



Essential Studio 2013 Volume 4 - v.11.4.0.26

Essential DocIO



Contents

1	Overview	6
1.1	Introduction to Essential DocIO	6
1.2	Prerequisites and Compatibility	8
1.3	Documentation	9
2	Installation and Deployment	11
2.1	Installation.....	11
2.2	Samples and Location	11
2.3	Deployment Requirements.....	21
2.3.1	DLLs	21
3	Getting Started	23
3.1	Class Diagram	23
3.2	Creating Platform Application.....	25
3.3	Deploying Essential DocIO.....	30
3.3.1	Windows	30
3.3.2	ASP.NET	35
3.3.3	WPF	38
3.3.4	Silverlight	41
3.3.5	ASP.NET MVC	45
3.4	Saving the Word Document.....	51
3.5	Feature Summary.....	53
3.6	API Changes	55
4	Concepts and Features	56
4.1	Basic Concepts.....	56
4.2	Word Document.....	59
4.2.1	Document Properties.....	66
4.2.2	Document Background	72
4.2.2.1	Document Color.....	73
4.2.2.2	Watermark	77
4.2.3	Docx Support.....	83
4.2.3.1	Word 2010 Support	86

4.2.4	Stream Support	87
4.2.5	Macro-enabled Document Support	88
4.2.6	Track Changes	90
4.2.7	Font Substitution for Word Documents	92
4.3	Section.....	94
4.3.1	Cloning and Merging	100
4.3.2	Headers and Footers.....	103
4.3.3	Table.....	111
4.3.3.1	Table Row	115
4.3.3.2	Table Cell	117
4.3.3.3	Row Format.....	122
4.3.3.4	Table Styles.....	124
4.3.4	Content Control	126
4.4	Paragraph.....	126
4.4.1	Paragraph Item.....	130
4.4.1.1	Text Range	132
4.4.1.2	Fields 135	
4.4.1.2.1	Merge Field.....	139
4.4.1.2.2	Embed Field	141
4.4.1.2.3	Seq Field	142
4.4.1.2.4	Form Field	143
4.4.1.2.5	Hyperlink.....	155
4.4.1.2.6	Document Variable	158
4.4.1.2.7	Fields Updating Engine	161
4.4.1.3	Bookmark	163
4.4.1.3.1	Bookmark Navigator.....	168
4.4.1.4	Shapes 173	
4.4.1.4.1	Picture	174
4.4.1.4.2	TextBox	178
4.4.1.4.3	Comment.....	182
4.4.1.5	Footnote and Endnote	187
4.4.1.5.1	Footnote and Endnote Separators	190
4.4.1.6	Symbol 191	
4.4.1.7	Break 193	
4.4.1.8	Table Of Contents	196

4.4.1.9 OLE Object	199
4.4.2 Styles and Formatting.....	204
4.4.2.1 Accessing Styles	205
4.4.2.2 Character Styles and Formats.....	206
4.4.2.3 Paragraph Styles	212
4.4.2.4 Paragraph Formats.....	216
4.4.2.4.1 Tabs.....	220
4.4.2.5 Lists 222	
4.4.2.5.1 List Format.....	230
4.4.2.6 Text Box Format.....	235
4.4.2.7 Importing XHTML	241
4.5 Find and Replace	248
4.5.1 TextSelection.....	249
4.5.2 TextBodyPart.....	250
4.5.3 TextBodySelection.....	252
4.5.4 Find.....	254
4.5.5 Replace	257
4.6 Mail Merge	261
4.6.1 Nested Mail Merge	266
4.6.2 Mail Merge Events.....	272
4.6.3 Additional Mail Merge Features.....	278
4.7 Security.....	281
4.7.1 Encryption and Decryption	282
4.8 Conversion	283
4.8.1 Doc to RTF	285
4.8.2 Doc to HTML	286
4.8.3 Doc to PDF	290
4.8.4 RTF to Doc	296
4.8.5 Doc to EPub	299
4.9 Docx Support.....	306
4.10 Supported Elements	309
5 Frequently Asked Questions	317
5.1 Does DocIO require MS Word to be installed on the report generation machine / server?.....	317
5.2 Would it be possible to view the generated report [DOC] using Essential DocIO?.....	317

5.3	Can Essential DocIO be used to read MS Word files? What are the MS Word versions that Essential DocIO can read?	317
5.4	How to format a Hyperlink by using DocIO?	318
5.5	How to format a Table?	319
5.6	How to modify the Built-in styles?	320
5.7	How to open a Document from Stream by using DocIO?	321
5.8	How to set the Page Size of the Document Section?	324
5.9	How to set OpenType features?	324
5.10	Does Essential DocIO provide support for Client profile?	328
5.11	How to insert section breaks?	328
5.12	Migration from Microsoft Office Automation to Essential DocIO	329
5.12.1	Mail Merge	329
5.12.2	Find and Replace	333
5.12.3	Bookmarks.....	338
5.12.4	Page numbers	342
5.12.5	Document Watermark.....	346
5.12.6	Header/Footer	352
5.12.7	Character Formatting.....	359
5.12.8	Tables 363	
5.12.9	Comments	367
5.12.10	Document Protection.....	376
5.12.11	Table of Contents	380
5.13	How to Attach a Template to a Word Document?	384

1 Overview

This section covers information on Essential DocIO, its key features, prerequisites to use the control, its compatibility with various OS and browsers, and finally the documentation details complimentary with the product. It comprises the following subsections:

1.1 Introduction to Essential DocIO

Essential DocIO is a 100% native .NET library that generates fully functional Microsoft Word documents in native Word format. Essential DocIO is used to read and write Microsoft Word files. It features a full-fledged object model similar to the Microsoft Office COM libraries. It does not use COM interop and is built from scratch in C#. The DocIO library can be used in any .NET environment including C#, VB.NET and managed C++. It is a non-UI component that is used in Windows Forms, WPF, Silverlight, ASP.NET, ASP.NET MVC, and Windows Store applications.

Use Case Scenario

Essential DocIO is used by companies to generate Newsletters to provide information of interest to their members. It is also used to generate Envelopes, Resumes, Newsletters, Invoice and Letter Creations.

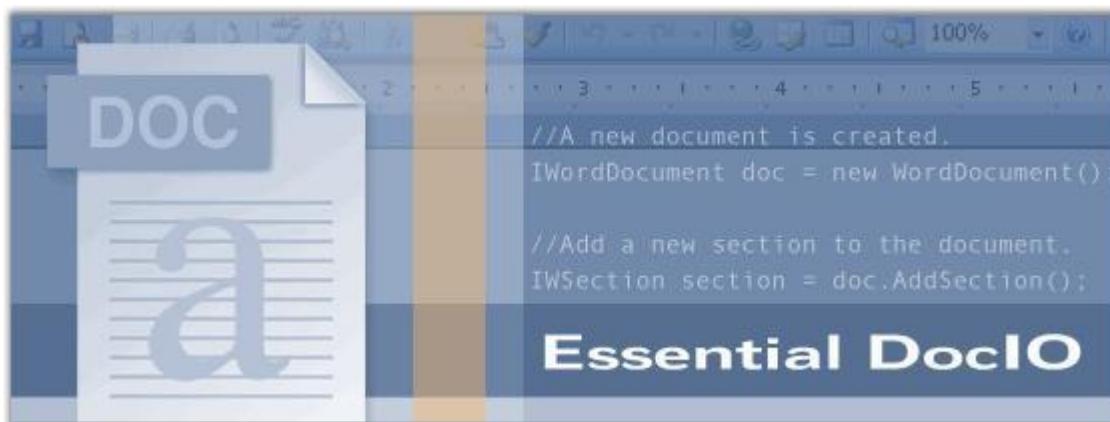


Figure 1: Essential DocIO

Key Features

Important features of Essential DocIO are listed below.

- Support to create and edit a new document.

- Support to modify existing MS Word documents.
- Support to read and write Built-In and Custom Document Properties.
- Advanced support to Find and Replace text with its original formatting.
- Support to insert Bookmarks.
- Advanced Mail Merge support with different data sources.
- DocIO lets you to encrypt and decrypt Word documents.
- Support to deploy applications with DocIO in medium trust.
- Support to insert and extract Ole objects.
- Support to import HTML contents.
- Support to export as PDF, RTF or HTML documents.

User Guide Organization

The product comes with numerous samples as well as an extensive documentation to guide you. This User Guide provides detailed information on the features and functionalities of Essential DocIO. It is organized into the following sections:

- **Overview**-This section gives a brief introduction to our product and its key features.
- **Installation and Deployment**-This section elaborates on the install location of the samples, license, and so on.
- **Getting Started**-This section guides you on getting started with Essential DocIO.
- **Concepts and Features**-The features of Essential DocIO are illustrated with use case scenarios, code examples and screen shots under this section.
- **Frequently Asked Questions**-This section covers the list of questions with expert solutions.

Document Conventions

The following conventions will help you to quickly identify the important sections of information while using the content.

Table 1: Document Conventions

Convention	Icon	Description
Note	 Note:	Represents important information
Example	 Example	Represents an example
Tip		Represents useful hints that will help you in using the controls/features
Additional Information		Represents additional information on the topic

1.2 Prerequisites and Compatibility

This section covers the requirements mandatory for using Essential DocIO. It also lists operating systems and browsers compatible with the product.

Prerequisites

The prerequisites details are listed below:

Table 2: Prerequisites

Development Environments	<ul style="list-style-type: none"> • Visual Studio 2010 (Ultimate, Premium, Professional and Express) • Visual Studio 2008 (Team System, Professional, Standard & Express) • Visual Studio 2005 (Professional, Standard & Express) • Silverlight 4.0
.NET Framework versions	<ul style="list-style-type: none"> • .NET 4.0 • .NET 4.0 Client Profile • .NET 3.5 SP1 Client Profile • .NET 3.5 SP1 • .NET 2.0

Compatibility

The compatibility details are listed below:

Table 3: Prerequisites

Operating Systems	<ul style="list-style-type: none"> • Windows 8 (32 bit and 64 bit) • Windows Server 2008 (32 bit and 64 bit) • Windows 7 (32 bit and 64 bit) • Windows Vista (32 bit and 64 bit) • Windows XP • Windows 2003
-------------------	--

MS Word Compatibility Support	<ul style="list-style-type: none"> • MS Word 97 • MS Word 2000 • MS Word 2002 (XP) • MS Word 2003 • MS Word 2007 • MS Word 2010 • MS Word 2013
-------------------------------	---

1.3 Documentation

Syncfusion provides the following documentation segments to provide all the necessary information pertaining to Essential DocIO.

