kurz & bündig	
Inhalt	Einfach mal schnell ein Worddokument ohne Word oder externe Komponenten erzeugen. Natürlich mit der Voraussetzung, dass der Nutzer was mit *.docx-Dokumenten anfangen kann.
Zusammenfassung	Hier soll kein Word nachgebaut werden, dafür existiert ja bereits ein Open-Pendant. Es soll ein kurzer Überblick über die Funktionsweise des OpenXml-Formats, bzw. der Syntax gegeben werden. Der Schwerpunkt liegt auf der Erarbeitung eines ersten erweiterbaren Frameworks für die Erzeugung von Worddokumenten.
Quellcode	VB.NET, VS 2008

Word on the Rocks

Worddokumente im Office OpenXml-Format programmatisch erzeugen

Mit dem Erscheinen von Office 2007 ging Microsoft das erste Mal einen neuen Weg für das Officepaket und legte die Syntax für alle Microsoft Office-Dokumente (Word, Excel, …) offen. Bisher waren selbst gestrickte Lösungen für das Erzeugen von Worddokumenten sehr mühsam bis deprimierend.

Von Jacques Kohler

Mit dem im Dezember 09 erschienen neuen Open XML Format SDK 2 [1] von Microsoft („MS Office programmieren – ohne Office“ dot.net magazin 02.2010) erscheint das hier erarbeitet Framework als verspäteter Overhead, da hier auf Basis der Version 1 [2] gearbeitet wurde. In dieser nun veralteten Version ist es nur möglich den Inhalt als XML-String zu schreiben, ohne auf typisierte Objekte zugreifen zu können, die im folgenden erarbeitet werden sollen. Die Version des SDK 1kann noch mit dem .Net-Framework 2.0 betrieben werden, während für das SDK 2.0 schon das .Net Framework 3.5 notwendig wird.

Bevor wir mit dem Konzept für das Framework loslegen, ist es wichtig, die Eigenschaften und den Aufbau des Open Xml-Formats zu kennen. Gleich vorab: es werden später in der Klassen-bibliothek nur die grundsätzlichen und wichtigsten Elemente behandelt, um ansehnliche, einfache Worddokumente zu erstellen. Alle weiteren möglichen Funktionen wie Diagramme (die z.B. über ein eingebundenes Excel-Sheet umgesetzt werden), Sprungmarken, … müssen selbst erarbeitet werden, bzw. folgen in einem späteren Artikel.
Das erarbeitete Framework ist auf einem deutschen System erarbeitet worden, von daher kann keine Garantie gegeben werden, dass nicht doch irgendwo auf einem anders sprachigen System ein Haken ist.

Ich weiß, dass ich als VBler eher in der Minderheit bin, von daher noch der Hinweis, dass Kommentare statt „\\“ mit „‘ “ gemacht werden (Nur falls einige nachher in den Listings etwas verwirrt über die komischen Zeichen sind).

Office OpenXml-Format

Generell handelt es sich im neuen Format um einen zip-komprimierten Ordner, der mit der passenden Endung für die jeweilige Officelösung versehen ist.

Endung	Office 2007
*.docx	Word
*.xlsx	Excel
*.pptx	PowerPoint

Wenn die Endung ein „m“ inne hat, enthält die Datei Macros, die uns hier im Rahmen des Artikels aber nicht interessieren sollen.

Jede *.docx-Datei kann durch die standardisierte Komprimierung einfach in *.zip umbenannt werden und schon hat man Zugriff auf den Inhalt. Dadurch kann man aus Betriebssystem-Bordmitteln ein Worddokument ohne Hilfsmittel lesen und erstellen. Dies soll im nächsten Abschnitt erläutert werden. Eine exemplarische Datei- und

Ordnerstruktur eines gespeicherten Worddokuments zeigt Abbildung 1.

