

Requirements to Run Tool

C# Compiler / Visual Studio .NET* Requirements

Microsoft .NET Framework and C# compiler: Version 4.0.3019.1 (or higher) --
<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=9cfb2d51-5ff4-4491-b0e5-b386f32c0992&displaylang=en>

Optional: Visual Studio 2010 (any edition) for IDE based development --
<http://www.microsoft.com/visualstudio/>

Note: The tool has been primarily compiled and tested using version 4.0.3019.1 of the .NET Framework and C# compiler. When building using this combination, you may experience compiler obsolete warnings. These can be ignored, although the tool may be updated to utilize more current methods.

Note: The tool may be compiled with Novell's Mono implementation of the C# language compiler. The resulting executable can then be run on the Microsoft implementation of the CLI. Full Mono CLI runtime support may be added in the future as this tool is updated and/or enhanced.
<http://www.mono-project.com/>

Microsoft Office* Requirements

Microsoft Office 2007/2010

A file named Word.dll needs to be created using the "tlbimp" tool that is provided with the Microsoft .NET Framework. For Microsoft Office 2007 or 2010, tlbimp should be run on its "mword.olb" file. The version of the created dll for Microsoft Office 2007 will be 8.4; Office 2010 will be 8.5. See the Instructions for Use section for more detail.

Note: The tool is mostly tested on the Office 2010 platform, so it is recommended that you run the tool with that version of Office. However, the tool should work with the build in the Office2007 directory as well.

XML

The tool uses the built in XML and XSL support in the .NET Framework that is provided with the .NET Framework redistributable or Visual Studio 2010.

Operating System Requirements

The operating systems that the tool runs on include Windows 7, Vista or XP. This tool has not been tested on any earlier operating system. The primary development platform for this tool was Windows 7.

Instructions for Use

The tool is currently implemented as a command line based tool.

Here are the steps to run the tool:

1. Extract the tool into any directory. Make sure all files are extracted into the same

directory

2. If not done already, create a COM interface to Word in order for the C# code to call into the Microsoft Word APIs via COM Interop that is provided with the .NET Framework. This is accomplished by using the 'tlbimp' command line tool that is part of the .NET Framework.
 - Open up a .NET aware Command Window
 - For Microsoft Office 2010
 1. Change directories to <Microsoft Office 2010 Path>\Office14
 2. Run tlbimp on msword.olb

```
C:\Program Files\Microsoft Office\Office>tlbimp msword.olb
```

- For Microsoft Office 2007
 1. Change directories to <Microsoft Office 2007 Path>\Office12
 2. Run tlbimp on msword.olb

```
C:\Program Files\Microsoft Office\Office>tlbimp msword.olb
```

- This will create a file called 'Word.dll'
 - Copy Word.dll into the directory where "docbuilder.exe" was extracted
3. Even though an executable version of the tool is provided, the tool may be compiled again using the .bat build files provided. Warnings may appear depending on the compiler/Word combination that is being utilized, but the tool should run as specified.
 4. When running the tool, run it from one of the two given directories associated with the proper Microsoft Office setup (for future versions of Office, the Word2010 should be sufficient).
 5. Run docbuilder.exe (from the appropriate directory associated with your Office build) from a .NET Framework aware command prompt.
 - The correct signature to run the tool is as follows:

```
docbuilder <Doc Type> <XML File> <XSL File> <Path to Save Docs>  
[-NSD] [-NSN]
```

Where

```
<Doc Type> is one of ".cs", ".doc", ".txt" (without the quotes)  
<XML File> is the name of the raw XML file to be transformed  
<XSL File> is the name of the XSL file that will do the  
transformation  
<Path to Save Docs> is the path to where the docs will be saved.  
[-NSD] - Optional - tells the tool to save the documents in a  
directory structure in accordance with the namespace of the class  
[-NSN] - Optional - tells the tool to prepend the namespace to  
the name of the document
```

Example (use this to run the tool out of the box)

```
docbuilder .cs Test.xml docs.xsl c:\temp\code\ -NSD
```

6. When the command in (3) is executed, the tool begins running. As it is running, Microsoft Word documents will be opening and closing.
7. The tool has completed when Microsoft Word has closed and the command prompt again has the focus. *Note: It could take a bit of time depending on the size of the XML and the makeup of the machine it is running on.*
8. The output files will be located in the directory specified at the command line above.

Advanced Feature

There is currently one advanced feature worth mentioning. Each class in the XML is part of a specific library. Currently the XML submission consists of six libraries. The default for the tool is to generate documentation or code for every class in every library. Within the two provided XSL files is a variable called "libraries-to-generate" which allows a user of the tool to specify which libraries are generated when it is run. Currently, the tool is capable of generating whole libraries and has no provision to selectively generate specific classes within a library.

Instructions for use of this feature are included in the XSL file, but here is the excerpt from the "sigs.xsl" file:

```
<!-- This is set by the user if he/she wants more fine grained control
of the libraries that are transformed. The ECMA submission is one big
XML file with more than one library (BCL, Implementation, etc.). By
default this stylesheet transforms all libraries. But if the user wants
on one or two, he/she:

- MUST comment out __ALL__
- UNCOMMENT others or add the libraries desired to the libraries-to-
generate variable by using the xsl:value-of for each library, MAKING
SURE that each library is spelled exactly as it is in the XML and a
COMMA follows
-->

<!-- What libraries in the current XML to generate code from -->
<xsl:variable name="libraries-to-generate">
  <!-- Default -->
  <xsl:value-of select="'__ALL__'" />
  <!-- Other Possible Options. Must comment out __ALL__ -->
  <!--<xsl:value-of select="'BCL,'" />-->
  <!--<xsl:value-of select="'RuntimeInfrastructure,'" />-->
  <!--<xsl:value-of select="'XML,'" />-->
  <!--<xsl:value-of select="'Reflection,'" />-->
  <!--<xsl:value-of select="'Networking,'" />-->
  <!--<xsl:value-of select="'ExtendedNumerics,'" />-->
</xsl:variable>
```

Tool Limitations

Performance

Performance of the tool is not necessarily optimal. Code generation usually takes less time than generating Word documentation, given the same set of data. For example, on a machine with an Intel Core i5 2.66 GHz CPU and 3 GB RAM, generation of the Word documentation will probably run anywhere between 5 and 7 minutes. Of course, different results will occur for different people.

Transforms

At this point, for all transforms, Word documents are created in the process. Also, unless you change the actual C# code, the #Page Break Here, #Document Break Here and #Type Break must remain in the XSL files as the tool only knows these keywords. Functionality to allow user defined keywords may be added to a future release.