Unfolding a Word document with WordApplication Sample

Introduction

In the course of debugging, it can often be difficult to visualize the structure of documents as they are manipulated by WordApplication.

Our documentation has an overview of how WordApplication represents a Word document and how to insert elements using WordApplicationThis post provides a generalized function for displaying the structure of a document.
The code below provides a function "UnfoldDocument" which will produce a formatted text representation of the element hierarchy in a given Document. Output assumes a monospaced font and is suitable for writing to a plain-text file, printing to a console or inserting in a tag on an ASP.Net page.

To make use of this function, pass it a reference to a document object as shown:

```
WordApplication wa = new WordApplication();
Document doc = wa.Open(chosefile.FileContent);
Console.WriteLine(UnfoldDocument(doc));
```

Code

```
public string UnfoldDocument(Document d)
{
    StringBuilder tree = new StringBuilder();
    RUnfold(d, 0, new List(), tree);
    return tree.ToString();
}

// Helper method that recursively traverses the document tree
private void RUnfold(Element e, int tab, List line, StringBuilder ret)
{
    // Handle "leaf element" classes and output meaningful text representations
    if (e.ElementType == Element.Type.Hyperlink)
    {
        string link = ((Hyperlink)e).GetUrlString();
        ret.AppendFormat("Hyperlink: '{0}' -> ({1})", e.Text, link);
        return;
    }
    else if (e.ElementType == Element.Type.MergeField)
    {
        string fname = ((MergeField)e).GetFieldName();
        ret.AppendFormat("MergeField: '{0}' -> ({1})", e.Text, fname);
        return;
    }
    else if (e.ElementType == Element.Type.InlineImage)
    {
        InlineImage image = (InlineImage)e;
        ret.AppendFormat("InlineImage: {0} x {1}", image.Width, image.Height);
        return;
    }
    else if (e.ElementType == Element.Type.CharacterRun)
    {
        // Escape control sequences and wrap CharacterRun
```

```
        // output to 60 columns for readability purposes
        string text = e.Text;
        text = text.Replace("\r", "\\r");
        text = text.Replace("\n", "\\n");
        text = text.Replace("\t", "\\t");
        ret.Append("CharacterRun: '");
        for (int x = 0; x < text.Length; x++)
        {
            if ((x % 61) == 60)
            {
                ret.AppendFormat("\n{0}",tabs(tab,line));
                ret.Append(' ', 14);
            }
            ret.Append(text[x]);
        }
        ret.Append('\'');
        return;
    }
    // Recursively traverse the children of non-leaf elements
    ret.Append(e.ElementType.ToString());
    line.Add(true);
    for (int x = 0; x < e.Children.Length; x++)
    {
        Element child = e.Children[x];
        ret.AppendFormat("\n{0}|\n{0}|--", tabs(tab, line));
        if (x == e.Children.Length - 1) { line[tab] = false; }
        RUnfold(child, tab + 1, line, ret);
    }
    line[tab] = true;
    return;
}

// Helper method that produces appropriate tabbing for each line of output
public string tabs(int n, List line)
{
    char[] ret = new char[n*3];
    for (int x = 0; x < (n * 3); x++)
    {
        if ((line[x/3])&&(x%3 == 0)) { ret[x] = '|'; }
        else { ret[x] = ' '; }
    }
```

```
    return new String(ret);
}
```