AODL-Lightweight

diub - Dipl.-Ing. Uwe Barth

04.05.2021

Inhalt

	AODL-Lightweight	1
1	Hintergrund-Wissen	2
1.1	Eigenarten des ursprünglichen Codes	2
1.1.1	XML-Dokument	2
1.1.2	Brute-force oder es funktioniert nicht (warum auch immer!)	2
1.1.3	Style	2
1.2	Eigene Typen	3
1.3	Usings	3
2	Beispiele	4
2.1	Neues Dokument anlegen	4
2.2	Paragraphen erzeugen	4
2.3	Leerzeichen, WhiteSpace, Tab	4
2.4	Querverweise: Bookmark, PageRef	5
2.5	Seitennummer, Seitenzahl: PageNo, PageCount	7

Dieses ***Dokument*** wurde mit

Documenter

erstellt und steht daher mit gleichem Inhalt als

- WebSeite
- Open-Document-Text-Datei (*.odt)
- PDF-Datei (via Open-Document-Text oder XeLaTeX)
- ePub
- Windows-Hilfe
- XeLaTeX-Datei

zur Verfügung!

Documenter **ist eine Entwicklung von** ***diub*** **zur Erstellung von Dokumentationen aus einer Quelle mit unterschiedlichen Zielformaten.**

Empfohlene ePub-Reader
ePub-Reader gibt es massenhaft. Nur leider sind die meisten davon schlicht unbrauchbar; falsche Schriften, kaputte Formatierungen usw. sind die Tagesordnung. Am schlimmsten ist jedoch die Unfähigkeit zu echter Skalierung, für Normalanwender die Änderung der 'Schriftgröße'.

Nachstehend die von mir bevorzugten Reader mit (weitgehend) korrekter Darstellung.

- Android
  GitdenReader (im PlayStore), schlicht und einfach und es funktioniert (fast) alles
- Windows
  Digital Editions 2.x (Adobe), leider - wie alle anderen auch - keine Skalierung der Bilder

1 Hintergrund-Wissen

1.1 Eigenarten des ursprünglichen Codes

1.1.1 XML-Dokument

Der originale Quellcode **operiert** (überwiegend) **direkt auf dem XML-Dokument**, welches später exportiert wird. Es müssen immer wieder korrekte *OpenDocumentText* - Anweisungen zusammengebastelt werden. Die Bibliothek nimmt einem da keine Arbeit ab.

Außerdem sind die Aufgaben eines Konstruktors über mehrere Funktionen verteilt: die Get/Set-Funktionen der Properties übernehmen die Initialisierung. Notwendige Korrekturen sind dadurch sehr umständlich, siehe Leerzeichen, WhiteSpace, Tab, Seite 4.

1.1.2 Brute-force oder es funktioniert nicht (warum auch immer!)

Der nachstehende Code funktioniert nicht wie erwartet. Zwar tritt **kein .Net-Fehler** auf, aber die Zuweisung des *Style* bleibt ohne Wirkung.

- Das gilt für alle Zuweisungen im ***AODL***-Namensraum.
- Und über alle Objekthierarchien. Daher auch der langatmige Code zum Beispiel in Paragraphen erzeugen, Seite 4.

Die Ursache ist mir nicht bekannt.

```
Paragraph p = new Paragraph (...);
ParagraphStyle pstyle = new ParagraphStyle(....);
// Funktioniert nicht!
p.ParagraphStyle = pstyle;
```

Dieser Code funktioniert hingegen.

```
Paragraph p = new Paragraph (...);
// Funktioniert!
p.ParagraphStyle = new ParagraphStyle(....);
```

1.1.3 Style

Für jeden angelegten Style muss ein ein-eindeutiger Name verwendet werden. Andernfalls geht es im Dokument später drunter und drüber.

Ich verwende deswegen überwiegend generische Namen.

1.2 Eigene Typen

Die Beispiele verwenden nicht weiter definierte Datentypen. Das tut der Lesbarkeit der Beispiele jedoch keinen Abbruch.

Text

Text ist eine Eigenentwicklung, im Prinzip eine Kreuzung von *StringBuilder* und *String*.

ContentNode

Ein Knoten in *Documenter*, der einen Inhalt inkl. Formatierungsangaben in Form von Snippet, Seite 3s enthält.

Snippet

Ein Snippet ist ein Schnipsel Inhalt in einem ContentNode, Seite 3. Innerhalb eines Snippet gibt es nur

- eine Formatierung
- einen Inhalts-Typen (Text, Bild, Querverweis, ...)

1.3 Usings

```
using AODL.Document.Content.Draw;
using AODL.Document.Content.Text;
using AODL.Document.Content.Text.Indexes;
using AODL.Document.Content.Text.TextControl;
using AODL.Document.Styles;
using AODL.Document.Styles.MasterStyles;
using AODL.Document.TextDocuments;
```

2 Beispiele

Die Beispiele entstammen meinem Projekt *Documenter*.

