

## Wprowadzenie do Open XML Format SDK 2.0

Microsoft w tym (2009) roku opublikował kolejną wersję pakietu SDK do manipulacji dokumentów w formacie OpenXML. Pakiet ten ma za zadanie ułatwić tworzenie i edycję plików w formacie OpenXML z poziomu języka programowania (np. C#), automatyzując standardowe operacje jakie zwykle implementować muszą programiści tworzący aplikacje wykorzystujące ten format. W tym artykule chciałbym przedstawić sposoby wykorzystania tego pakietu. Oraz przybliżyć czytelnikom jak można wykreować dokument typu DOCX z

Microsoft w tym roku opublikował kolejną wersję pakietu SDK do manipulacji dokumentów w formacie OpenXML. Pakiet ten ma za zadanie ułatwić tworzenie i edycję plików w formacie OpenXML z poziomu języka programowania (np. C#), automatyzując standardowe operacje jakie zwykle implementować muszą programiści tworzący aplikacje wykorzystujące ten format. Pakiet można pobrać ze stron Microsoft'u: <http://www.microsoft.com/downloads/details.aspx?FamilyID=C6E744E5-36E9-45F5-8D8C-331DF206E0D0&displaylang=en>.

Więcej informacji na temat pakietu i formatu Open XML można znaleźć tutaj:

- <http://msdn.microsoft.com/en-us/library/dd440953.aspx>  
- pierwsza część (z trzech) artykułu poświęcona podstawom tworzenia dokumentów OpenXML z wykorzystaniem języka C#.
- <http://blogs.msdn.com/ericwhite/archive/2008/09/06/announcing-the-first-ctp-of-open-xml-sdk-v2.aspx>  
- pewne informacje na temat tego SDK i co w nim nowego.
- <http://openxmldeveloper.org/>  
- portal poświęcony temu standardowi.
- [http://msdn.microsoft.com/en-us/library/bb448854\(office.14\).aspx](http://msdn.microsoft.com/en-us/library/bb448854(office.14).aspx) - dokumentacja do tego pakietu.

W tym artykule chciałbym się skupić na pokazaniu kilku przykładów z wykorzystania tego pakietu.

Aby stworzyć dowolną aplikację wykorzystującą Open Xml Format SDK, do projektu należy dodać następujące referencje:

- DocumentFormat.OpenXml
- WindowsBase

W kodzie dołączamy następujące przestrzenie nazw:

- DocumentFormat.OpenXml;
- DocumentFormat.OpenXml.Packaging;
- DocumentFormat.OpenXml.Wordprocessing;

Teraz zobaczymy funkcję realizującą proste "Hello World" w OpenXML:

## [Kod C#]

```
private void HelloWorld(string documentFileName)
{
    // Create a Wordprocessing document.
    using (WordprocessingDocument myDoc = WordprocessingDocument.Create(documentFileName, WordprocessingDocumentType.Document))
    {
        // Add a new main document part.
        MainDocumentPart mainPart = myDoc.AddMainDocumentPart();
        //Create Document tree for simple document.
        mainPart.Document = new Document();
        //Create Body (this element contains other elements that we want to include
        Body body = new Body();
        //Create paragraph
        Paragraph paragraph = new Paragraph();
        Run run_paragraph = new Run();
        // we want to put that text into the output document
        Text text_paragraph = new Text("Hello World!");
        //Append elements appropriately.
        run_paragraph.Append(text_paragraph);
        paragraph.Append(run_paragraph);
        body.Append(paragraph);
        mainPart.Document.Append(body);
        // Save changes to the main document part.
        mainPart.Document.Save();
    }
}
```

Powyższa funkcja tworzy dokument w pliku podanym przez użytkownika. Do dokumentu dodany jest paragraf który zawiera tekst "Hello World!". W ostatnim kroku dokument jest zapisywany.

