD-ROM zur Verfügung. Damit wird mit XSLT und Daten aus der Nordwind2007.accdb-Datenbank ein Bericht mit 1:n-Tabellen erstellt. Der Ausgangspunkt ist in Abbildung 22.8 ersichtlich und das Endresultat in Abbildung 22.9.

Abbildung 22.8: Die Vorlage für einen 1:n-Bericht, generiert mit einer Transformation von Office Open XML

| Bericht von Bestellungen durch Kunden | | | | | | |
|---------------------------------------|---------------------|---------------|------------------|------------------|---------------------|---------------|
| Firma: @Firma@ | | | | | | |
| Kunden-Code: @Kunden-Code@ | | | | | | |
| Bestell-Nr. | Bestelldatum | Endpreis | Verkaufskunden | Liefdatum | Vermanddatum | Vermand per |
| @Bestel- nr@ | @Bestelldatu- m@ | @Endpreis@ | @Verkaufskunden@ | @Liefdatum m@ | @Vermanddatum m@ | @Vermand per@ |
| | | @Gesamtpreis@ | | | | |
| Ende des Berichts. | | | | | | |
| @Produktionsfirma@ | | | | | | |

Abbildung 22.9: Das durch XSLT erstellte 1:n-Ergebnis



NORDWIND
GmbH

Bericht von Bestellungen durch Kunden

Firma: **Alfreds Futterkiste**
 Kunden-Code: **ALFKI**

| Bestell-Nr | Bestelldatum | Endpreis | Verkauft/Bestellt von | Umfeldatum | Versanddatum | Versand per |
|------------|--------------|----------|---------------------------|------------|--------------|-----------------------------|
| 30643 | 25.09.1997 | 407,25 | Michael Seabra | 21.09.1997 | 02.09.1997 | Speedy Express |
| 30691 | 02.10.1997 | 439,00 | Margaret Peacock | 21.10.1997 | 12.10.1997 | United Package |
| 30701 | 12.10.1997 | 165,00 | Margaret Peacock | 24.11.1997 | 21.10.1997 | Speedy Express |
| 30635 | 15.01.1998 | 412,90 | Nancy Davolio | 12.02.1998 | 21.01.1998 | Federal Shipping |
| 30652 | 16.02.1998 | 225,60 | Nancy Davolio | 27.04.1998 | 24.02.1998 | Speedy Express |
| 31011 | 09.04.1998 | 466,75 | Janet Levinski | 07.05.1998 | 12.04.1998 | Speedy Express |
| | | 2.136,50 | | | | |

Firma: **Ana Trujillo Emparedados y helados**
 Kunden-Code: **ANATR**

| Bestell-Nr | Bestelldatum | Endpreis | Verkauft/Bestellt von | Umfeldatum | Versanddatum | Versand per |
|------------|--------------|----------|---------------------------|------------|--------------|-----------------------------|
| 30528 | 18.09.1996 | 84,40 | Robert King | 16.10.1996 | 24.09.1996 | Federal Shipping |
| 30675 | 09.09.1997 | 239,88 | Janet Levinski | 09.09.1997 | 14.09.1997 | Speedy Express |
| 30759 | 28.11.1997 | 160,00 | Janet Levinski | 26.12.1997 | 12.12.1997 | Federal Shipping |
| 30926 | 04.02.1998 | 257,20 | Margaret Peacock | 01.04.1998 | 11.02.1998 | Federal Shipping |
| | | 701,48 | | | | |

Firma: **Antonio Moreno Taquería**
 Kunden-Code: **ANTON**

| Bestell-Nr | Bestelldatum | Endpreis | Verkauft/Bestellt von | Umfeldatum | Versanddatum | Versand per |
|------------|--------------|----------|---------------------------|------------|--------------|-----------------------------|
| 30365 | 27.11.1996 | 201,60 | Janet Levinski | 25.12.1996 | 01.12.1996 | United Package |
| 30507 | 15.04.1997 | 974,52 | Robert King | 12.05.1997 | 22.04.1997 | Speedy Express |
| 30525 | 12.05.1997 | 970,42 | Margaret Peacock | 10.06.1997 | 21.05.1997 | Speedy Express |
| 30572 | 19.06.1997 | 1.041,00 | Robert King | 17.07.1997 | 20.06.1997 | Federal Shipping |
| 30677 | 22.09.1997 | 406,69 | Nancy Davolio | 20.10.1997 | 26.09.1997 | Federal Shipping |
| 30692 | 25.09.1997 | 187,75 | Janet Levinski | 22.10.1997 | 01.10.1997 | United Package |
| 30956 | 29.01.1998 | 220,00 | Janet Levinski | 25.02.1998 | 10.02.1998 | United Package |
| | | 2.511,98 | | | | |

