Using HTMLToWord

HTMLToWord is an open source project from SoftArtisans. It can be downloaded from SourceForge.net.

Table of Contents

What is HTMLToWord?

HTMLToWord is an Open Source Microsoft Windows .NET C# project by SoftArtisans, Inc. that allows users to insert fragments of well-formed HTML (XHTML) into a Word document as formatted text. HTMLToWord uses SoftArtisans' OfficeWriter for Word (WordWriter) WordApplication object.

HTMLToWord parses XHTML fragments and inserts the contents into a Word document at the location you specify. In general, you can use HTMLToWord to insert headings, paragraphs, lists and tables into a new or existing Word document that you open or create with the SoftArtisans WordApplication object. The current version of HTMLToWord is not designed to convert entire HTML pages to Word documents.

HTMLToWord understands a subset of the tags defined in the HTML specification. See the documentation page on supported tags for a complete list of supported tags and their support level. For any HTML tag not listed in the table, or if your XHTML documents have custom tags, HTMLToWord will attempt to insert the text contents of the tag into the Word document. You can override this default behavior by setting the AcceptUnknownTags and IncludeContentsOfUnknownTags properties found in the HTMLInsertProperties property container and by implementing the InsertElementDelegate and FormatElementDelegate delegate methods.

Requirements

- OfficeWriter Enterprise Edition
- WordWriter dll (SAWW3NET.DLL or SoftArtisans.OfficeWriter.WordWriter.dll) version 3.8.1.665 or higher

  If you download the binary version of HTML-to-Word, your WordWriter dll must be the current version for which HTML-to-Word is built. Please see the HTML-to-Word open source project homepage for up-to-the-minute WordWriter dll version requirement info.
- OfficeWriter server requirements as described on the SoftArtisans System Requirements page.

Installation

Obtaining HTMLToWord

If you have not already done so, download either the binary version or the source version of the HTMLToWord project from SourceForge.net. Unzip the files into a directory of your choosing.

Building the DLL from the source

If you downloaded the binary version of the project, you can skip this step and move on to Setting up your project to use HTMLToWord.

Create a new Visual C# Class Library project in Visual Studio and include the 3 source files from the source distribution:

- HTMLToWord.cs - The main source file

- AttributeSetter.cs - Helper classes for setting attributes
- XHTMLCleanUp.cs - Helper class for modifying the user-supplied well-formed HTML string to try to make it XHTML-conforming.

In this new project, add a reference to System.XML and to the WordWriter DLL (SAWW3NET.DLL or SoftArtisans.OfficeWriter.WordWriter.dll). Make sure the DLL you have is version 3.8.1 build 665 or later. (If you have an older version of the WordWriter DLL, contact your SoftArtisans sales representative or send an e-mail to sales@softartisans.com.) Press "Ctrl-Shift-B" to build the project. If the build is successful, you will now have a file called HTMLToWord.DLL in your project's Bin\Release directory.

Setting up your project to use HTMLToWord

The HTMLToWord object is defined in the HTMLToWord.DLL file. HTMLToWord makes use of the WordWriter WordApplication object. WordWriter Enterprise Edition must be installed on your system. You must add a reference both to the WordWriter DLL (SAWW3NET.dll or SoftArtisans.OfficeWriter.WordWriter.dll) and to HTMLToWord.DLL in your project. You must have SAWW3NET.DLL or SoftArtisans.OfficeWriter.WordWriter.dll version 3.8.1 build 665 or higher on your system. If you have an older version of the WordWriter DLL, contact your SoftArtisans sales representative or send an e-mail to sales@softartisans.com.

To use the HTMLToWord DLL:

- Make sure you have a reference in your project to the WordWriter DLL (SAWW3NET.DLL or SoftArtisans.OfficeWriter.WordWriter.dll).
- Create a reference in your project to the HTMLToWord DLL (HTMLToWord.DLL).

Inserting HTML into a Word document

Before you begin, you should make sure to import the HTMLToWord namespace in your project.

C#

```
using SoftArtisans.HTMLToWord
```

VB.NET

```
Imports SoftArtisans.HTMLToWord
```

To insert HTML into a WordWriter Document as formatted text, you must first open or create a Document using the SoftArtisans WordApplication object. You will then need to create an instance of the HTMLToWord object. Lastly, you will need some well-formed HTML (XHTML) text to insert into your document. The XHTML can be stored in a file or in a database or can be generated programatically.

In the examples below, we will simply create an XHTML string directly in the code. This text will be inserted at the end of our document.