Warto również przyjrzeć się zawartości generowanej przez tą funkcję i zapisanej w języku WordprocessingML:

## [Kod XML]

```
<?xml version="1.0" encoding="utf-8"?>
<w:document xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Hello World!</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

Jak widać w powyższym kodzie XML widać duże powiązanie pomiędzy kodem napisanym w C#, a powyższym. Zadeklarowany jest dokument (document = WordprocessingDocument) wewnątrz którego jest część Body (body), w niej znajduje się paragraf (p = Paragraph) z elementem typu Run (r) i odpowiednim tekstem (t = Text).

A jak tą funkcję wykorzystać? Poniższy przykład pyta użytkownika o plik, do którego ma być zapisany tekst, wywołuje wcześniej stworzoną funkcję i otwiera przygotowany plik przy pomocy Word (lub innej ustawionej w systemie przeglądarki plików docx).

#### [Kod C#]

```
SaveFileDialog mySaveFileDialog=new SaveFileDialog();
mySaveFileDialog.Filter = "Word 2007 file (DOCX)|*.docx";
//save dialog display:
if (mySaveFileDialog.ShowDialog() == DialogResult.OK)
{
    //call creation of HellowWord document
    HelloWorld(mySaveFileDialog.FileName);
    // let's open this document in Word:
    Process.Start(mySaveFileDialog.FileName);
}
```

Teraz można iść nieco dalej i dołożyć elementy związane z formatowaniem tekstu, czyli zajmę się teraz stylami. Założmy, że chcemy przygotować dokument, który będzie się składał z dwóch linijek tekstu. Pierwsza linijka będzie nagłówkiem akapitu, druga tekstem akapitu. To jak przygotować zwykły tekst (tak jak tutaj w drugiej linijce) pisałem o tym już wcześniej, tym razem napiszę jak przygotować tą pierwszą (odpowiednio sformatowaną) linijkę.

Pierwszą czynnością jaką należy wykonać, to trzeba dodać do tworzonego dokumentu część związaną z definicją stylów (dodajemy ją do głównego elementu dokumentu):

#### [Kod C#]

```
WordprocessingDocument myDoc = WordprocessingDocument.Create(documentFileName, WordprocessingDocumentType.Document)
MainDocumentPart mainPart = myDoc.AddMainDocumentPart();
StyleDefinitionsPart stylePart = mainPart.AddNewPart<StyleDefinitionsPart>();
```

Definiujemy teraz czcionkę (Font), którą chcemy wykorzystać (w tym przykładzie: czerwony i pogrubiony Arial, rozmiaru 28):

#### [Kod C#]

```
// we have to set the properties
RunProperties rPr = new RunProperties();
Color color = new Color() { Val = "FF0000" }; // the color is red
RunFonts rFont = new RunFonts();
rFont.Ascii = "Arial"; // the font is Arial
rPr.Append(color);
rPr.Append(rFont);
rPr.Append(new Bold()); // it is Bold
rPr.Append(new FontSize() { Val = 28 }); //font size (in 1/72 of an inch)
```

Teraz czas na definicję stylu, będzie się on nazywać "My Heading 1", a jego identyfikator (używany później w dokumencie) to "MyHeading1". Oto definicja:

[Kod C#]

```
//creation of a style
Style style = new Style();
style.StyleId = "MyHeading1"; //this is the ID of the style
style.Append(new Name() { Val = "My Heading 1" }); //this is name
style.Append(new BasedOn() { Val = "Heading1" }); // our style based on Normal style
style.Append(new NextParagraphStyle() { Val = "Normal" }); // the next paragraph is Normal type
style.Append(rPr); //we are adding properties previously defined
```

W tym momencie nadszedł czas na dodanie stworzonego stylu do dokumentu i zapisanie części związanej ze stylami:

[Kod C#]

```
// we have to add style that we have created to the StylePart
stylePart.Styles = new Styles();
stylePart.Styles.Append(style);
stylePart.Styles.Save(); // we save the style part
```

Styl mamy już stworzony, teraz trzeba go przypisać do jakiegoś paragrafu. Służy do tego klasa ParagraphProperties, której trzeba przypisać odpowiednie ParagraphStyleId. Przykład poniżej:

[Kod C#]

```
Paragraph heading = new Paragraph();
Run heading_run = new Run();
Text heading_text = new Text("This is Heading");
ParagraphProperties heading_pPr = new ParagraphProperties();
heading_pPr.ParagraphStyleId = new ParagraphStyleId() { Val = "MyHeading1" }; // we set the style
heading.Append(heading_pPr);
heading_run.Append(heading_text);
heading.Append(heading_run);
```

Teraz można już cieszyć się efektem naszych prac (przykład jest na załączonym rysunku), zobaczymy jeszcze całą procedurę która generuje ten dokument:

[Kod C#]