Table 4: Documentation

Type of Documentation	Location
Readme	<p>Windows Forms-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\Release Notes\\readme.htm</p> <p>ASP.NET-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\asp release notes\\readme.htm</p> <p>WPF-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\WPF release notes\\readme.htm</p> <p>Silverlight-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\Silverlight Release Notes\\readme.htm</p> <p>ASP.NET MVC-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\MVC Release Notes\\readme.htm</p>
Release Notes	<p>Windows Forms-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\Release Notes\\Release Notes.htm</p> <p>ASP.NET-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\asp release notes\\Release Notes.htm</p>

	<p>WPF-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\WPF release notes\\Release Notes.htm</p> <p>Silverlight-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\Silverlight Release Notes\\Release Notes.htm</p> <p>ASP.NET MVC-[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x.x\\Infrastructure\\Data\\MVC Release Notes\\Release Notes.htm</p>
User Guide (this document)	<p>Online</p> <p>http://help.syncfusion.com/resources (Navigate to the DocIO User Guide.)</p> <p> Note: Click Download as PDF to access a PDF version.</p> <p>Installed Documentation</p> <p>Dashboard -> Documentation -> Installed Documentation.</p>
Class Reference	<p>Online</p> <p>http://help.syncfusion.com/resources (Navigate to the Reporting User Guide. Select Doc/O, and then click the Class Reference link found in the upper right section of the page.)</p> <p>Installed Documentation</p> <p>Dashboard -> Documentation -> Installed Documentation.</p>

2 Installation and Deployment

This section covers information on the install location and samples of Essential DocIO. It comprises the following sub sections:

2.1 Installation

For step-by-step installation procedure for the installation of Essential Studio, refer to the **Installation** topic under **Installation and Deployment** in the **Common UG**.

For More Information Refer:

For licensing, patches and information on adding or removing selective components refer the following topics in **Common UG** under **Installation and Deployment**.

- Licensing
- Patches
- Add/Remove Components

2.2 Samples and Location

This section covers the location of the installed samples and describes the procedure to run the samples through the sample browser and online. It also lists the location of source code.

Sample Installation Locations

Sample install locations for different platforms are listed below:

- **Windows Forms**-The Windows Forms samples are installed in the following location:

*...My Documents\Syncfusion\EssentialStudio\Version
Number\Windows\DocIO.Windows\Samples\2.0*

- **ASP.NET**-The ASP.NET samples are installed in the following location:

*...My Documents\Syncfusion\EssentialStudio\Version
Number\Web\DocIO.Web\Samples\2.0*

- **WPF**-The WPF samples are installed in the following location:

...\\My Documents\\Syncfusion\\Essential Studio\\Version Number\\WPF\\DocIO.WPF\\Samples\\3.5

- **Silverlight**-The Silverlight samples are installed in the following location:

...\\My Documents\\Syncfusion\\Essential Studio\\Version Number\\Silverlight\\DocIO.Silverlight\\Samples\\3.5

- **ASP.NET MVC**-The ASP.NET MVC samples are installed in the following location:

...\\My Documents\\Syncfusion\\Essential Studio\\<Version Number>\\MVC\\dociomvc\\samples\\3.5

- **Windows Store** -The Windows Store samples are installed in the following location:

...\\<UserName>\\AppData\\Local\\Syncfusion\\Essential Studio\\<Version Number>\\WinRT\\SampleBrowser\\SampleBrowser

Viewing Samples

To view the samples, follow the steps below:

1. Click **Start** -> **All Programs**-> **Syncfusion** -> **Essential Studio <x.x.x.x>** -> **Dashboard**. The UI samples are displayed by default.
2. Select **Reporting**.

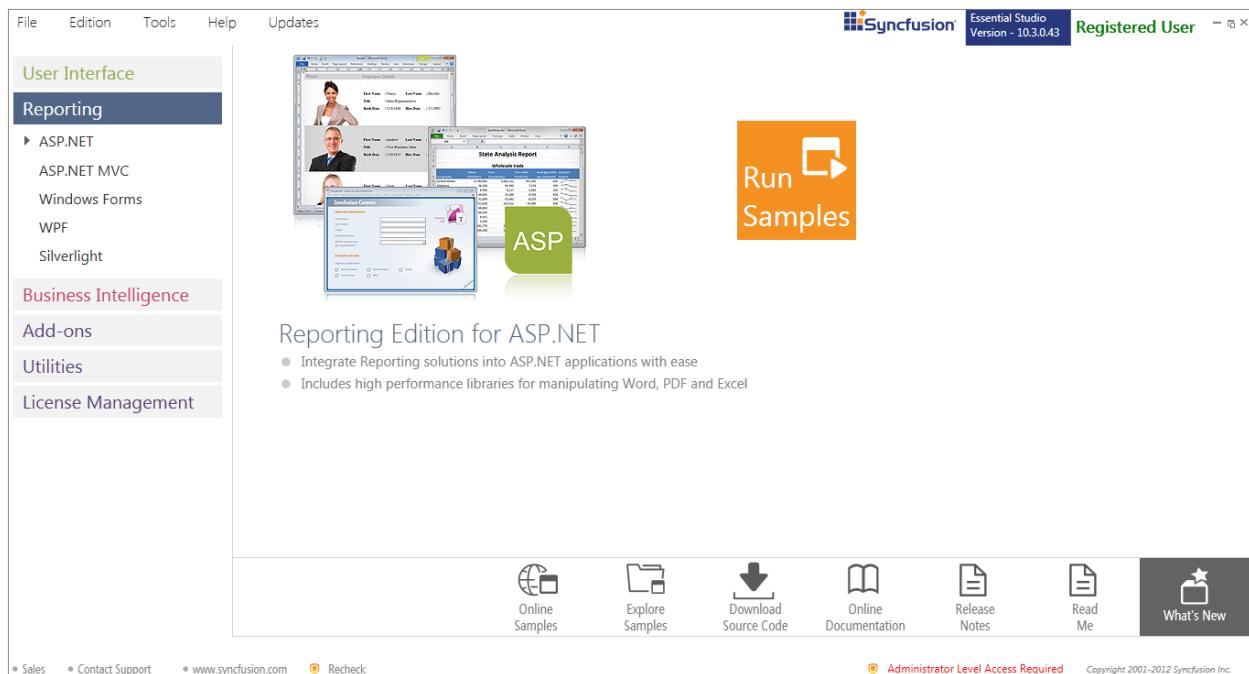


Figure 2: Syncfusion Essential Studio Reporting Dashboard

The steps to view the DocIO samples in various platforms are discussed below:

Windows

1. In the Dashboard window, click **Run Samples** for **Windows Forms** under **Reporting** Edition panel. The **Windows Forms** Sample Browser window is displayed.

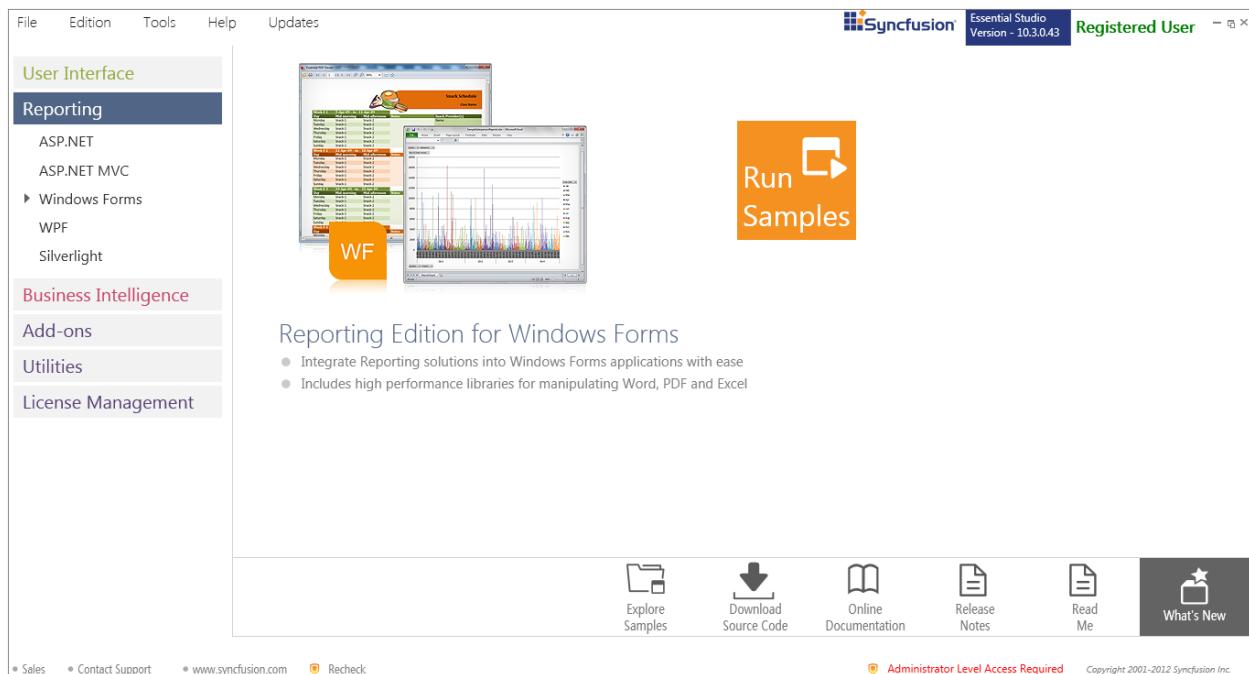


Figure 3: Reporting Edition Windows Forms Sample Browser



Note: You can view the samples in any of the following three ways:

- **Run Samples**-Click to view the locally installed samples.
- **Online Samples**-Click to view online samples.
- **Explore Samples**-Explore Windows Forms samples on disk.

2. Click **DocIO** from bottom-left pane. The DocIO samples are displayed.

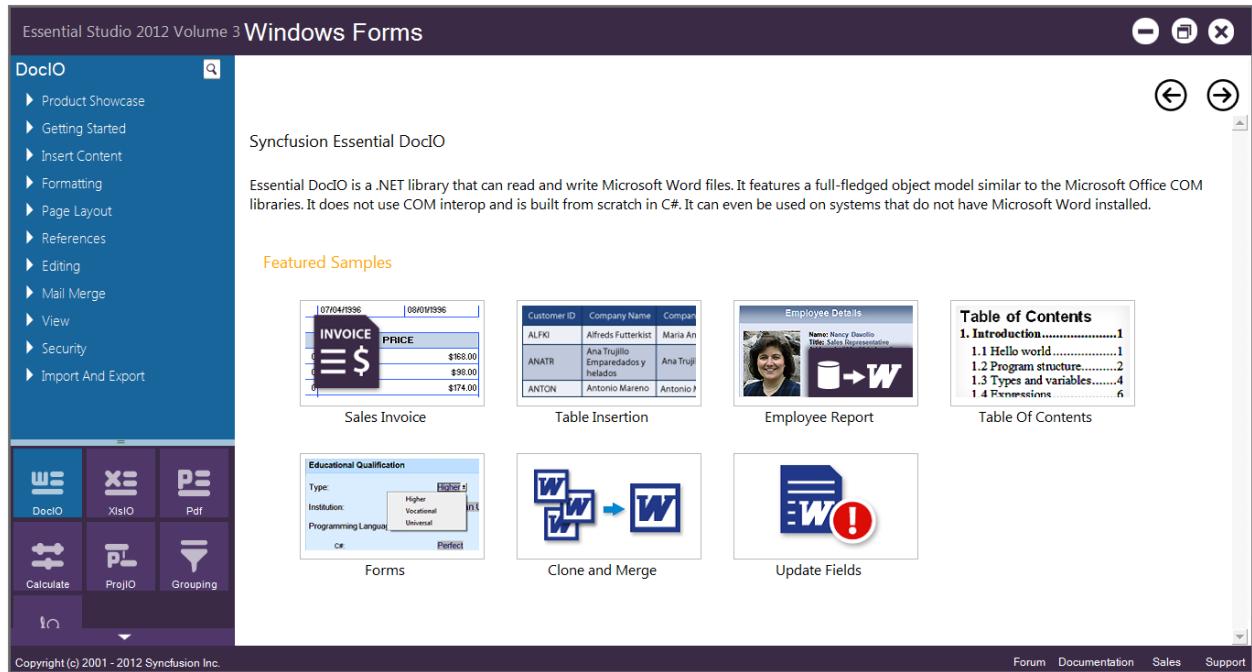


Figure 4: DocIO Samples Displayed in the Windows Forms Sample Browser

3. Select any sample and browse through the features.

ASP.NET

1. In the Dashboard window, click **Run Samples** for **ASP.NET** under **Reporting Edition** panel. The **ASP.NET** Sample Browser window is displayed.