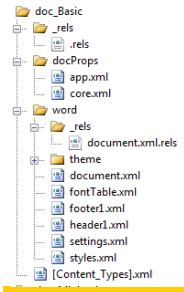


Abb_1.png

Abb. 1: Struktur Worddokument

Das primäre Element ist die Datei „document.xml“ im Ordner „Word“. Parallel zur Hauptdatei gibt es Xml-Dateien, die einzelne Einheiten wie Header, Styles ... repräsentieren. Diese Xml-Elemente werden Parts genannt. Dessen Erstellung ist mittels dem später verwendeten SDK [3] relativ einfach zu handhaben. Im folgenden seien alle möglichen Parts kurz erwähnt (die mit X gekennzeichneten Elemente werden hier erläutert, bzw. verwendet)

Comments
Styles [x]
Headers [x]
Footer [x]
Charts
Diagramms
Main document [x]
Images[x]
Footnotes[x]
Endnotes
Font Tables
Code
Glossary
Settings

Um die verschiedenen Parts mit der Datei document.xml zu verknüpfen, oder um z.B. Bilder in Headern zu referenzieren, liegen im Ordner „_rels“ für jede externe Referenzierung eines Parts jeweils eine Xml-Datei. Im Ordner „docProps“ sind die allgemeinen Dokumenteigenschaften wie Erstelldatum, Bearbeiter, … hinterlegt.

Eine schrittweise Einführung in die Thematik bietet das e-Book (neudeutsch für Buch im PDF-Format bereitgestellt im Internet) von Van Vugt [3]. Wem das E-Book zu wenig Information enthält, dem sei wärmstens die gesamte Spezifikation des Standards ans Herz gelegt (ca. 5.000 Seiten) [4].

世界您好 (shi jie nin hao)

Ich hätte den nächsten Abschnitt auch einfach „Hello World“ nennen können. Im Zuge der Globalisierung sollte man sich wohl frühzeitig ein wenig an die von vielen prophezeite neue Weltsprache gewöhnen.

Von einer rein manuellen Modellierung im Texteditor sollte man Abstand nehmen, da dies sehr schnell komplex und unübersichtlich werden kann. Als Einstiegsübung zum besseren Verständnis der Struktur der Hauptdatei „document.xml“ werden wir dies trotzdem angehen. Wenn kein Interesse besteht einfach weiter mit dem Kapitel „Das Basisdokument“.

Absolute Grundlage für die *.docx-Datei bilden die in Abbildung 2 dargestellten Dateien und Struktur. Die Ordner anlegen und die Dateien entsprechend als leere Textdateien anlegen und mit der richtigen Endung versehen. Bei exotischen Zeichen, die eventuell im Text vorkommen(wie z:B. Chinesisch) darauf achten, dass die Datei im Editor mittels „speichern unter“ als Codierung UTF-8 oder Unicode abgespeichert wird.

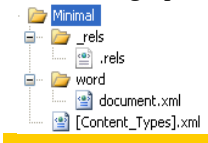


Abb_2.png

Abb 2 Minimale Struktur

Die Minimal notwendige Syntax um Inhalt im Hauptdokument „document.xml“ zu erzeugen zeigt Listing 1 (Bei Verwendung eines Betriebssystems vor Vista werden statt den chinesischen Zeichen nur Rechtecke angezeigt. In dem Fall dann doch „Hello World“ verwenden).

Listing 1

```
    <?xml version="1.0"
encoding="UTF-8" standalone="yes"
?>
    <w:document
xmlns:w="http://schemas.openxmlfo
rmats.org/wordprocessingml/2006/m
ain">
    <w:body>
    <w:p>
    <w:r>
    <w:t>世界您好</w:t>
    </w:r>
    </w:p>
    </w:body>
    </w:document>
```

/Listing

Die Texte für die anderen beiden Dateien sind Listing 2 zu entnehmen.

Listing 2

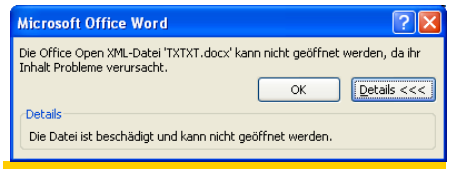
```
    //_.rels
    <?xml version="1.0"
encoding="UTF-8"
standalone="yes"?>
    <Relationships
xmlns="http://schemas.openxmlform
ats.org/package/2006/relationship
s">
        <Relationship Id="rId1"
Type="http://schemas.openxmlforma
ts.org/officeDocument/2006/relati
onships/officeDocument"
Target="word/document.xml"/>
    </Relationships>

    //[Content_Types].xml
```

```
    <?xml version="1.0"
encoding="UTF-8"
standalone="yes"?>
    <Types
xmlns="http://schemas.openxmlform
ats.org/package/2006/content-
types">
        <Default Extension="rels"
ContentType="application/vnd.open
xmlformats-
package.relationships+xml"/>
        <Default Extension="xml"
ContentType="application/xml"/>
        <Override
PartName="/word/document.xml"
ContentType="application/vnd.open
xmlformats-
officedocument.wordprocessingml.d
ocument.main+xml"/>
    </Types>
```