```
private void HelloWorld_withStyle(string documentFileName)
{
    // Create a Wordprocessing document.
    using (WordprocessingDocument myDoc = WordprocessingDocument.Create(documentFileName, WordprocessingDocumentType.Document))
    {
        // Add a new main document part.
        MainDocumentPart mainPart = myDoc.AddMainDocumentPart();
```

```

//Add new style part
StyleDefinitionsPart stylePart = mainPart.AddNewPart();
// we have to set the properties
RunProperties rPr = new RunProperties();
Color color = new Color() { Val = "FF0000" }; // the color is red
RunFonts rFont = new RunFonts();
rFont.Ascii = "Arial"; // the font is Arial
rPr.Append(color);
rPr.Append(rFont);
rPr.Append(new Bold()); // it is Bold
rPr.Append(new FontSize() { Val = 28 }); //font size (in 1/72 of an inch)
//creation of a style
Style style = new Style();
style.StyleId = "MyHeading1"; //this is the ID of the style
style.Append(new Name() { Val = "My Heading 1" }); //this is name
style.Append(new BasedOn() { Val = "Heading1" }); // our style based on Normal style
style.Append(new NextParagraphStyle() { Val = "Normal" }); // the next paragraph is Normal type
style.Append(rPr); //we are adding properties previously defined
// we have to add style that we have created to the StylePart
stylePart.Styles = new Styles();
stylePart.Styles.Append(style);
stylePart.Styles.Save(); // we save the style part

//Create Document tree for simple document.
mainPart.Document = new Document();
//Create Body (this element contains other elements that we want to include
Body body = new Body();
//Create paragraph
Paragraph paragraph = new Paragraph();
Run run_paragraph = new Run();
// we want to put that text into the output document
Text text_paragraph = new Text("Hello World!");
//Append elements appropriately.
run_paragraph.Append(text_paragraph);
paragraph.Append(run_paragraph);
Paragraph heading = new Paragraph();
Run heading_run = new Run();
Text heading_text = new Text("This is Heading");
ParagraphProperties heading_pPr = new ParagraphProperties();
heading_pPr.ParagraphStyleId = new ParagraphStyleId() { Val = "MyHeading1" }; // we set the style
heading.Append(heading_pPr);
heading_run.Append(heading_text);
heading.Append(heading_run);
body.Append(heading);
body.Append(paragraph);
mainPart.Document.Append(body);
// Save changes to the main document part.
mainPart.Document.Save();
}
}

```

A co z kodem zapisanym przy pomocy WordprocessingML? W tym przypadku generowane są dwa pliki, jeden zawierający definicję stylu:

**[Kod XML]**

```

<?xml version="1.0" encoding="utf-8"?>
<w:styles xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:style w:styleId="MyHeading1">
    <w:name w:val="My Heading 1" />
    <w:basedOn w:val="Heading1" />
    <w:next w:val="Normal" />
    <w:rPr>
      <w:color w:val="FF0000" />
      <w:rFonts w:ascii="Arial" />
      <w:b />
      <w:sz w:val="28" />
    </w:rPr>
  </w:style>
</w:styles>

```

oraz drugi z definicją z konkretnym dokumentem:

**[Kod XML]**

```
<?xml version="1.0" encoding="utf-8"?>
<w:document xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p>
      <w:pPr>
        <w:pStyle w:val="MyHeading1" />
      </w:pPr>
      <w:r>
        <w:t>This is Heading</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

Jak widać w tym przypadku otrzymujemy kod WordProcessingML bardzo podobny do tego z poprzedniego przykładu, jednak dodana została specjalna właściwość (pPr = ParagraphProperties), w której został wybrany odpowiedni styl (pStyle = ParagraphStyleId), który jest zdefiniowany w innym pliku.