You must make sure that your HTML is well-formed. That is, every open tag must have a corresponding close tag or be self-closing (e.g.
). If your HTML is older and does not conform to XHTML standards, you can use any number of utilities, such as the Open Source utility HTMLTidy, to clean-up your HTML before passing it to HTMLToWord's InsertHTML method.

C#

```
//--- Create a new WordWriter document.
WordApplication wApp = new WordApplication();
Document doc = wApp.Create();

//--- Create the HTMLToWord object.
HTMLToWord h2w = new HTMLToWord();

//--- The XHTML to insert into the WordWriter document.
string xhtml = "<h1>Greetings</h1><p><b>Hello</b>, <i>world</i>!</p>";

//--- Insert the XHTML at the end of the WordWriter document.
Element lastInsertedElement = h2w.InsertHTML(xhtml, doc, null, doc);
```

VB.NET

```
'--- Create a new WordWriter document.
Dim wApp As New WordApplication()
Dim doc As Document = wApp.Create()

'--- Create the HTMLToWord object.
Dim h2w As HTMLToWord = New HTMLToWord()

'--- The XHTML to insert into the WordWriter document.
Dim xhtml As String = "<h1>Greetings</h1><p><b>Hello</b>, <i>world</i>!</p>"

'--- Insert the XHTML at the end of the WordWriter document.
Dim lastInsertedElement As Element = h2w.InsertHTML(xhtml, doc, null, doc)
```

Using Delegate Methods

Delegate methods enable you to write custom code that is called by HTMLToWord while processing an HTML string. If you specify them, delegate methods can be called:

1. When a tag in the XHTML string is encountered. This allows you to optionally process the tag yourself before handing control back to HTMLToWord. You can also tell HTMLToWord how to handle each tag: process the tag and all of its contents and sub-tags; or skip over this tag and all of the sub-tags.
   See documentation for InsertDelegate delegate.
2. After an Element has been inserted into the document, HTMLToWord gives you the opportunity to do additional processing on the new element, such as adding formatting.
   See documentation for FormatDelegate.
3. Whenever an HTML img tag is encountered. HTMLToWord itself does not know how to find image file that is specified with an HTML img tag. Using a delegate method, HTMLToWord looks to you to find the image file's contents and pass it back as a System.IO.Stream.
   See documentation for ImageDelegate delegate.

HTMLToWord includes three properties that allow you to specify your own methods for handling HTML tags.

- InsertDelegate - method called by HTMLToWord each time it encounters a tag in the HTML string.
- FormatDelegate - method called by HTMLToWord after it has inserted an element into the Word document.
- ImageDelegate - method called by HTMLToWord each time it encounters an img tag.

Using delegate methods, you can add handling for any tag in the XHTML string, even tags that are not part of HTML.

If you implement a InsertElementDelegate or FormatElementDelegate delegate method, you must import the System.Xml namespace into your project. HTMLToWord turns your XHTML string into an XML DOM tree, and hands you an XmlNode object representing the HTML tag being processed.

The example below shows how to create a delegate method that will be called each time an HTML is encountered by HTMLToWord and assign it to the InsertDelegate property. The delegate method does 2 things. First, it handles a custom para tag that is in our HTML by inserting a paragraph into the Word document. Second, it tells HTMLToWord to ignore anything inside a custom private tag so that sensitive information is not written into the document.

C#

```
//--- This method will be used as a delegate by HTMLToWord and
//--- is an example of type HTMLToWord.InsertElementDelegate.

//--- We have defined 2 custom XHTML tags for our documents: para and private.
//--- If we encounter a <para> tag, we insert a paragraph into the Word document and
tell HTMLToWord
//--- to continue processing.  If we encounter a <private> tag, we write a short
message into
//--- the Word document and tell HTMLToWord to skip this tag altogether.
HTMLToWord.HTMLTagAction MyInserter(Document doc, Element insertAfterElement, XmlNode
node)
{
     //--- Get the name of the HTML tag and convert it to lower case.
     string nodeName = node.Name.ToLower();

     if (nodeName.equals("para"))
     {
          //--- There is a special <para> tag in our markup, but we should just treat
it
          //--- like a standard HTML <p> tag.  Insert a paragraph into the document
and
          //--- let HTML continue processing.
          insertAfterElement.InsertParagraphAfter(null);
     }
     else if (nodeName.equals("private"))
     {
          //--- The "private" tag is used to indicate information that should not be
sent
          //--- to outside clients; therefore, we tell HTMLToWord to skip this tag and
its
          //--- contents.
          return HTMLToWord.HTMLTagAction.Skip;
     }

     //--- We have handled all of the special cases.  Let HTMLToWord know that
     //--- it should process this tag.
     return HTMLToWord.HTMLTagAction.Process;
}
```