Note: You can view the samples in any one of the three options displayed:

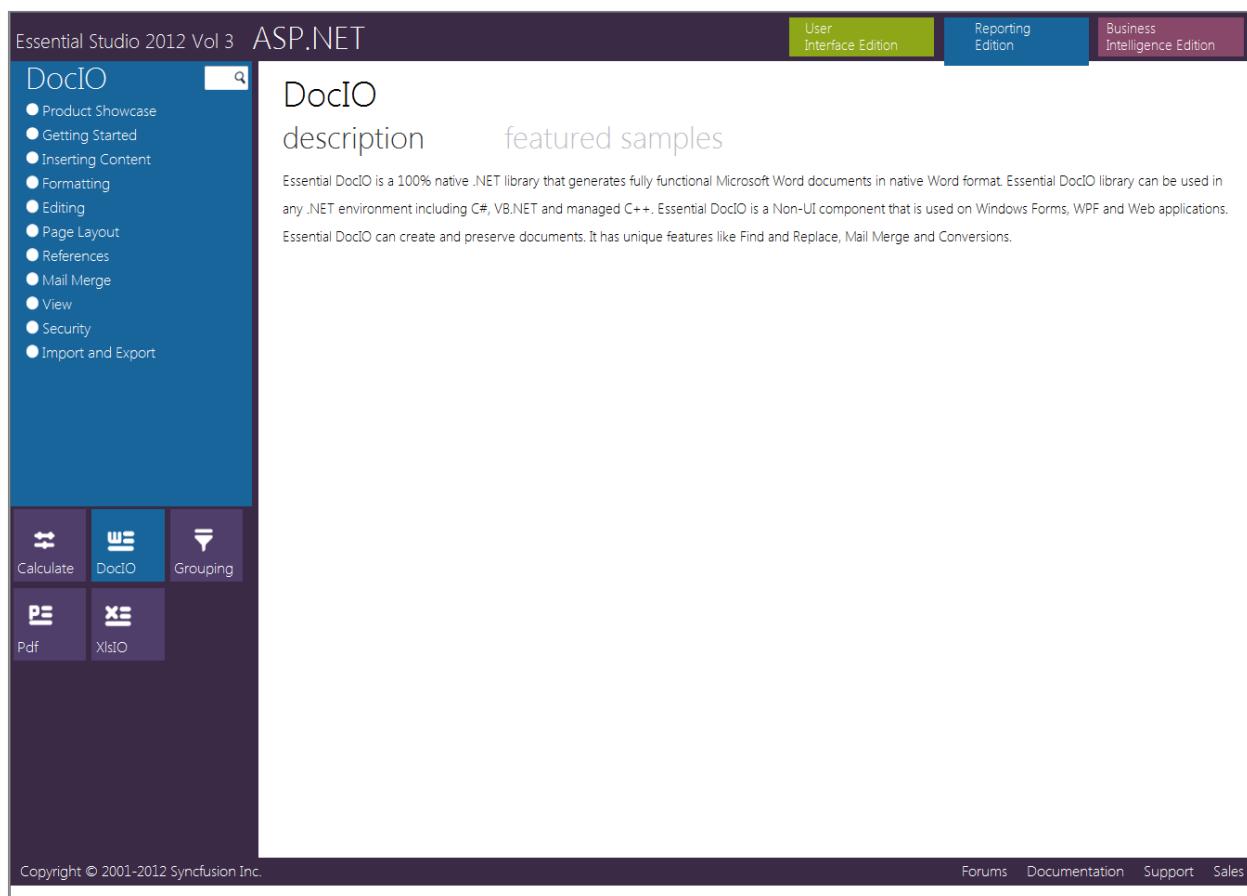


Figure 5: ASP.NET Sample Browser

2. Select any **DocIO** sample and browse through the features.

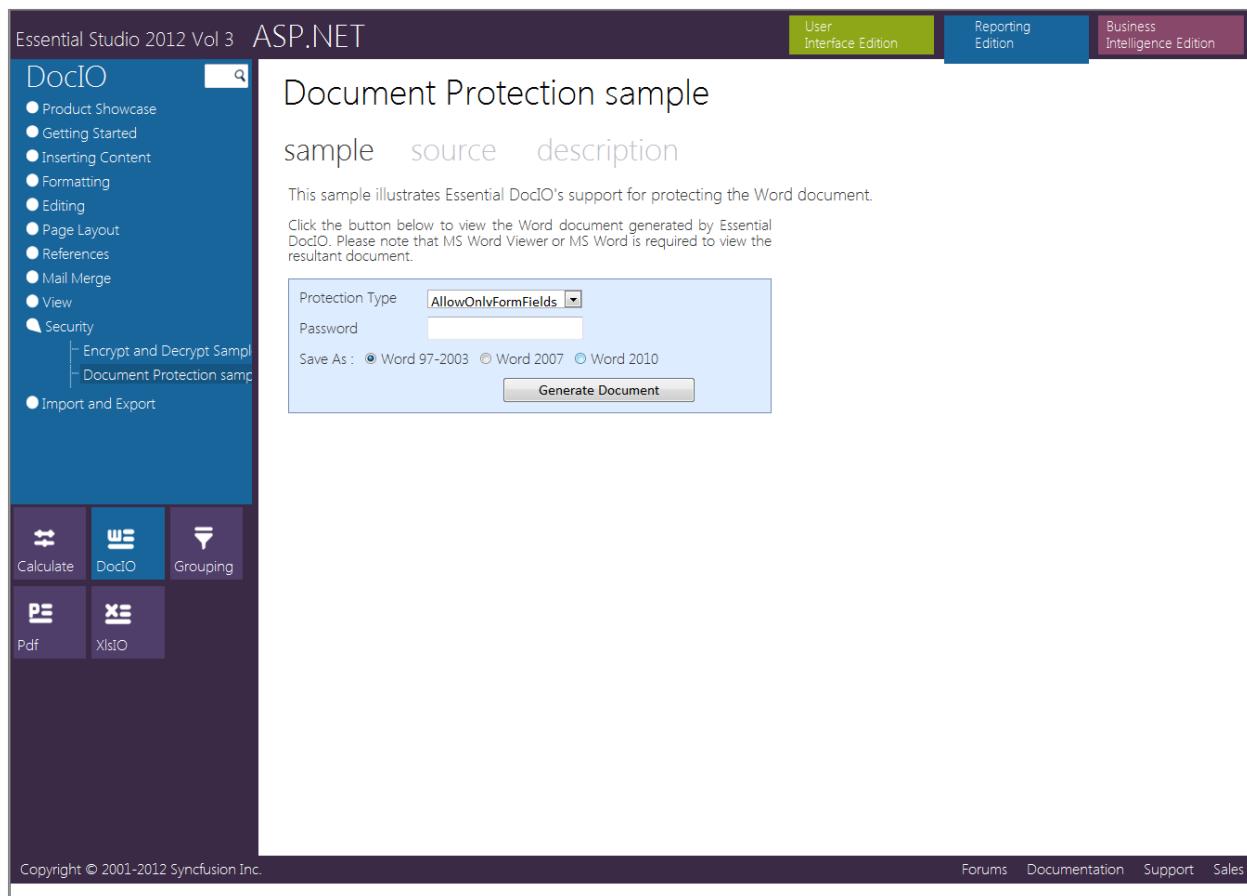


Figure 6: Documentation protection sample

WPF

1. In the Dashboard window, click **Run Samples** for **WPF** under **Reporting Edition** panel. The **WPF Sample Browser** window is displayed.



Note: You can view the samples in any one of the three options displayed:

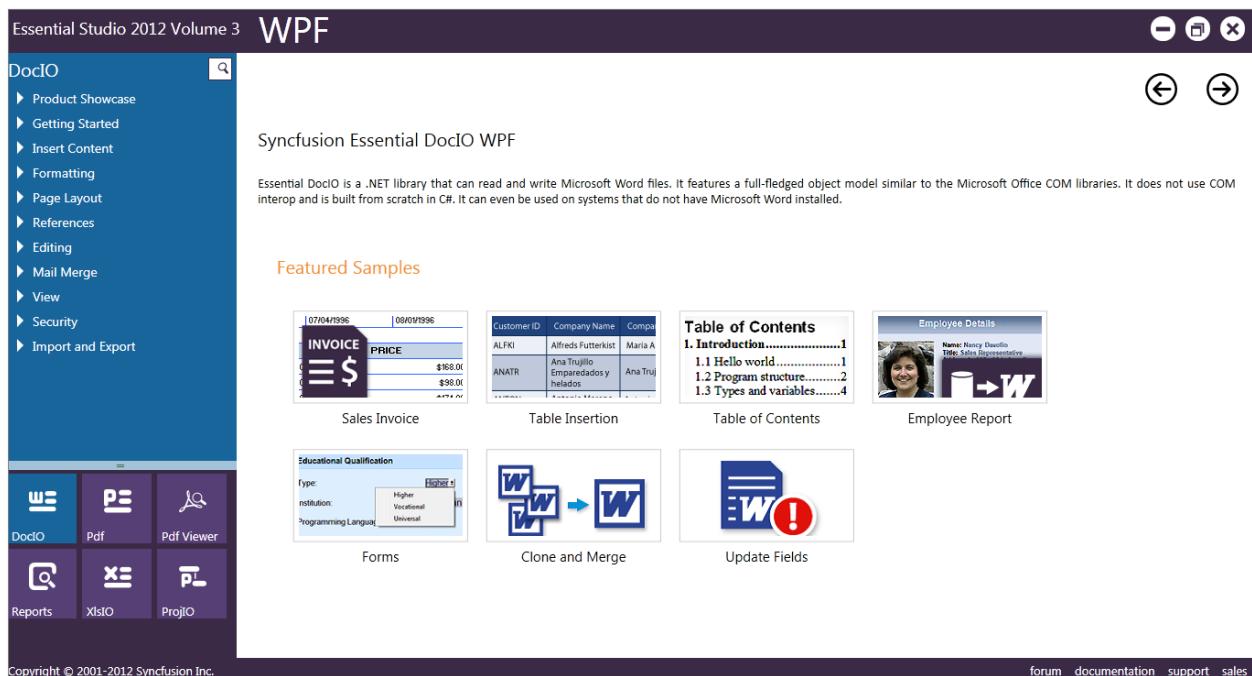


Figure 7: DocIO Samples Displayed in the WPF Sample Browser

2. Select any DocIO sample that is displayed on the left pane and browse through the features.

Silverlight

1. In the **Dashboard** window, click **Run Samples for Silverlight** under **Reporting** edition panel. The **Silverlight** Sample Browser window is displayed.