Zajmijmy się teraz wyrównaniem tekstu: do lewej, do prawej, do środka i wyjustowaniem tekstu. Jak można ustawiać wyrównanie tekstu? W tym celu należy ustawić odpowiednie właściwości paragrafu dla stylu, który chcemy zmienić lub paragrafu w którym chcemy ustawić wyrównanie. Wykorzystujemy więc właściwość **Justification** klasy **ParagraphProperties**, której należy nadać wartość wchodzącą w skład elementu wyliczeniowego: **JustificationValues Enumeration**, o następujących elementach:

- Center: Center (Equation)
- CenterGroup: Centered as Group (Equations)
- Left: Left Justification
- Right: Right

Zakończmy więc teorię i przejdźmy do kodu, który będzie krótki i łatwy. Zakładając że na przykład chcemy wycentrować dany tekst (w paragrafie, bądź przy użyciu stylu) wystarczy wykonać następujący kod:

**[Kod C#]**

```
paragraph_or_style.Append( new ParagraphProperties(
    new Justification() { Val = JustificationValues.Center } ) );
```

Kod ten powoduje dodanie do dokumentu, napisanego w WordProcessingML, następującego kodu XML:

**[Kod XML]**

```
<w:jc w:val="center"/>
```

Tutaj ważna jest jeszcze jedna kwestia, a mianowicie odpowiednie formatowanie powinno być dodane do elementu paragrafu na początku i koniecznie przed dodaniem do niego tekstu, czyli np. oczekiwany efekt uzyskamy wykonując kod:

**[Kod C#]**

```
Paragraph paragraph = new Paragraph();
Run run_paragraph = new Run();
// we want to put that text into the output document
Text text_paragraph = new Text( "Hello World!" );
//Append elements appropriately.
run_paragraph.Append( text_paragraph );
paragraph.Append( new ParagraphProperties(
    new Justification() { Val = JustificationValues.Center } ) );
paragraph.Append( run_paragraph );
```

Natomiast tekst nie zostanie sformatowany, gdy do paragrafu najpierw zostałby dodany

element Run, a dopiero później ParagraphProperties.

Zajmijmy się teraz tworzeniem tabel przy pomocy Open XML Format SDK. Dzięki wygodnemu API jakie oferuje Open XML Format SDK tworzenie tabel jest bardzo proste i polega na dodaniu do głównego dokumentu elementu typu Table, w którym następnie osadzamy elementy typu TableRow i TableCell. Najprostszy przykład takiej tabeli wygląda następująco:

**[Kod C#]**

```
Body body = new Body();
Table table = new Table(new TableRow(new TableCell(
    new Paragraph(new Run(new Text("Hello World!"))))););
body.Append(table);
```

Teraz w najbliższym przykładzie stworzymy tabelkę z tabliczką mnożenia (zupełnie taką jak w szkole ;)):

Multiplication table										
*	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Na początku pokazałem jak przygotować prostą tabelkę, poprzez osadzanie elementów komórki i wiersza w tabeli. Teraz, w przykładzie też będziemy osadzać w sobie obiekty TableRow i TableCell, ale dodatkowo zadbamy jeszcze o krawędzie i ustawimy by komórka w pierwszym wierszu tabeli rozszerzona była na wszystkie kolumny poniżej.

Zacznijmy więc do ustawienia obramowania, dla całej tabeli ustawimy proste pojedyncze obramowanie, w tym celu wykorzystujemy klasy: TableProperties i TableBorders. Przykład poniżej:

**[Kod C#]**

```
Table table = new Table();
TableProperties tblPr = new TableProperties();
TableBorders tblBorders = new TableBorders();
tblBorders.TopBorder = new TopBorder();
tblBorders.TopBorder.Val = new EnumValue<BorderValues>(BorderValues.Single);
tblBorders.BottomBorder = new BottomBorder();
tblBorders.BottomBorder.Val = new EnumValue<BorderValues>( BorderValues.Single);
tblBorders.LeftBorder = new LeftBorder();
tblBorders.LeftBorder.Val = new EnumValue<BorderValues>(BorderValues.Single);
tblBorders.RightBorder = new RightBorder();
tblBorders.RightBorder.Val = new EnumValue<BorderValues>(BorderValues.Single);
tblBorders.InsideHorizontalBorder = new InsideHorizontalBorder();
tblBorders.InsideHorizontalBorder.Val = BorderValues.Single;
```

```
tblBorders.InsideVerticalBorder = new InsideVerticalBorder();
tblBorders.InsideVerticalBorder.Val = BorderValues.Single;
tblPr.Append(tblBorders);
table.Append(tblPr);
```

Zajmijmy się teraz pierwszym wierszem i jego komórką, dla której musimy ustawić cechę by była rozszerzona na 11 kolumn znajdujących się niżej. W tym celu ustawiamy odpowiednie właściwości komórki (TableCellProperties), a dokładniej właściwość GridSpan.