2.1 Neues Dokument anlegen

Der Konstruktoraufruf erledigt den (sonst) notwendigen *New*-Aufruf gleich mit.

```
// Initialisieren
OpenDocumentSequencer.OpenDocument = new TextDocument ();
```

2.2 Paragraphen erzeugen

Erzeugt ein *Paragraph* Objekt mit einem eindeutigem generischen Namen für den *Style*.

```
private Paragraph ParagraphFormat (ContentNode Paragraph) {
    Text text;
    string style_name;
    AODL.Document.Content.Text.Paragraph p;

    text = new Text ("PF");
    text.Append ("A", Paragraph.HorizontalAlignment.ToString (),
        "T", Paragraph.ContentType);
    text = text.RemoveAllChar (false, ' ', ',');
    style_name = text.ToString ();

    p = new AODL.Document.Content.Text.Paragraph (OpenDocument, style_name);
    if (Paragraph.HorizontalAlignment == HorizontalAlignment.Right)
        p.ParagraphStyle.ParagraphProperties.Alignment = "end";
    if (Paragraph.HorizontalAlignment == HorizontalAlignment.Center)
        p.ParagraphStyle.ParagraphProperties.Alignment = "center";
    if (Paragraph.HorizontalAlignment == HorizontalAlignment.Stretch)
        p.ParagraphStyle.ParagraphProperties.Alignment = "justify";
    if (Paragraph.ContentType == ContentType.Capitals)
        p.ParagraphStyle.TextProperties.Capitalize = "small-caps";
    return p;
}
```

2.3 Leerzeichen, WhiteSpace, Tab

Der Code wurde von mir stark verändert: Ohne Angabe eines *Style* wird sonst nämlich der Standard-Font des Absatzes verwendet.

Das Tab-Zeichen harrt noch der Korrektur.

ODT MIST

Ein einzelnes Leerzeichen im Text darf als Leerzeichen kodiert werden.

"Ähhhh, ja? Und?"

Führende, nachstehende oder Gruppen von 2 oder mehr Leerzeichen müssen so im XML-Dokument landen, alle anderen dürfen auch:

```
<text:span text:style-name="STYLENAME"
xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0">
    <text:s text:c="1" />
</text:span>
```

Dabei gibt der Wert hinter 'c' die Anzahl an.

Wer immer sich diese Idiotie für das ODT-Format ausgedacht hat: ich kann ihn absolut nicht leiden!

Vielleicht ist es ja nur ein Vorurteil von mir, aber Amis (gemeint sind hier speziell Entwickler in den USA) sind scheinbar einfach unfähig. (MS hat - nach meiner Erinnerung - ja auch so etwas einmal über (seine eigenen) Entwickler gesagt; als Begründung dafür, warum .Net keine Mehrfach-Ableitung kann.)

2.4 Querverweise: Bookmark, PageRef

PageRef ist neu von mir.

Das Einfügen eines Seitenverweises ist nicht ganz trivial. Es erfolgt in 3 Schritten:

- Einfügen des *Bookmark* mit einer ID.
- Einfügen des *PageRef*, *Xlink*, **auch mehrfach auf dieselbe *Bookmark***.
- Da ein *PageRef*, *XLink* ja durchaus vor dem referenzierten *Bookmark* liegen kann: ganz am Ende Korrektur aller Querverweistexte in den *PageRef* Textstellen.

Bedeutet: ihr Code braucht eine Merk-Liste für die erzeugten *PageRef* inklusive der einzufügenden Texte.

Das Beispiel zeigt die Sonderbehandlung von Querverweis-Snippets und das Merken der Referenzen in während des Einfügens in ein *Paragraph*-Objekt.

- *XLink* ist ein Hyptertext-Querverweis; also die Stelle mit dem Text.
- *hash* enthält die ID.
- *back_references* ist die Liste der gemerkten IDs.
- *references* die Liste (für mehrfache *PageRef*) der Querverweis-Stellen.

```
private Dictionary<string,List< BackReference>> back_references
    = new Dictionary<string,List<BackReference>>();

private void CycleSnippetList (Paragraph Paragraph,
    ContentNode ContentNode, List<Snippet> Snippets) {
    int i;
    string hash;
    List<IText> itl;
    PageRef page_ref;
    XLink xlink;
    List<BackReference> references;

    foreach (Snippet snippet in Snippets) {
        if (snippet.IsLink) {
            hash = snippet.Value.ToUpperInvariant ();
            if (back_references.ContainsKey (hash)) {
                references = back_references [hash];
            } else {
                references = new List<BackReference> ();
                back_references.Add (hash, references);
            }
            xlink = new XLink (OpenDocument, "#" + hash, hash);
            i = Paragraph.TextContent.Add (xlink);
            Paragraph.TextContent.Add (
                new SimpleText (OpenDocument, ", Seite ")
            );
            page_ref = new PageRef (OpenDocument, hash);
            Paragraph.TextContent.Add (page_ref);
            references.Add (
                new BackReference () { Index = i, Paragraph = Paragraph }
            );
            continue;
        }
        itl = TextFormat (snippet);
        Paragraph.TextContent.Add (itl);
    }
}
```

Dieser Code schreibt die gemerkten Texte in die *PageRef*-Stellen.

```
public void CompleteBackReferences () {
    LinkInfo li;
    XLink xlink;
    List<BackReference> references;

    foreach (KeyValuePair<string,
        List<BackReference>> backref in back_references) {
        references = backref.Value;
        foreach (BackReference item in references) {
            if (!document_info.links.ContainsKey (backref.Key))
                continue;
            li = document_info.links [backref.Key];
            xlink = item.Paragraph.TextContent [item.Index] as XLink;
            xlink.Text = li.title;
```

```
            }
        }
}
```

2.5 Seitennummer, Seitenzahl: PageNo, PageCount

Neu von mir.

```
// Seitenzahlen:   1 /  26
p.TextContent.Add (new PageNumber (OpenDocument));
p.TextContent.Add (new SimpleText (OpenDocument, " / "));
p.TextContent.Add (new PageCount (OpenDocument));
```