Note: You can view the samples in any one of the three options displayed:

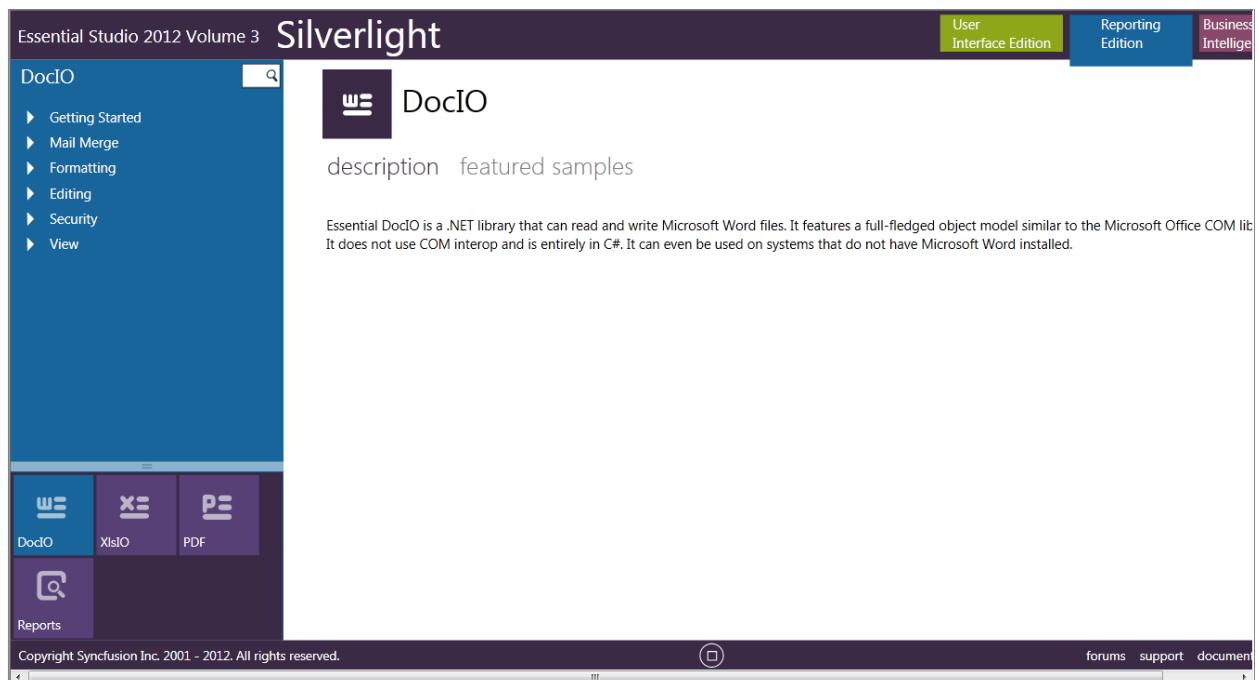


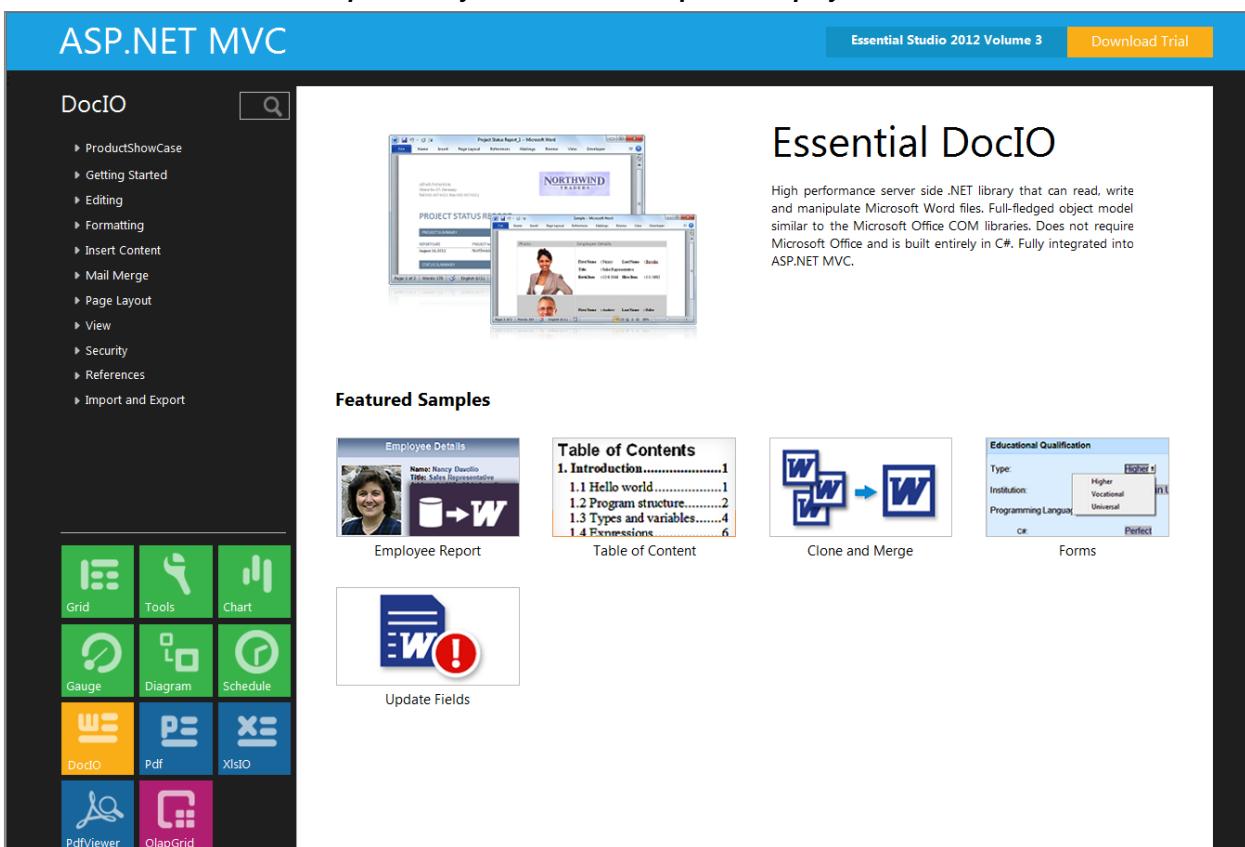
Figure 8: Silverlight Sample Browser

2. Select any DocIO sample that is displayed on the right pane and browse through the features.

ASP.NET MVC

1. In the Dashboard window, click **Run Samples** for ASP.NET MVC under **Reporting** Edition panel. The **ASP.NET MVC** Sample Browser window is displayed.

 Note: You can view the samples in any one of the three options displayed:



The screenshot shows the ASP.NET MVC Sample Browser interface. At the top, there's a navigation bar with links for 'Essential Studio 2012 Volume 3' and 'Download Trial'. Below the navigation bar, the main content area has a title 'ASP.NET MVC' and a sub-section 'DocIO'. On the left, there's a sidebar with a search bar and a list of 'Featured Samples' including 'ProductShowCase', 'Getting Started', 'Editing', 'Formatting', 'Insert Content', 'Mail Merge', 'Page Layout', 'View', 'Security', 'References', and 'Import and Export'. Below this is a grid of icons representing various features: Grid, Tools, Chart, Gauge, Diagram, Schedule, DocIO (highlighted in yellow), Pdf, XlsIO, PdfViewer, and OlapGrid. To the right of the sidebar, there are three sample preview windows: 'Employee Report' showing a photo of a woman and some text, 'Table of Contents' showing a table of contents for 'Employee Details' with sections like 'Introduction', 'Hello world', 'Program structure', 'Types and variables', and 'Expressions', and 'Clone and Merge' showing two overlapping document icons. Further down, there's a section titled 'Educational Qualification' with fields for Type (Higher), Institution (Vocational), Programming Language (Universal), and a dropdown menu for 'Higher'. At the bottom, there's a large 'Update Fields' button with a red exclamation mark icon.

Figure 9: DocIO Samples Displayed in the ASP.NET MVC Sample Browser

2. Select any DocIO sample and browse through the features.

Source Code Location

The source code for DocIO under different platforms is available at the following default locations:

- **Windows-[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\Windows\DocIO.Windows\Src**
- **ASP.NET-[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\Web\DocIO.Web\Src**
- **WPF-[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\WPF\DocIO.WPF\Src**
- **ASP.NET MVC-[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\MVC\DocIO.MVC\Src**

2.3 Deployment Requirements

The various deployment requirements are illustrated under the following sections:

2.3.1 DLLs

The following assemblies need to be referenced in your application for the usage of Essential DocIO.

Full Trust Mode

In full trust mode, add references to the following assemblies corresponding to the platform:

DocIO (Full-Trust) – Windows Forms, ASP.NET, WPF, ASP.NET MVC

- Syncfusion.Core.dll
- Syncfusion.Compression.dll
- Syncfusion.DocIO.Base.dll

DocIO – Silverlight

- Syncfusion.Compression.Silverlight.dll
- Syncfusion.DocIO.Silverlight.dll

Medium/Partial Trust Mode

In medium/partial trust mode, add references to the following assemblies corresponding to the platform:

DocIO (Partial-Trust) - ASP.NET

- Syncfusion.Core.dll
- Syncfusion.Compression.dll
- Syncfusion.DocIO.Web.dll

DocIO (Partial-Trust) - ASP.NET MVC

- Syncfusion.Core.dll
- Syncfusion.Compression.dll
- Syncfusion.DocIO.MVC.dll

DocIO (Client Profile) – Windows Forms and WPF

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.DocIO.ClientProfile.dll

3 Getting Started

This section covers information on the following topics:

3.1 Class Diagram

The following illustration shows the Class Diagram for Essential DocIO.