VB.NET

```
'--- This method will be used as a delegate by HTMLToWord and
'--- is an example of type HTMLToWord.InsertElementDelegate.

'--- We have defined 2 custom XHTML tags for our documents: para and private.
'--- If we encounter a <para> tag, we insert a paragraph into the Word document and
tell HTMLToWord
'--- to continue processing.  If we encounter a <private> tag, we write a short
message into
'--- the Word document and tell HTMLToWord to skip this tag altogether.
Function MyInserter(ByVal doc As Document, ByVal insertAfterElement As Element, ByVal
xmlNode As XmlNode)
     As HTMLToWord.HTMLTagAction

     '--- Get the name of the HTML tag and convert it to lower case.
     Dim nodeName As String = node.Name.ToLower()

     If (nodeName.Equals("para")) Then
          '--- There is a special <para> tag in our markup, but we should just treat
it
          '--- like a standard HTML <p> tag.  Insert a paragraph into the document and
          '--- let HTML continue processing.
          insertAfterElement.InsertParagraphAfter(Nothing)
     Else If (nodeName.Equals("private"))
          '--- The "private" tag is used to indicate information that should not be
sent
          '--- to outside clients; therefore, we tell HTMLToWord to skip this tag and
its
          '--- contents.
          Return HTMLToWord.HTMLTagAction.Skip
     End If

     '--- We have handled all of the special cases.  Let HTMLToWord know that
     '--- it should process this tag.
     Return HTMLToWord.HTMLTagAction.Process
End Function
```

You must tell HTMLToWord about your delegate method so that HTMLToWord can call your custom code. To do that, assign your delegate method to the appropriate HTMLToWord property.

C#

```
//--- Create an HTMLToWord instance.
    HTMLToWord h2w = new HTMLToWord();

    //--- Using the code from the example above, tell HTMLToWord that it should
    //--- call our "MyInserter" delegate method whenever it encounters a tag in
    //--- the XHTML string.
    h2w.InsertDelegate = new HTMLToWord.InsertElementDelegate(MyInserter);
```

VB.NET

```
'--- Create an HTMLToWord instance.
    Dim h2w As New HTMLToWord()

    '--- Using the code from the example above, tell HTMLToWord that it should
    '--- call our "MyInserter" delegate method whenever it encounters a tag in
    '--- the XHTML string.
    h2w.InsertDelegate = New HTMLToWord.InsertElementDelegate(AddressOf MyInserter)
```

Note that HTMLToWord does not allow you to chain delegate methods.

Troubleshooting

This section contains information to help you if you are having trouble using the HTMLToWord.DLL in your project.

Error: The located assembly's manifest definition with name 'SAWW3NET' does not match the assembly reference.

Exact error message: java.lang.ClassNotFoundException: The located assembly's manifest definition with name 'SAWW3NET' does not match the assembly reference.

This error usually indicates that the version of the WordWriter DLL that you have on your system does not match the version that was used to create the HTMLToWord.DLL file.

If you downloaded the binary version of the project from the SourceForge web site, check the ReadMe.txt file in the directory where you unzipped the distribution. It contains the version number of the WordWriter DLL (SAWW3NET.DLL) that was used to create the HTMLToWord.DLL file. If the version of the WordWriter DLL on your system is older than the version documented in the ReadMe.txt file, you should contact your SoftArtisans sales representative or send an e-mail to sales@softartisans.com to get an newer version of the WordWriter DLL.

If you downloaded the source version of the project from the SourceForge web site, make sure that the version of the WordWriter DLL you used to build the HTMLToWord.DLL file is the same version and build number as the version of the WordWriter DLL file (SAWW3NET.DLL) you are referencing in your project.

Participating in the Open Source Project

SoftArtisans has released the source code for HTMLToWord as an Open Source project. If you would like to download the lastest version of HTMLToWord in either binary format (the DLL plus user documentation for the API) or source code (the Visual C# source files plus user documentation for the API), feel free to visit our project page on SourceForge.net at http://sourceforge.net/projects/htmltoword. Note that the binary version will work on 32-bit platforms with .NET 1.1 or later. If you would like a build for 64-bit platforms, please download the source distribution and rebuild the DLL on your system.