/Listing

Der Inhalt in der Datei _rels erzeugt eine Verknüpfung, bzw. einen relativen Link auf das Hauptdokument. Über die _rels-Dateien, die in fast jedem Subordner enthalten sein müssen, werden die Einzeldateien referenziert und können so über die hier definierte ID angesprochen und manipuliert werden. Die Datei [Content_Types].xml gibt an, welches das Hauptdokument sein soll, da es auch möglich ist, mehrere Textdokumente in einer Datei zu halten. Zudem wird gleich ersichtlich, welche Parts innerhalb des Worddokuments zu erwarten sind, da hier den enthaltenen Elementen ihre Verwendungsart zugewiesen wird

Der korrekt erstellte Ordner mit den Dateien und Unterordnern muss nun komprimiert werden. Ein einfaches rechte Maus, „Senden an komprimierter zip-Ordner“ reicht leider nicht aus. Folgender Fehler würde erscheinen.

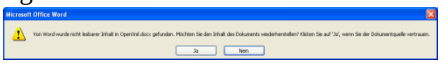


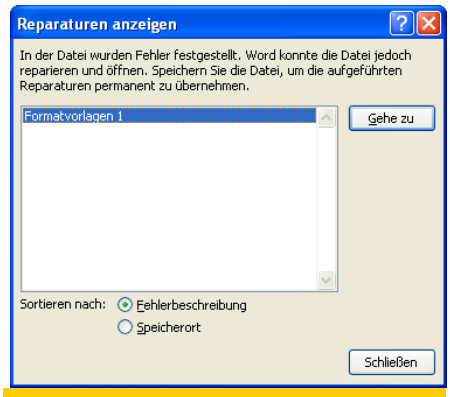
kohler_openxml_1.jpg

Abb. 3: Fehleranzeige beim Öffnen in Word

Dieser aussagekräftige Fehler erscheint übrigens in vielen Fällen, wenn etwas schief geht (z.B. falsche Xml-Syntax), unabhängig um welche Fehlerart es sich handelt.

Manchmal kann Word noch etwas retten und zeigt dies auch an, wie bei einem Test, bei dem ich versuchte eine Tabelle in einer Tabelle zu erzeugen. Leider konnte ich bisher noch nicht den Fehler entdecken, da die Xml-Syntax eigentlich korrekt sein sollte.





kohler_openxml_1.jpg

Abb. 3: Reperaturversuch von Word

Aber nun zum letzten Schritt.

[FEHLT NOCH]

Struktur des Frameworks

Wie schon erläutert stellt das neue Format lediglich ein, bzw. viele XML-Dokumente in einer zip-Datei dar. Für die gesamte Thematik der Komprimierung, Erzeugung von Referenzen und Ablage der Xml-Dateien wird das von Microsoft zur Verfügung gestellte Open XML Format SDK verwendet werden [1].

Das zentrale Element im Framework ist die Klasse WordDokument. Hier werden die grundsätzlichen Funktionen für die Erzeugung der verschiedenen Parts und deren Speicherung hinterlegt. Dem Worddokument sollen nach der Instanziierung die verschiedenen Inhaltselemente - wie auch aus anderen Frameworks bekannt - nach und nach hinzugefügt werden. Abschließend kann das Dokument mittels Angabe eines Speicherpfades erzeugt werden.

Für die Erzeugung eines einfachen Worddokuments werden folgende Parts benötigt:

Part		Anzahl
MainDocument	Hauptdokument	1
Styledfinition	Auslagerung der Formatierungen als eine Art Stylesheet	1 (können aber theoretisdch auch mehr sein)
Image	Nur	n
Header	Text und Formatierung der Header	n
Footer	Text und Formatierung der Footer	n

Das SDK übernimmt - wie schon gesagt - den Hauptteil für das Persistieren und Komprimieren der Daten. Worum wir uns jetzt kümmern müssen, ist die Erzeugung des richtigen XML-Strings für die verschiedenen Parts, also des eigentlichen Inhalts.