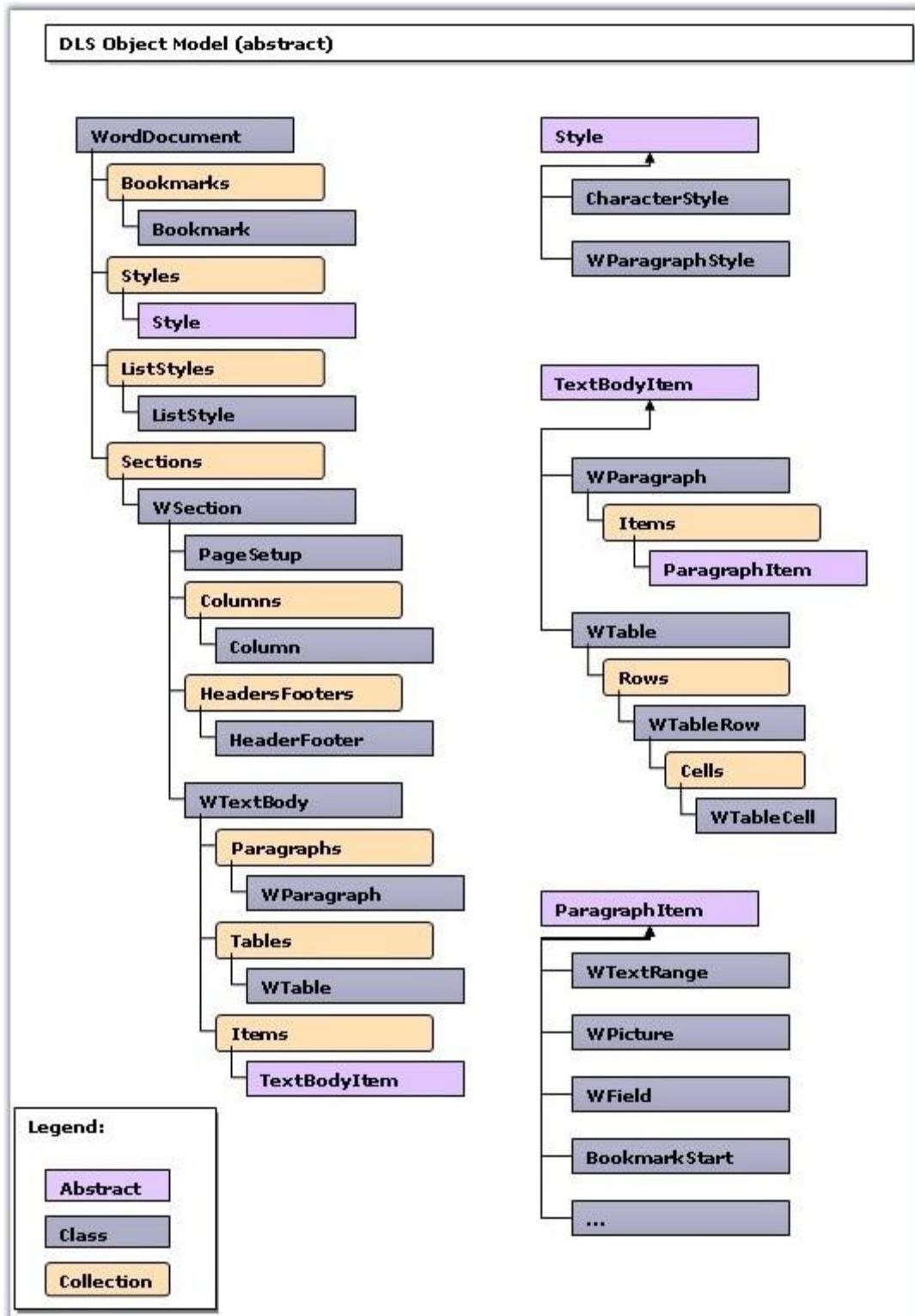


Figure 10: Class Diagram for Essential DocIO



Note:

- **DOM Basics (DLS)-Document Logical Structure [DLS]** is an object model of DocIO that enables you to interact with the Word document.
- **Bookmarks**-the list of bookmarks (MS Word: Insert->Bookmarks...)
- **Styles**-the list of document styles (MS Word: Format->Styles and Formatting)
- **Sections**-sections collection (Every section is a region with its own PageSetup options and HeaderFooters)
- **TextBodyItem**-base class for container elements of a document such as paragraphs and tables
- **ParagraphItem**-elements which contain paragraphs: formatted text, pictures, special symbols, bookmark markers, fields and so on

3.2 Creating Platform Application

This section illustrates the step-by-step procedure to create the following platform applications.

- Windows
- ASP.NET
- WPF
- Silverlight
- ASP.NET MVC

Windows Application

1. Open Microsoft Visual Studio. Go to **File** menu and click **New Project**. In the New Project dialog box, select **Windows Forms Application** template, name the project and click **OK**.

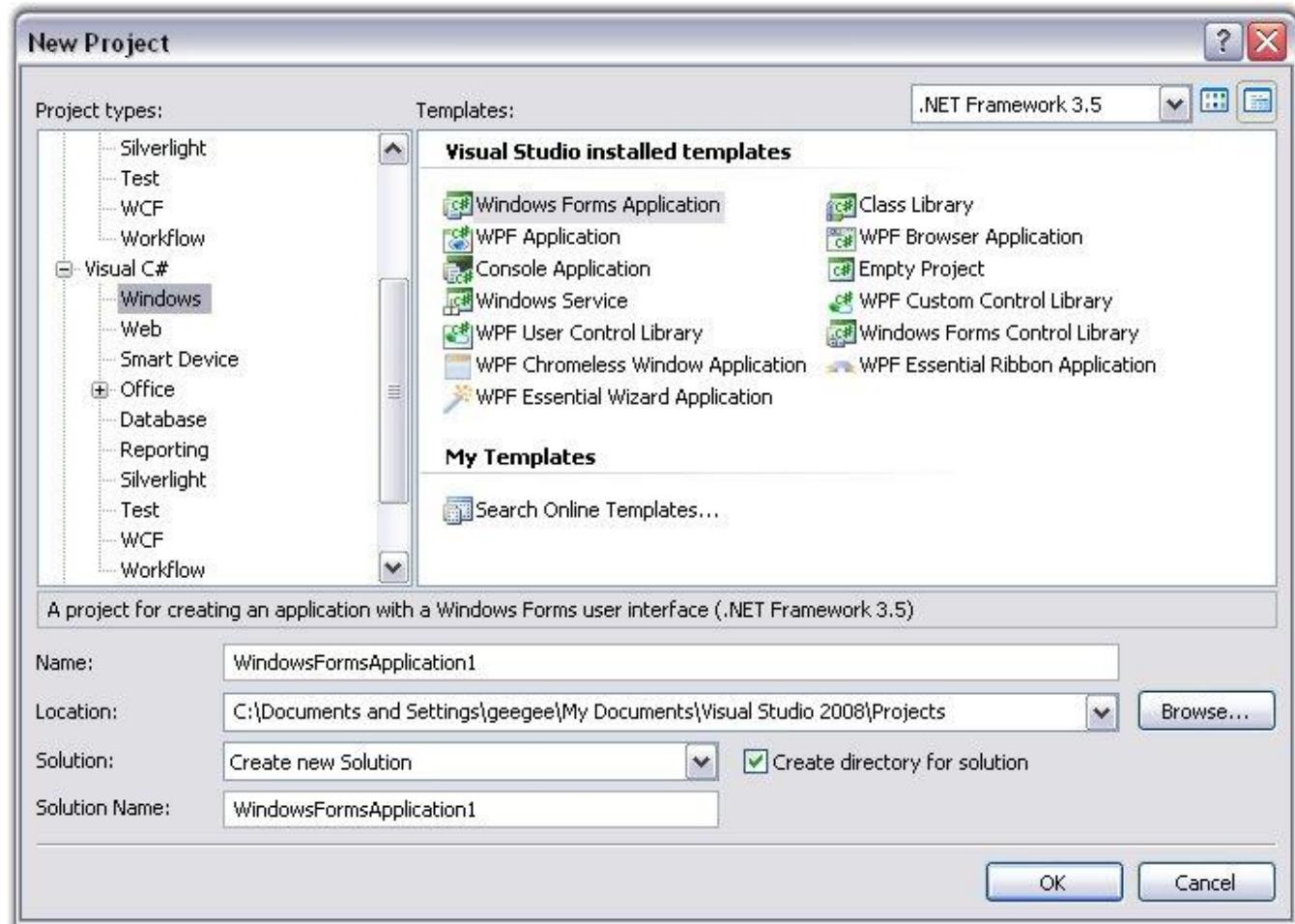


Figure 11: Windows Forms Application template selected in the New Project Dialog Box

A Windows application is created.

2. Now you need to deploy Essential DocIO into this Windows application. Refer [Windows](#) topic for detailed info.

ASP.NET Application

1. Open Microsoft Visual Studio. Go to **File** menu and click **New Web Site**. In the New Website dialog box, select **ASP.NET Web Site** template, name the website and click **OK**.

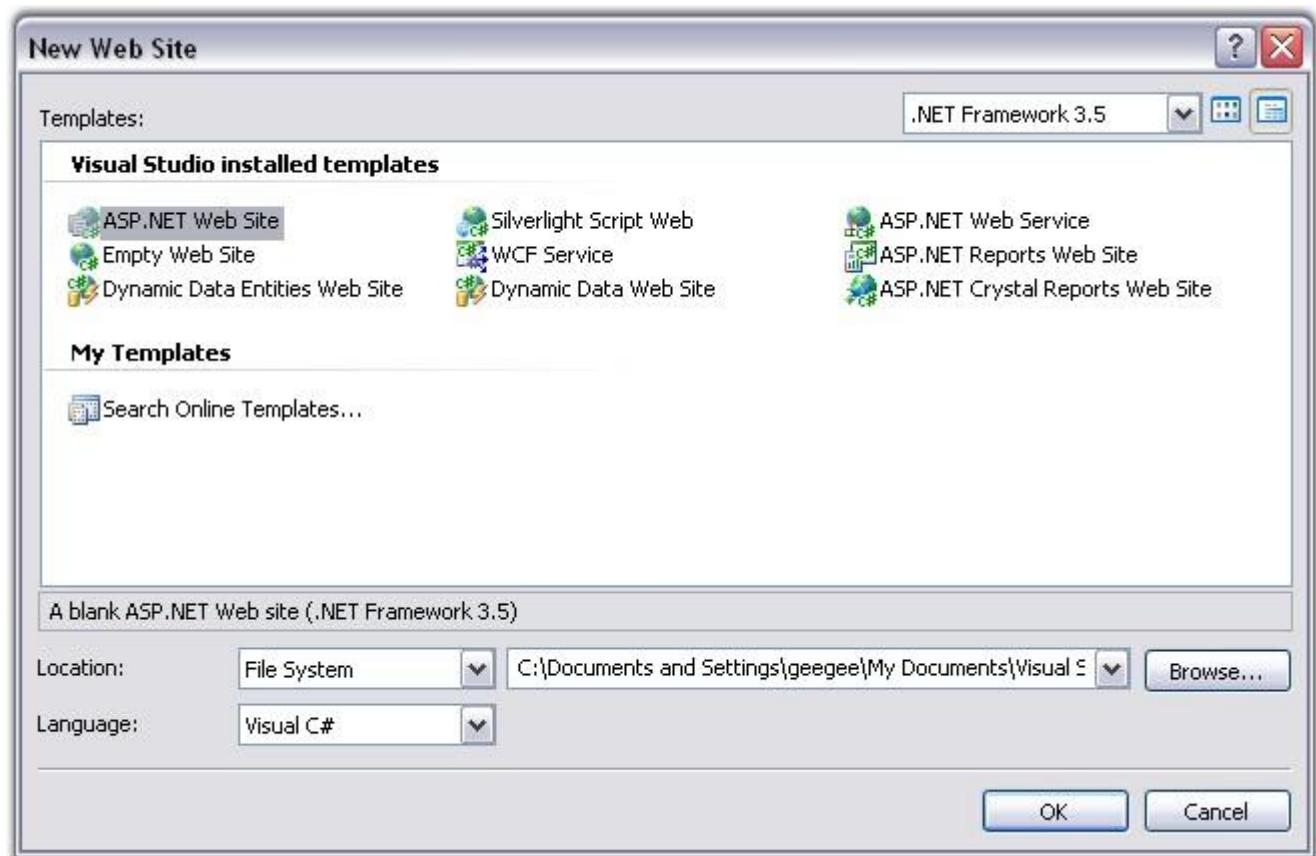


Figure 12: ASP.NET Web Site template selected in the New Web Site Dialog Box

An ASP.NET application is created.

2. Now you need to deploy Essential DocIO into this ASP.NET application. Refer [ASP.NET](#) topic for detailed info.

WPF Application

1. Open Microsoft Visual Studio. Go to **File** menu and click **New Project**. In the New Project dialog box, select **WPF Application** template, name the project and click **OK**.

