Mastering Text in Open XML WordprocessingML Documents

# Introduction

Processing text in Open XML word-processing documents seems very simple at first—you have the body of the document, paragraphs and tables in the body, and rows and cells in tables, exactly like HTML, right? Then it seems very hard—you see the markup for revision tracking, numbered and bulleted lists, content controls, markup that does not affect text, such as bookmarks and comments. Styles seem like they do not affect text, but if there are numbered and bulleted lists, they do. Actually, the truth is, it is somewhere around the middle. There is a lot to track, but each one of these features, taken by itself, is not very difficult.

That said, there are some basic ideas and abstractions that can simplify how you think about word-processing markup. These abstractions are relevant regardless of whether you are working with word-processing markup by using the Open XML SDK 2.0 strongly-typed object model by using the [Open XML SDK 2.0 for Microsoft Office](http://msdn.microsoft.com/en-us/library/bb448854.aspx) with LINQ to XML, or using some other platform, such as Java or PHP. We can write code that addresses these abstractions. The code can expose exactly those elements that you are interested in, in an organized, predictable manner. In this article, I present Microsoft Visual C# code written with both LINQ to XML and with the Open XML SDK 2.0 strongly-typed object model. Because the semantics of some useful methods are defined carefully, they are easy to implement in whatever language and platform that you are using.

# Understanding Text Content in WordprocessingML

In the main body part of a document all text is contained in paragraphs. We find paragraphs in three locations: as a child of the body element (w:body), as a child of a cell in a table (w:tc), and as a child of text box content (w:txbxContent). A cell can itself contain a table. There are other instances of text in the main document part. Pictures can contain alternative text, and SmartArt graphic contains text. However, those pieces of text are more isolated. The issues around assembling the text of multiple strings into a single string do not apply to them.

# Best Practice: Accept Revisions before Processing

The first and most important point about simplifying how you process WordprocessingML content is that you should first accept all tracked revisions. For more information about the semantics of tracked revisions, see, [Accepting Revisions in Open XML Word-Processing Documents](http://msdn.microsoft.com/en-us/library/ee836138.aspx). Also see the Microsoft Visual C# 3.0 code sample for accepting tracked revisions at the [PowerTools for Open XML](http://www.codeplex.com/powertools) project on CodePlex. Click the Downloads tab, and then download RevisionAccepter.zip.

The thing about accepting tracked revisions first is that after you do this, you can safely ignore more than 40 elements that complicate how you process content. Many of those elements have complicated semantics. Therefore, it is much better to process them, and then process the contents of the document. Until I wrote that MSDN article, and wrote the code to accept revisions, I did not appreciate all of the cases in which more primitive approaches result in retrieving the wrong text for a paragraph.

In many circumstances, you want to query a document without modifying it. You can use a simple technique of reading the document into a byte array, creating a resizable memory stream from the byte array, and then opening the document from the memory stream. For more information about how to do this, see the blog post [Simplifying Open XML WordprocessingML Queries by First Accepting Revisions](http://blogs.msdn.com/ericwhite/archive/2010/01/11/simplifying-open-xml-wordprocessingml-queries-by-first-accepting-revisions.aspx). This example lets you accept revisions and query the document without touching the actual document on disk.

# Understanding WordprocessingML Abstractions

To help understand WordprocessingML markup, let's define some abstractions:

* Block-level content container
* Block-level content
* Run-level content container
* Run-level content
* Sub-run-level content

After you accept tracked revisions, and decide to ignore some elements that are only applicable in advanced scenarios, you are left with the following list of elements to process.

## Block-Level Content Containers

Block-level content containers are those WordprocessingML elements that contain block-level content such as paragraphs or tables. There are only three block-level content container elements that occur in the main document part:

|  |  |  |
| --- | --- | --- |
| Element | Element Name | Open XML SDK 2.0 Class Name Namespace: DocumentFormat.OpenXml.Wordprocessing |
| Body | w:body | Body |
| Table Cell | w:tc | TableCell |
| Text Box Content | w:txbxContent | TextBoxContent |

As I mentioned, there are other block-level content containers in WordprocessingML that contain paragraphs, such as the w:comment element in the comments part and the w:hdr element in the header part. However, they are not in the main document part. Therefore, they do not present the same processing challenges.

## Block-Level Content

Block-level content elements are the **WordprocessingML** elements that occupy all the width of the layout surface. They are bounded on the top and bottom, and occupy the available width from left to right of the available space. As an example, in the usual course of layout of a document, you do not see two paragraphs on the same physical line, or see a paragraph and a table side-by-side.

There appears to be exceptions to this rule, but in fact these apparent exceptions are not really exceptions. One example in which you see paragraphs side-by-side is using a multi-column page layout. In this case, the available width for layout of the paragraph or table is the column, not the complete page. Another example is when there is a text box on the page, but in this case, the available width for layout of the block-level content does not include the space reserved for the text box. Further, the text box itself has its own layout surface.

After accepting revisions, there are only two block-level content elements.