Daraus ergibt sich die Notwendigkeit die hinzu zu fügenden Elemente (Items) eines Parts in einer Collection zu halten. Für jede Parts-Collection wird ein typisiertes Add() in der Klasse WordDokument bereit gestellt. Das einzig öffentliche Add ist

für Elemente des Types Content reserviert, die direkt in die Hauptdatei als Basisnode „document.xml“ schreiben. Alle anderen überladenen Methoden Add() werden nur projektintern verwendet und deshalb als friend deklariert (siehe auch Übersicht Abb. 3). Bei der Funktion WordDocument.Create() werden die einzelnen Einträge der Collections je nach Struktur iteriert und analog wie in Listing 3 als typisierter Part hinzugefügt und somit durch das SDK erzeugt und gespeichert.

Listing 3

```
    Dim wordDoc As
Packaging.WordprocessingDocument
=
Packaging.WordprocessingDocument.
Create(path,
Microsoft.Office.DocumentFormat.O
penXml.WordprocessingDocumentType
.Document)
    //Part erzeugen und
hinzufügen
    Dim mainPart As
Microsoft.Office.DocumentFormat.O
penXml.Packaging.MainDocumentPart
= wordDoc.AddMainDocumentPart
    Using wordDoc
    //Erzeugen des Xml-Dokumentes
    //Eigener Code
    Dim docXml As String =
getXmlAsString()

    Dim stream1 As
System.IO.Stream = part.GetStream
    Dim utf8encoder1 As
System.Text.UTF8Encoding = New
System.Text.UTF8Encoding()
    Dim buf() As Byte =
utf8encoder1.GetBytes(docXml)
    stream1.Write(buf, 0,
buf.Length)

    End Using
```