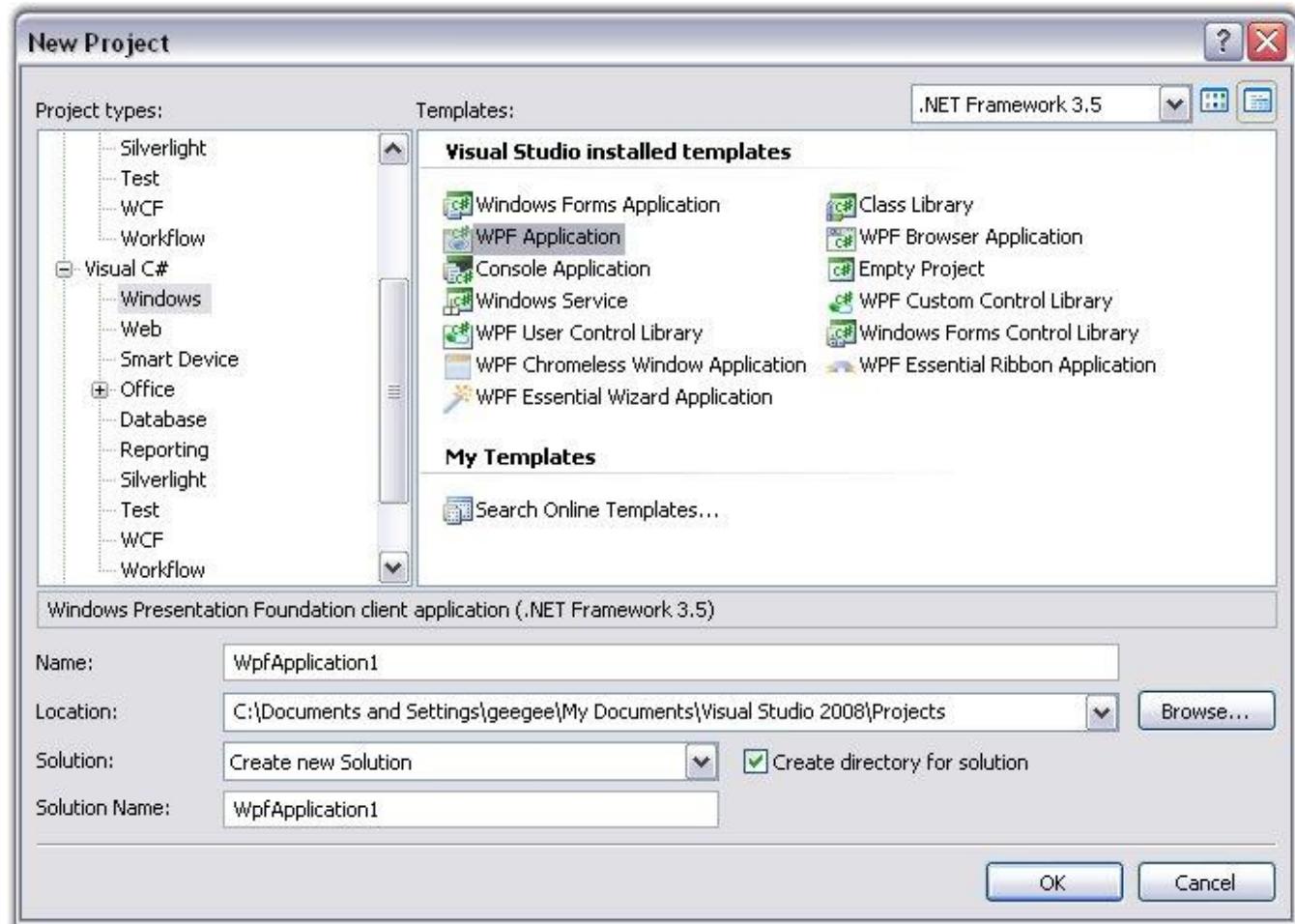


Figure 13: WPF Application template selected in the New Project Dialog Box

A new WPF application is created.

2. Open the main form of the application in the designer.
3. Now you need to deploy Essential DocIO into this WPF application. Refer [WPF](#) topic for detailed info.

Silverlight Application

1. Open Microsoft Visual Studio. Go to **File** menu and click **New Project**. In the New Project dialog box, select **Silverlight Application** template, name the project and click **OK**.

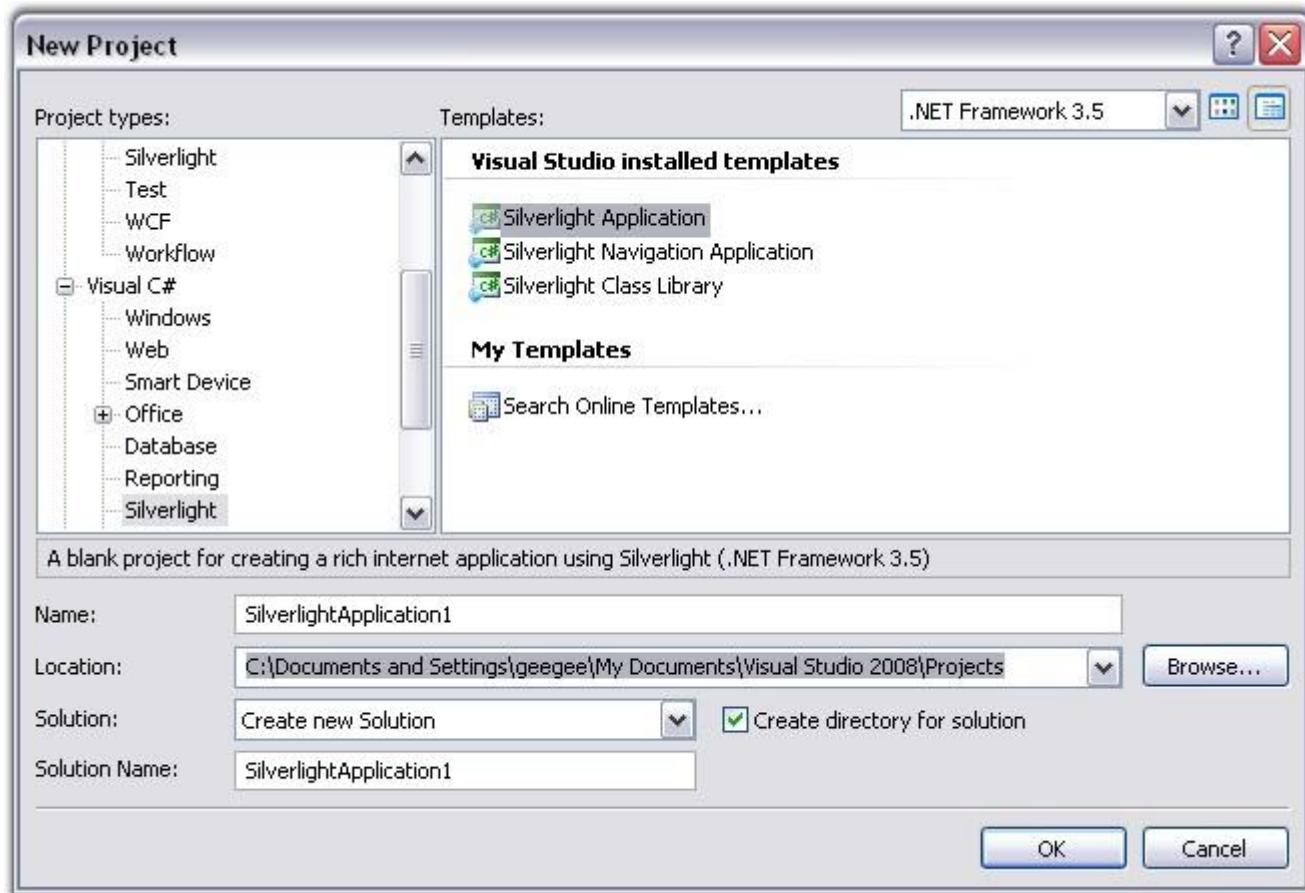


Figure 14: Silverlight Application template selected in the New Project Dialog Box

A **New Silverlight Application** dialog box appears as follows.

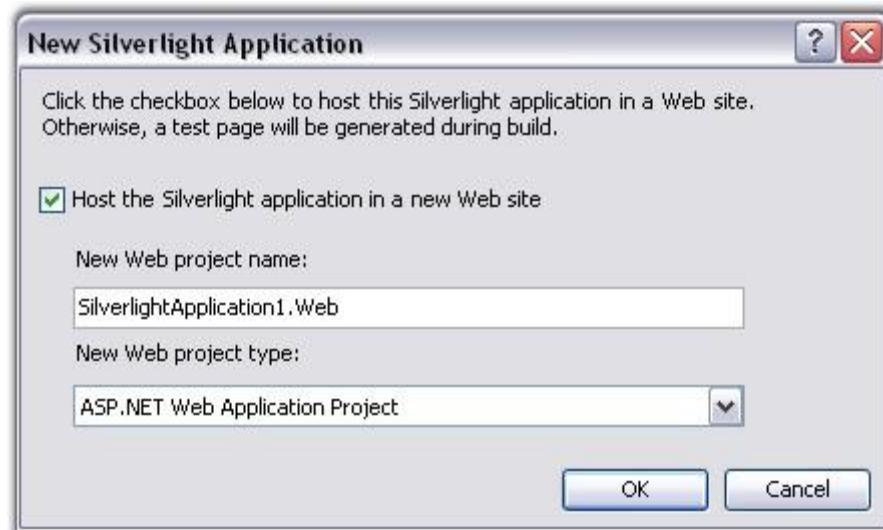


Figure 15: New Silverlight Dialog Box

2. Click **OK** to host the Silverlight application in a new website.

A new Silverlight application is created.

3. Open the main form of the application in the designer.

4. Refer [Silverlight](#) topic to know how to deploy Essential DocIO to the application.

ASP.NET MVC Application

Refer **ASP.NET MVC -> Grid -> Getting Started -> Creating an MVC** application topic to know how to create an MVC application.

To know how to deploy Essential DocIO to this application, refer [ASP.NET MVC](#) topic.

See Also

[Deploying Essential DocIO](#)

3.3 Deploying Essential DocIO

We have now created a platform application in the previous topic ([Creating Platform Application](#)). This section will guide you to deploy Essential DocIO in those applications under the following topics:

- Windows: This topic illustrates how to deploy DocIO in Windows applications.
- ASP.NET: This topic illustrates how to deploy DocIO in ASP.NET application.
- WPF: This topic illustrates how to deploy DocIO in WPF application.
- Silverlight: This topic illustrates how to deploy DocIO in Silverlight application.
- ASP.NET MVC: This topic illustrates how to deploy DocIO in ASP.NET MVC application.

3.3.1 Windows

Now, you have created a Windows application (refer [Creating Platform Application](#)). This section covers information on the following:

- Deploying Essential DocIO in a Windows Application
- Creating a Word Document

Deploying Essential DocIO in a Windows Application

In order to deploy an application that uses the Syncfusion assemblies, the referenced Syncfusion assemblies should reside in the application folder where the exe file exists, in the target machine.

In order to do that, go to the Solution Explorer; Under References, select all the Syncfusion assemblies and then change the Copy Local property of the Syncfusion assemblies to true and compile the project.

Now, the Syncfusion assemblies referenced in the project will be copied to the output directory along with the application executable (bin/debug/).