#### [Kod C#]

```
tr = new TableRow();
tc = new TableCell(new Paragraph(new Run(new Text("Multiplication table"))));
TableCellProperties tcp=new TableCellProperties();
GridSpan gridSpan=new GridSpan();
gridSpan.Val=11;
tcp.Append(gridSpan);
tc.Append(tcp);
tr.Append(tc);
```

Teraz zostają już do wykonania tylko proste wyliczenia i ostatecznie otrzymujemy funkcję

#### [Kod C#]

```
public void HelloWorld_table(string docName)
{
    // Create a Wordprocessing document.
    using (WordprocessingDocument myDoc = WordprocessingDocument.Create(docName, WordprocessingDocumentType.Document))
    {
        // Add a new main document part.
        MainDocumentPart mainPart = myDoc.AddMainDocumentPart();
        //Create DOM tree for simple document.
        mainPart.Document = new Document();
        Body body = new Body();
        Table table = new Table();
        TableProperties tblPr = new TableProperties();
        TableBorders tblBorders = new TableBorders();
        tblBorders.TopBorder = new TopBorder();
        tblBorders.TopBorder.Val = new EnumValue
(BorderValues.Single);
        tblBorders.BottomBorder = new BottomBorder();
        tblBorders.BottomBorder.Val =new EnumValue( BorderValues.Single);
        tblBorders.LeftBorder = new LeftBorder();
        tblBorders.LeftBorder.Val = new EnumValue(BorderValues.Single);
        tblBorders.RightBorder = new RightBorder();
        tblBorders.RightBorder.Val = new EnumValue(BorderValues.Single);
        tblBorders.InsideHorizontalBorder = new InsideHorizontalBorder();
        tblBorders.InsideHorizontalBorder.Val = BorderValues.Single;
        tblBorders.InsideVerticalBorder = new InsideVerticalBorder();
        tblBorders.InsideVerticalBorder.Val = BorderValues.Single;
        tblPr.Append(tblBorders);
        table.Append(tblPr);
        TableRow tr;
        TableCell tc;
        //first row - title
        tr = new TableRow();
        tc = new TableCell(new Paragraph(new Run(new Text("Multiplication table"))));
        TableCellProperties tcp=new TableCellProperties();
        GridSpan gridSpan=new GridSpan();
        gridSpan.Val=11;
        tcp.Append(gridSpan);
        tc.Append(tcp);
        tr.Append(tc);
        table.Append(tr);
        //second row
        tr = new TableRow();
        tc = new TableCell();
        tc.Append(new Paragraph(new Run(new Text("**"))));
        tr.Append(tc);
        for (int i = 1; i <= 10; i++)
        {
            tr.Append(new TableCell(new Paragraph(new Run(new Text(i.ToString()))));
        }
        table.Append(tr);
        for (int i = 1; i <= 10; i++)
        {
            tr = new TableRow();
            tr.Append(new TableCell(new Paragraph(new Run(new Text(i.ToString()))));
            for (int j = 1; j <= 10; j++)
            {
                tr.Append(new TableCell(new Paragraph(new Run(new Text((i*j).ToString()))));
            }
            table.Append(tr);
        }
        //appending table to body
```

```

body.Append(table);
// and body to the document
mainPart.Document.Append(body);
// Save changes to the main document part.
mainPart.Document.Save();
}
}

```

W wyniku działania tej funkcji zostanie wygenerowana następująca zawartość zapisana w języku WordprocessingML:

#### [Kod XML]

```

<?xml version="1.0" encoding="utf-8" ?>
<w:document xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:tbl>
      <w:tblpr>
        <w:tblborders>
          <w:top w:val="single" />
          <w:left w:val="single" />
          <w:bottom w:val="single" />
          <w:right w:val="single" />
          <w:insideh w:val="single" />
          <w:insidev w:val="single" />
        </w:tblborders>
      </w:tblpr>
      <w:tr>
        <w:tc>
          <w:p>
            <w:r>
              <w:t>Multiplication table</w:t>
            </w:r>
          </w:p>
          <w:tcpr>
            <w:gridspan w:val="11" />
          </w:tcpr>
        </w:tc>
      </w:tr>
      <w:tr>
        <w:tc>
          <w:p>
            <w:r>
              <w:t>*</w:t>
            </w:r>
          </w:p>
        </w:tc>
      <w:tr>
        <w:tc>
          <w:p>
            <w:r>
              <w:t>1</w:t>
            </w:r>
          </w:p>
        </w:tc>
      <!-- many cells and rows declarations -->
    </w:tbl>
  </w:body>
</w:document>