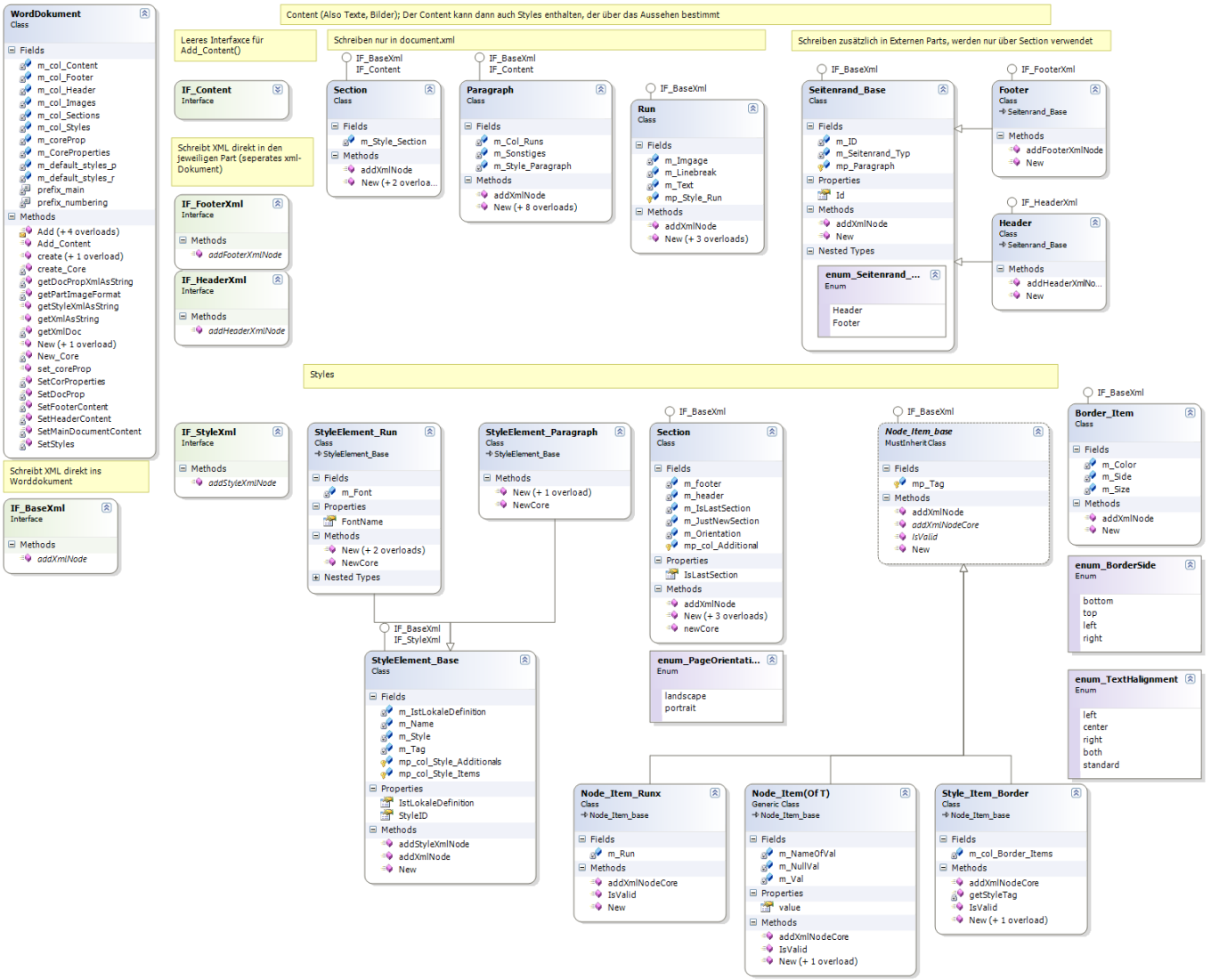
/Listing

Für jede Klasse bzw. Element, das Xml-Code in der „document.xml“-Datei erzeugen muss, wird das Interface IF_BaseXml mit einer einzigen Funktion definiert. Es soll dabei der Knoten herunter gereicht werden, an dem sich das jeweilige Element anhängen soll. Dies ist später bei den Styles wichtig, die von ihrem Parent-Knoten die Tag-Information benötigen.

```
    Public Interface IF_BaseXml
        Sub addXmlNode(ByRef
Rootnode As Xml.XmlNode, _
                       Optional
ByVal Prefix As String =
"http://schemas.openxmlformats.or
g/wordprocessingml/2006/main")
    End Interfac
```

Als Inhaltselemente

Die Inhaltselemente werden mittels add_content()



kohler_openxml_1.jpg
Abb. x: Klassenstruktur

Her mit dem Inhalt

In Listing 1 zeigt sich sehr schön die Texthierarchie, wie sie in Word verwendet wird. Es gibt den Paragraph (p; deutsch:Absatz), der Runs (r; deutsch: Zeichenabschnitte) und letztendlich Texte (t; deutsch: Zeichen) enthalten kann.

Der Paragraph stellt das Basiselement zur Strukturierung der Seite dar. Ein Run kann niemals ohne Paragraph auskommen, ebenso wie der Text nicht ohne Run interpretiert wird. Der Paragraph und der Run können Formatierungen enthalten. Der Text selbst dient nur zur Untergliederung, (z.B. Zeilenumbruch in extra Text-Tag). Da das Element Text relativ unwichtig ist und auch keine Formatierungen enthält, werden lediglich 2 Klassen erzeugt, die beide das Interface IF_BaseXml referenzieren.

Durch die Bedingung, dass ein run niemals als alleinstehendes Element im Body-Knoten stehen darf, also nur bestimmte Elemente direkt dem Dokument hinzugefügt werden dürfen, wurde das Dummy Interface IF_Content hinzugefügt. So kann garantiert werden, dass nur die richtigen Inhalt-Elemente (Paragraphen, Header…) dem Worddokument hinzugefügt werden. So kann ein run niemals direkt über die Funktion WordDokument.Add_Content() eingefügt werden, sonder muss immer über einen Paragraphen definiert werden.

```
Public Interface IF_Content

End Interface
```

Paragraphen

Der Paragraph ist der Container für runs, Bilder, Sections und somit das primäre Basiselement innerhalb der document.xml. Er enthält folgende Elemente

- Runs, die als Collection gehalten werden
- Styleelement, um den Pragraphen zu Formatieren

- Sonstiges, mit dem Interface IF_XmlBase, für z.B. Tabellen oder andere Elemente, die noch nicht bedacht wurden

Runs

Der Run wird folgende Membervariablen enthalten:

- Text, als der eigentliche Inhalt
- Styleelement, um den Text zu Formatieren
- Linebreak as boolen, da der Text bei mehreren runs innerhalb eines Paragraphen nicht automatisch umgebrochen wird
- Image, das jedoch nicht erläutert wird, da ich die Thematik Drawing auf Grund der Komplexität sehr pragmatisch gelöst habe