Deploy the exe file along with the Syncfusion assemblies found, to the target machine. Be sure that these Syncfusion assemblies reside in the same location as the application exe file, in the target machine.

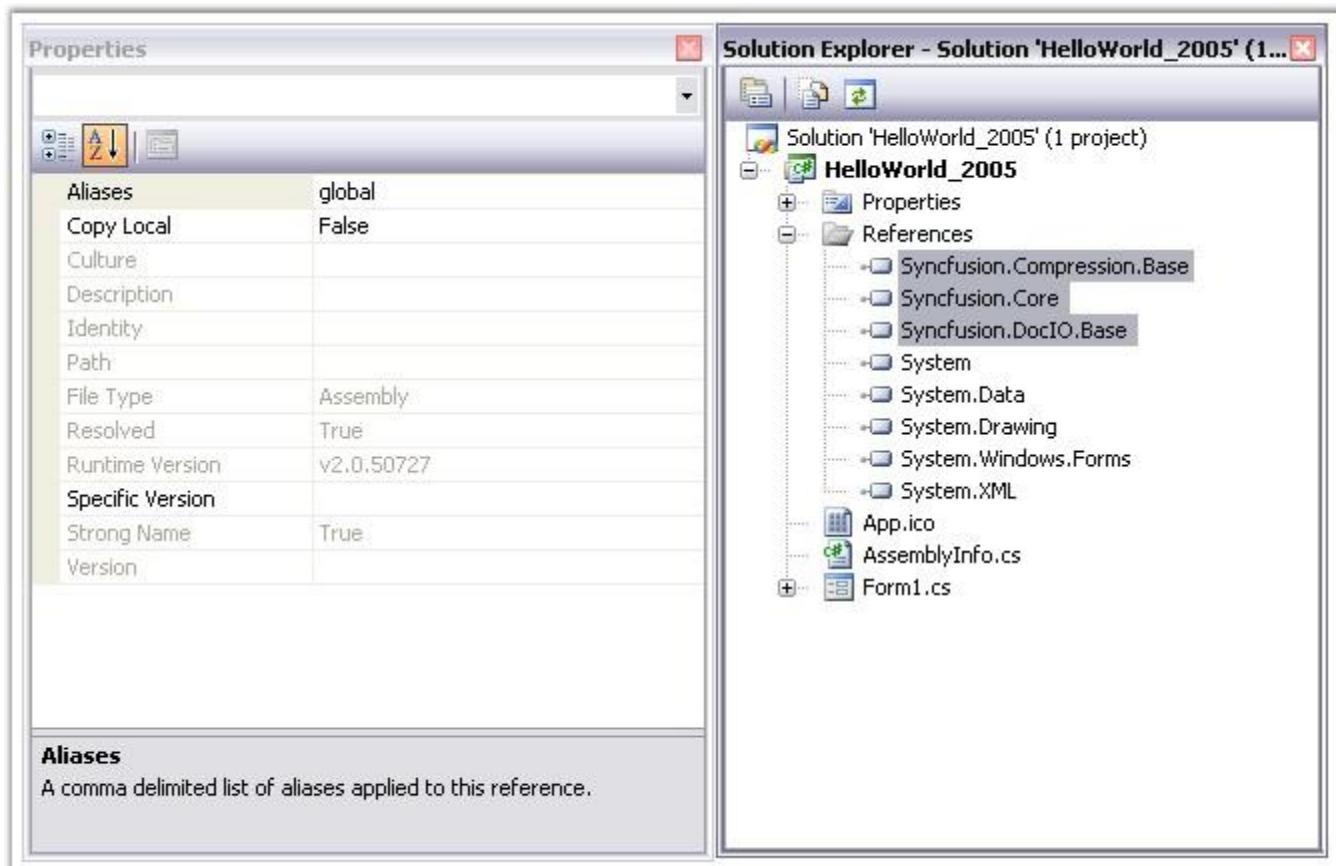


Figure 16: DLLs to be referenced in the Application



Note: Application with Essential DocIO needs the following dependent assemblies for deployment.

- Syncfusion.Core.dll
- Syncfusion.Compression.dll
- Syncfusion.DocIO.Base.dll



Note: For detailed documentation on Windows Application deployment, see http://www.syncfusion.com/support/user/uploads/DeployingWindowsApplication_bdaf76f7.pdf.

Essential DocIO is deployed in the Windows application.

Creating a Word Document

In this section, you will learn how to create a simple Word document with "Hello World" written on the first paragraph of the first section.

1. Include the following namespaces in your application.

- **Syncfusion.DocIO.DLS** (using Syncfusion.DocIO.DLS)
- **Syncfusion.DocIO** (using Syncfusion.DocIO)

2. Instantiate the **WordDocument** class.

[C#]

```
// Create a new Word document.  
// This document has no section and no paragraph by default.  
WordDocument document = new WordDocument();
```

[VB .NET]

```
' Create a new Word document.  
' This document has no section and no paragraph by default.  
Dim document As WordDocument = New WordDocument()
```

 **Note:** The `WordDocument` class represents the Word documents created in memory. This is the memory representation of the Word document written to disk.

3. Add a section to the newly created Word document by using the following code.

[C#]

```
// Add a new section to the document.  
IWSection section = document.AddSection();
```

[VB .NET]

```
' Add a new section to the document.  
Dim section As IWSection = document.AddSection()
```

4. Add a paragraph to the newly created section in the Word document by using the following code.

[C#]

```
// Add a new paragraph to the section.  
IWParagraph paragraph = section.AddParagraph();
```

[VB .NET]

```
' Add a new paragraph to the section.  
Dim paragraph As IWParagraph = section.AddParagraph()
```

5. Add a paragraph to the newly created section in the Word document by using the following code.

[C#]

```
// Insert text into the paragraph.  
paragraph.AppendText("Hello World!");
```

[VB .NET]

```
' Insert text into the paragraph.  
paragraph.AppendText("Hello World!")
```

6. Finally, save the Word document. The following code example illustrates how to do this.

[C#]

```
// Saving the document to disk.  
document.Save("Sample.doc", FormatType.Doc);
```

[VB.NET]

```
' Saving the document to disk.  
document.Save("Sample.doc", FormatType.Doc)
```

7. The sample Word document created through the above procedure is shown below.

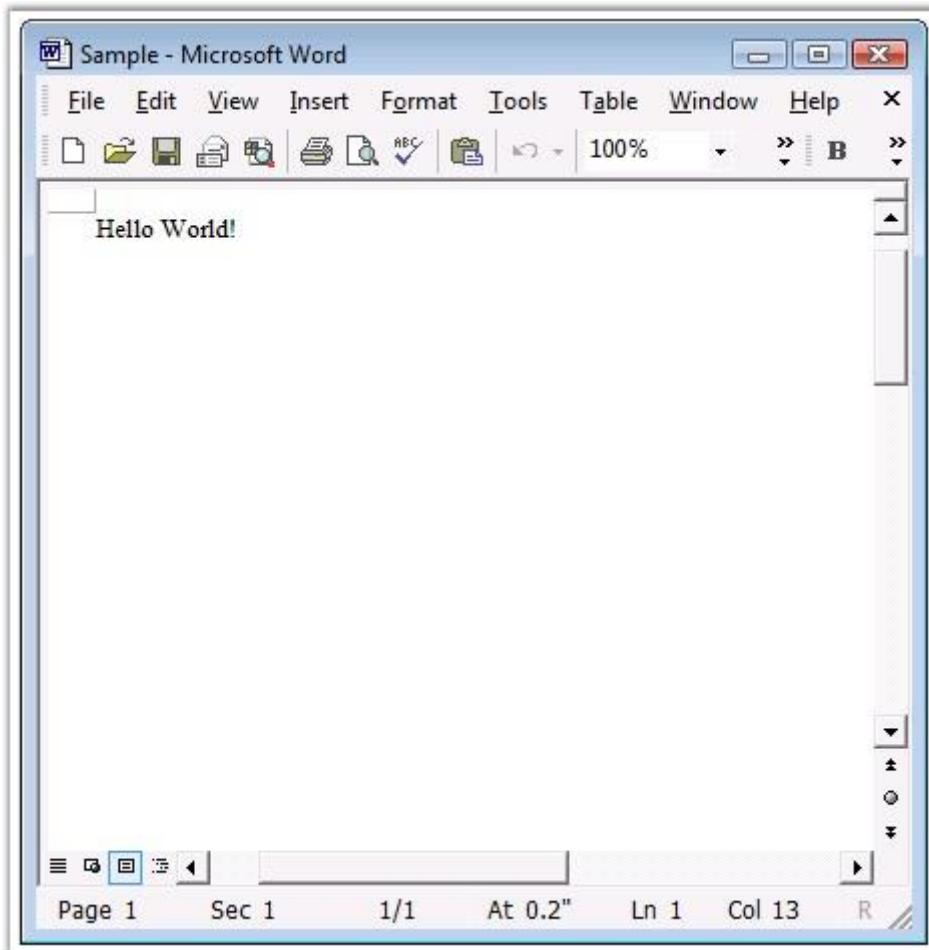


Figure 17: Word Document

A Word document is created in the Windows application.

3.3.2 ASP.NET

Now, you have created an ASP.NET application (refer to [Creating Platform Application](#)). This section covers the following:

- Deploying Essential DocIO in an ASP.NET Application
- Creating a Word Document

Deploying Essential DocIO in an ASP.NET Application

This section provides information and instructions for deploying ASP.NET applications that use Essential DocIO for ASP.NET. This is in addition to the section on Deploying Essential Studio for ASP.NET ([Common-->Deploying Essential Studio for ASP.NET](#)) in the Getting Started Guide. Essential DocIO ships with .NET Framework 2.0 (Visual Studio 2005) version of the C# and VB.NET samples which are installed beneath 2.0 directories. During installation, application directories are created in IIS for each of the C# and VB.NET samples.

Web application by default is deployed in full trust mode.

Deploying in Medium Trust or Partial Trust Scenarios

There are two such scenarios in which Syncfusion assemblies might be deployed.

Example 1

If the Syncfusion Assemblies are in GAC (Global Assembly Cache), and the Web Application is running in medium trust, then the Syncfusion assemblies actually runs in full trust. Hence this scenario is fully supported and there are no additional steps necessary for deployment.

Example 2

Say, the Syncfusion Assemblies are present in the application's bin folder and the Web Application is running in medium trust, then the Syncfusion assemblies will run in medium trust.

Essential DocIO allows to work in medium trust by using following assemblies.

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.DocIO.Web.dll

However, the following features are not currently supported under this scenario.

- OLE Object

The following steps will guide you to deploy Essential DocIO in an ASP.NET application:

1. **Marking the Application directory**-The appropriate directory, usually where the aspx files are stored, must be marked as Application in IIS.
2. **Syncfusion Assemblies**-The Syncfusion assemblies need to be in the bin folder that is beside the aspx files.



Note: They can also be in the GAC, in which case, they should be referenced in Web.config file.

[ASPX]

```
<configuration>
  <system.web>
    <compilation>
      <assemblies>
        <add assembly="Syncfusion.DocIO.Base, Version=X.X.X.X,
Culture=neutral, PublicKeyToken=3D67ED1F87D44C89"/></assemblies>
      </compilation>
    ...
  </system.web>
</configuration>
```



Note: X.X.X.X in the above code corresponds to the correct version number of the Essential Studio version that you are currently using.