```

Czyli mamy tutaj takie elementy jak tabela (tbl = Table ) z odpowiednimi właściwościami (tblpr = TableProperties ) i ramkami (tblborders = TableBorders ), w której znajdują się wiersze (tr = TableRow ) i kolumny (tc = TableCell ).

Jak widać na powyższych przykładach jest duży związek pomiędzy wykorzystywanymi elementami pakietu Open Document Format SDK a konkretnym elementami zapisywanymi w języku WordProcessingML, dlatego jeżeli nie wiemy jak coś przygotować przy pomocy SDK, można zawsze przygotować odpowiedni plik przy pomocy edytora tekstu (np. Word), a następnie sprawdzić, co rzeczywiście zostało w nim utworzone (w następnej sekcji opowiem jak to zrobić).

Na koniec przeanalizujemy jak zbudowane są pliki oparte o format Open XML? Każdy z nich jest zestawem plików (głównie XML), które następnie są spakowane przy pomocy standardowego algorytmu zip, co w efekcie daje plik wynikowy. Dlatego można dowolnemu plikowi opartemu o ten format zmienić rozszerzenie (lub dodać kolejne) na zip, a następnie rozpakować zawartość tak spreparowanego archiwum. Tak na marginesie dodam, że w przypadku formatu rywała, czyli ODF (Open Document Format) idea pakowania algorytmem zip jest dokładnie taka sama.

Zobaczymy na przykład pliku Test.docx, któremu dodałem rozszerzenie zip i w efekcie otrzymałem plik Test.docx.zip, którego zawartość przedstawiona jest poniżej:

Test.docx.zip

```
| [Content_Types].xml
|
+---docProps
|   app.xml
|   core.xml
|
+---word
| | document.xml
| | fontTable.xml
| | numbering.xml
| | settings.xml
| | styles.xml
| | webSettings.xml
| |
| +---media
| | image1.gif
| | image2.gif
| | image3.gif
| | image4.gif
| | image5.gif
| | image6.gif
| |
| +---theme
| | theme1.xml
| |
| \---_rels
|   document.xml.rels
|
\---_rels
    .rels
```

Najbardziej oczywiście interesującym nas plikiem zwykle jest plik: word\document.xml, który zawiera właściwy dokument. Plik ten w przypadku formatu Word 2007 (docx) jest zgodny z językiem WordProcessingML. Właśnie tam możemy znaleźć informacje na temat układu elementów XML, które w efekcie dają efekt, jaki widzimy podczas edycji tego dokumentu w edytorze Word.

Reasumując gdy nie wiemy jakich elementów użyć (i jak je poukładać) by uzyskać dokument Word o zadanej strukturze, można tą strukturę stworzyć przy pomocy edytora Word, a następnie "rozpakować" plik i podejrzeć jak to jest zrobione. Być może dla niektórych czytelników to banał, ale przyznam, że sam z tej metody korzystam. Jest ona prosta i skuteczna.

Mam nadzieję że w tym artykule przybliżyłem czytelnikom sposoby tworzenia dokumentów typu DOCX (zgodnych z formatem Open XML). Oczywiście zaprezentowane tutaj przykłady pokazują zaledwie skrawek wszystkich możliwości pakietu i formatu OpenXML.

Zachęcam do odwiedzenia mojego bloga:

<http://maciej-progtech.blogspot.com> - znajdują tam się inne interesujące artykuły i materiały.

- 2009.04.27 - 2009.08.11 - Pierwsza wersja artykułu opublikowana w częściach na moim [blogu](#).
- 2009.10.25 - Kolejna wersja artykułu przygotowana na portal [CodeGuru.pl](#)

#### **Załączniki:**

- [OpenXML\\_SDK>HelloWorld\\_src.zip](#)

[Idź na górę strony](#)