Formatierungen

Die Formatierung eines Elementes wie dem Paragraphen oder dem Run erfolgt als Childnode <rPr> bzw. <pPr> (also Grundelementname + Pr) im jeweiligen Inhaltsknoten<r> bzw. <p>

```
<w:r> // Run
  <w:rPr>
   […]'Formatierungen
  </w:rPr>
  <t>
    Hier kommt Text
..</t>
</w:r>
```

Innerhalb dieses Tags sind je nach Element verschiedene Formatierungen möglich. Exemplarisch seinen hier ein paar Elemente genannt.

Paragraph	Run
Margin	Font
Spacing	Fontsize
Border	Color
	Bold
	Italic
	Spacing
	BgColor

Die Formatierung kann auf zwei verschiedene Arten erfolgen:

1. Direkte Definition im Dokument

```
<w:r>
  <w:rPr>
    <w:b />
    <w:color w:val='#669966'
/>
  </ w:rPr>
</ w:r>
```

2. Auslagerung in einer eigenen Style-Datei und Einbindung mittels Referenz. Im Hauptdokument stellt es sich dann so dar:

```
<r>
  <w:rPr>
    <w:rStyle w:val='X' >
  </ w:rPr>
</ w:r>
</r>
```

Im Part-Dokument „Styles.xml“ werden dann die eigentlichen Style-Definitionen hinterlegt.

```
<?xml version="1.0"
encoding="UTF-8" standalone="yes" ?>
<w:styles
xmlns:w="http://schemas.openxmlfo
rmats.org/wordprocessingml/2006/m
ain">
- <w:style w:type="paragraph"
w:default="1" w:styleId="X">
<w:name w:val="x" />
<w:next w:val='Normal' />
<w:basedOn w:val='Normal' />
<w:qFormat />
<pPr>
   //eigentliche
Styledefinition
</pPr>
</w:style>
```

Der Tag <w:next /> wird verwendet, um festzulegen, welcher Paragraphenstyle nach z:b. einer Überschrift kommen soll. Mit der Eigenschaft <w:basedOn /> kann man eine Vererbungshierachrie aufbauen, so dass die nicht definierten Eigenschaften immer vom Basiselement stammen.

Um diese beiden verschiedenen Möglichkeiten der definition umzusetzen gibt es zwei New(). Wenn der Style ausgelagert werden soll, muss zusätzlich noch ein Name und das Worddokument übergeben werden.

Header/Footer

Header und Footer werden als eigene Parts definiert. Im Gegensatz zu den Styles, die in einem Part definert sind und somit in einer xml-Datei abgelegt werden, ist für jeden Header, bzw. Footer ein Part, also eine separate Xml-Datei notwendig. Im Hauptdokument wird der Verweis in einer ID mittels Header.addXmlNode (aus dem Intefrace IF_Xml_Base) abgelegt.

Der Inhalt eines Header kann, wie aus Word bekannt ist, eigentlich alles sein. D.h. der Paragraph ist hier auch wieder das Basiselement, in dem verschiedenen Elemente unter gebracht sein können. Der Inhalt wird in das externe Xml-Dokument mittels der Funktion AddHeaderXmlNode() (aus dem Interface IF_HeaderXml) geschrieben. Da Header und Footer sich bis auf die Bezeichnung nicht unterscheiden, wurden die Properties in das Basiselement Seitenrand_Base verlegt, das von den beiden Klassen Header und Footer geerbt wird.

Abschnitte

Da die Footer/Header Eigenschaften eines Abschnittes sind, können diese nur über die Section (Abschnitt) hinzugefügt werden. Zwar befindet sich die Section selbst wiederum nur in der Style-Definition des Paragraphen, jedoch wird sie auf

Grund ihrer Wichtigkeit als zweites Content-Element definiert.

```
<w:Pr>
  <w:sectPr>
    [Eigenschaften]
  </ w:sectPr>
< /w:Pr>
```

Sie kann jedoch auch als Parameter dem Paragraphen übergeben werde. Auch hier erfolgt wieder eine Trennung der Klassen in Content und Style, der die eigentlichen Eigenschaften enthält

Innerhalb eines Abschnittes kann man verschiedene Eigenschaften wie Anzahl Spalten, Abstand zwischen Spalten, Ausrichtung, … definieren. Dabei ist zu Berücksichtigen, dass diese Definition nicht im voraus gemacht wird, wie man es vielleicht erwarten würden, nach dem Motto „ab hier schauen die Seiten jetzt so aus“. Nein, es wird rückwirkend definiert - „Bis hierhin haben die Seiten bis hierhin das folgende Layout“. Der Abschnitt wird im übernächsten Absatz erläutert.