Firma: **Around the Horn**
 Kunden-Code: **AROUT**

| Bestell-Nr | Bestelldatum | Endpreis | Verkauft/Bestellt von | Umfeldatum | Versanddatum | Versand per |
|------------|--------------|----------|---------------------------|------------|--------------|-----------------------------|
| 30335 | 15.11.1996 | 240,00 | Michael Seabra | 12.12.1996 | 20.11.1996 | Speedy Express |
| 30393 | 16.12.1996 | 449,50 | Laura Callahan | 12.01.1997 | 18.12.1996 | Federal Shipping |
| 30453 | 21.02.1997 | 202,85 | Nancy Davolio | 21.02.1997 | 26.02.1997 | United Package |
| 30556 | 04.06.1997 | 1.071,45 | Nancy Davolio | 02.07.1997 | 10.06.1997 | United Package |
| 30707 | 16.10.1997 | 820,50 | Margaret Peacock | 20.10.1997 | 22.10.1997 | Federal Shipping |
| 30741 | 14.11.1997 | 114,00 | Margaret Peacock | 28.11.1997 | 18.11.1997 | Federal Shipping |
| 30742 | 17.11.1997 | 159,60 | Nancy Davolio | 18.12.1997 | 21.11.1997 | United Package |
| 30769 | 06.12.1997 | 728,50 | Janet Levinski | 05.01.1998 | 12.12.1997 | United Package |

23.06.2010

Zusammenfassung

Im vorliegenden Kapitel haben wir einen kurzen Überblick des Office Open XML-Dateiformats und die Arbeit damit geboten:

- Begonnen wurde das Kapitel mit einer Diskussion über die Ursprünge und Ziele des Office Open XML-Dateiformats (Seite 1 ff.).
- Als nächstes haben wir gezeigt, wie mit einem Texteditor ein einfaches Open XML Word-Dokument im flachen OPC-Format erstellt wird. Die Grundkonzepte Pakete, Teile und Beziehungen wurden vorgestellt (Seite 2 ff.).
- Es folgte ein einfaches Beispiel, wie mit VBA (ohne Bezug des Word-Objektmodells) ein Teil in ein Paket eingefügt wird (Seite 11 ff.).
- Anschließend haben wir vertiefte Informationen über die zur Verfügung stehenden Typen von Teilen und Beziehungen präsentiert. Die Bestimmungen für die Benennungen von Teilen wurden vorgestellt, sowie eine kurze Einführung in die Standarddokumentation (Seite 19 ff.).
- Im nächsten Abschnitt wurde über die programmtechnische Bearbeitung von Open XML sowie geeignete Methoden und Werkzeugen diskutiert. Die Vor- und Nachteile der Arbeit mit dem Objektmodell gegenüber der direkten Bearbeitung von Paketen und Teilen wurden erwägt (Seite 28 ff.).
- Ein Beispiel zum Navigieren durch die Teile und Beziehungen eines Pakets mit VBA folgte. Es zeigte auf, wie XML und Beziehungen entfernt werden. Das alles ohne Hinzuziehung des Objektmodells (Seite 32 ff.).
- Dokumenteigenschaften und ihren Zusammenhang mit Custom XML Parts war das Thema im nächsten Abschnitt. Diese Diskussion wurde abgerundet durch ein VBA-Beispiel (basierend auf *Bsp_07_03_CC.dotm*) für das Einfügen eines Custom XML Parts in ein Word-Dokument, ohne das Objektmodell zu bemühen (Seite 39 ff.).
- Das Kapitel wurde abschlossen mit einer kurzen Vorstellung der Nützlichkeit von XSLT bei der Arbeit mit Open XML-Dokumenten (Seite 51 ff.).