Worddokument erzeugen

Nachdem wir jetzt geduldig die grundlegendste Syntax erörter haben nun endlich zum ersehnten Dokument. Das Grundprinzip ist Listing 4 zu entnehmen, bei dem wir das oben mühsammit der Hand erzeugte „Hello World“ erzeugen.

Listing 4

```
Dim wordDoc As New
WordDokument


Dim P as Paragraph
P= new Paragraph("你好")
wordDoc.Add_Content(p)
wordDoc.create('C:/Test.docx')
```

/Listing

Das ganze kann man jetzt natürlich noch verfeinern, indem man erst den run mit Styles definiert und diesen dann dem Paragraphen übergibt, der auch wieder gestyled wird … .

Listing 5

```
Dim wordDoc As New
WordDokument
Dim p_style As New
Word.Style.StyleElement_Paragraph
(New
Word.Style.Style_Item_Border(
Word.Style.enum_BorderArt.P, 1,
"aaaaaa"),
Word.enum_TextHalignment.left,
10, 5, )


Dim r_style As New
Word.Style.StyleElement_Run("Aria
l", 12, "FF0000", False, True, ,
0, "ffffff")
Dim P as Paragraph
P= new Paragraph("你好",
r_style, p_style)
wordDoc.Add_Content(p)
wordDoc.create('C:/Test.docx')
```

/Listing

Demo

Um das bisher mögliche Spektrum an Funktionalität sichtbar zu machen liegt das erarbeite Framework als Quellcode mit einem kleinem Testprojekt bei. Das Open XML Format SDK 1.0 [1] muss wie immer noch als Referenz erneuert werden, wenn es nicht zufälligerweise im GAC liegt.

Mit jedem einzelnen Button (Abbildung X) wird jeweils ein Word-Dokument erzeugt und direkt über Process.Start() angesprochen. Von daher sollte ein MS-Word installiert sein, dass mit *.docx weiß umzugehen. Parallel dazu wird der XML-Inhalt der Dateien „document.xml“ und „styles.xml“ angezeigt.

Vieles ist mit Sicherheit noch ausbaufähig (Tabellen, Nummerierungen, ..). Jedoch sollte man gerade bei komplexeren Anforderungen (z.B. eingefügte Grafiken, die auf einem internen Excelsheet basieren) einen Blick in das neue SDK 2.0 werfen, ob hier nicht einfachere Lösungsansätze zu finden sind.

demo.jpg

Abb. : Demoprojekt

Ich kann nur hoffen, dass der Artikel gemundet hat, ohne allzu langatmig geworden zu sein und wünsche viel Spaß beim Rumprobieren und Erweitern des kleinen Frameworks.

再见 (zai jian) Tschüss, Auf Wiedersehen.

jk.jpg

Jacques Kohler arbeitet als Senior Software Developer bei IngSoft GmbH in Nürnberg. Seine Schwerpunkte liegen in der kundenspezifischen Entwicklung von .Net-Lösungen im On- und Offlinebereich. Sie erreichen Ihn unter jacques.kohler@ingsoft.de.

Links & Literatur

[1] Open XML Format SDK 2.0 (Stand 08.12.2009): http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=c6e744e5-36e9-45f5-8d8c-331df206e0d0

[2] Open XML Format SDK 1.0 (Stand 25.08.2008): http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=ad0b72fb-4a1d-4c52-bdb5-7dd7e816d046

[3] ECMA-376 Part 4. Der Standard, bzw. die Syntax in seiner Vollständigkeit: http://www.ecma-international.org/publications/standards/Ecma-376.htm

[4] e-Book „OpenXml explained“ von Wouter van Vugt: http://openxmldeveloper.org/articles/1970.aspx

[4] MSDN, Ressourcen rund um das OpenXml-Format: http://msdn.microsoft.com/en-us/office/bb265236.aspx