3. **Data Files**-If you have .xml, .mdb, or other data files, ensure that they have sufficient security permission. Authenticated Users should have full control over the files and the directories in order to give ASP.NET code, enough permission to open the file during runtime.

Refer to the document in the following path, for step-by-step process of Syncfusion assemblies deployment in ASP.NET:

http://www.syncfusion.com/support/user/uploads/webdeployment_c883f681.pdf

 **Note:** Application with *Essential DocIO* needs following dependent assemblies for deployment.

- ***Syncfusion.Core.dll***
- ***Syncfusion.Compression.Base.dll***
- ***Syncfusion.DocIO.Base.dll***

Essential DocIO is now deployed in your ASP.NET application.

Creating a Word Document

The following code snippet will guide you to create and add a Word document to this application:

[C#]

```
// Include the following namespaces.  
using Syncfusion.DocIO.DLS;  
using Syncfusion.DocIO;  
  
// Create a new Word document.  
// This document has no section and no paragraph by default.  
WordDocument document = new WordDocument();  
  
// Add a new section to the document.  
IWSection section = document.AddSection();  
  
// Add a new paragraph to the section.  
IWParagraph paragraph = section.AddParagraph();  
  
// Insert Text into the paragraph  
paragraph.AppendText("Hello World!");  
  
// Saving the document to disk.  
document.Save("Sample.doc", FormatType.Doc);
```

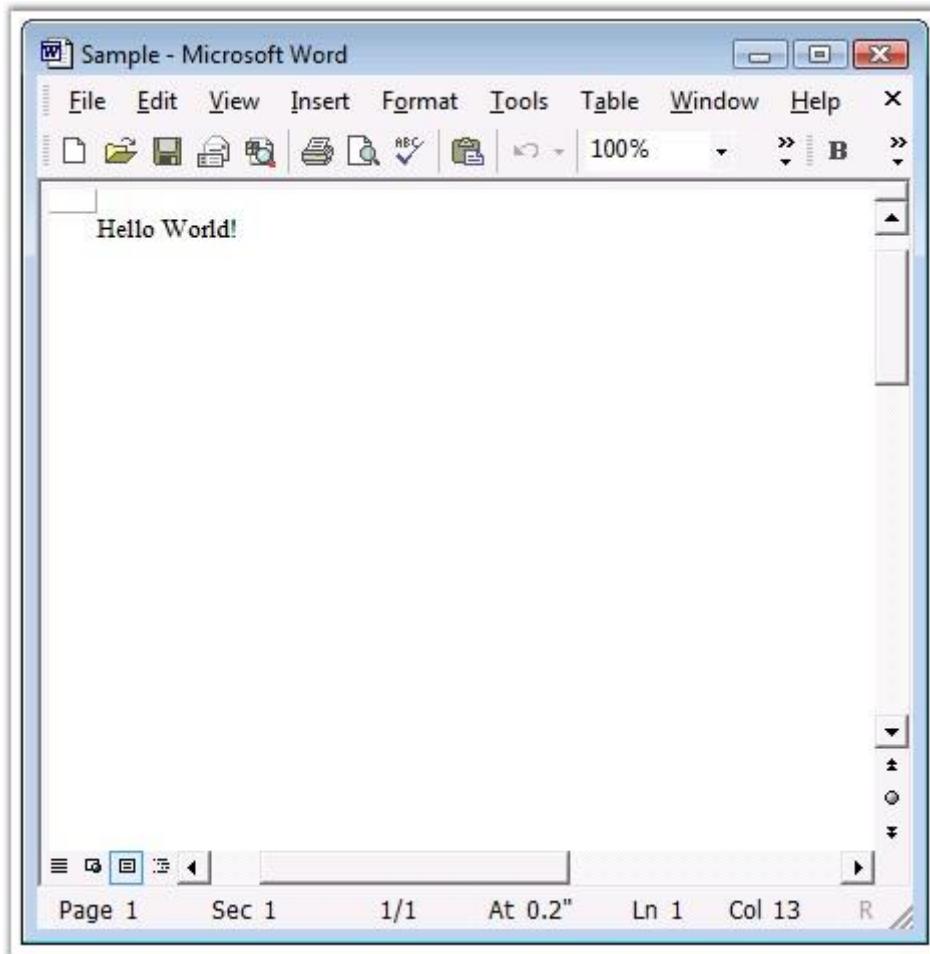


Figure 18: Word Document

A Word document is created in the ASP.NET application.

3.3.3 WPF

Now, you have created a WPF application (refer [Creating Platform Application](#)). This section covers the following:

- Deploying Essential DocIO in a WPF Application
- Creating a Word Document

Deploying Essential DocIO in a WPF Application

The following steps will guide you to deploy Essential DocIO:

1. Go to **Solution Explorer** of the application you have created. Right-click **Reference** folder and then click **Add References**.
2. Add the below mentioned assemblies as references in the application,
 - Syncfusion.Core.dll
 - Syncfusion.Compression.Base.dll
 - Syncfusion.DocIO.Base.dll



Note: There is no toolbox support for DocIO in WPF application.

Essential DocIO is now deployed in the WPF application.

Creating a Word Document

In this section, you will learn how to create a simple Word document with "Hello World" written on the first paragraph of the first section.

1. Include the following namespaces in your application.

- **Syncfusion.DocIO.DLS** (using Syncfusion.DocIO.DLS)
- **Syncfusion.DocIO** (using Syncfusion.DocIO)

2. Instantiate the **WordDocument** class.

[C#]

```
// Create a new Word document.  
// This document has no section and no paragraph by default.  
WordDocument document = new WordDocument();
```

[VB .NET]

```
' Create a new Word document.  
' This document has no section and no paragraph by default.  
Dim document As WordDocument = New WordDocument()
```



Note: The **WordDocument** class represents the Word documents created in memory. This is the memory representation of the Word document written to disk.

3. Add a section to the newly created Word document by using the following code.

[C#]

```
// Add a new section to the document.  
IWSection section = document.AddSection();
```

[VB .NET]

```
' Add a new section to the document.  
Dim section As IWSection = document.AddSection()
```

4. Add a paragraph to the newly created section in the Word document by using the following code.

[C#]

```
// Add a new paragraph to the section.  
IWParagraph paragraph = section.AddParagraph();
```

[VB .NET]

```
' Add a new paragraph to the section.  
Dim paragraph As IWParagraph = section.AddParagraph()
```

5. Add a paragraph to the newly created section in the Word document by using the following code.

[C#]

```
// Insert text into the paragraph.  
paragraph.AppendText("Hello World!");
```

[VB .NET]

```
' Insert text into the paragraph.  
paragraph.AppendText("Hello World!")
```

6. Finally, save the Word document. The following code example illustrates how to do this.

[C#]

```
// Saving the document to disk.
```

```
document.Save("Sample.doc", FormatType.Doc);
```

[VB.NET]

```
' Saving the document to disk.  
document.Save("Sample.doc", FormatType.Doc)
```

The sample Word document created through the above procedure is shown below.

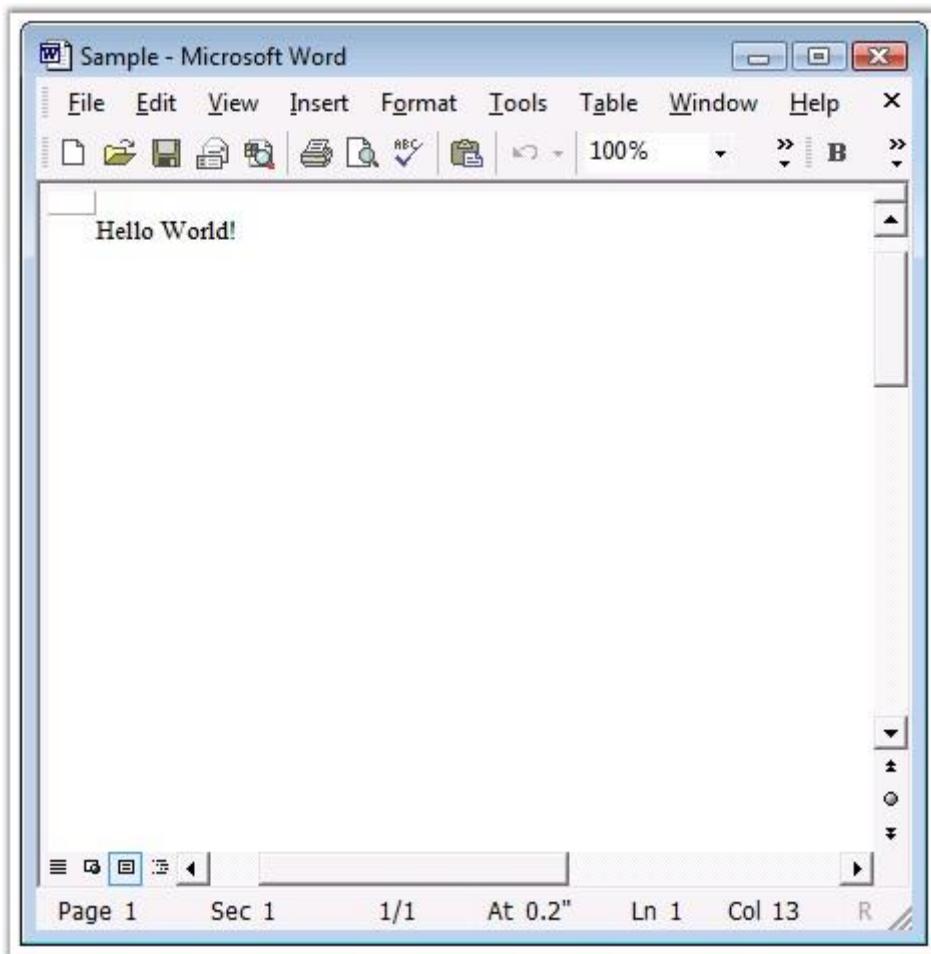


Figure 19: Word Document

A Word document is created in the WPF application.

3.3.4 Silverlight

Now, you have created a Silverlight application (refer [Creating Platform Application](#)). This section covers the following:

- Deploying Essential DocIO in a Silverlight Application
- Creating a Word Document

Essential DocIO has native support for creating and manipulating Word documents in a Silverlight application.



Note: DocIO provides support to create documents only in the .doc and .dot formats. It does not support the Word 2007 format.

Deploying Essential DocIO in a Silverlight Application

The following steps will guide you to deploy Essential DocIO:

1. Open the MainPage.xaml of the application in the designer.
2. Add the **Syncfusion.DocIO.Silverlight.dll** assembly as a reference to the application.

Essential DocIO is now deployed in your Silverlight application.

Creating a Word Document

Essential DocIO for Silverlight has support for creation and manipulation of richly formatted Word [97-2003] format documents. Advanced features like Mail merge, Search and replace can also be used in this approach.

Following steps will guide you to create a simple Word document with "Hello World" written on the first paragraph of the first section.

1. Add reference to the following assemblies in your Silverlight application.
 - Syncfusion.Compression.Silverlight.dll
 - Syncfusion.DocIO.Silverlight.dll
2. The next step is to add references to the following namespaces.
 - **Syncfusion.DocIO.DLS** (using Syncfusion.DocIO.DLS)
 - **Syncfusion.DocIO** (using Syncfusion.DocIO)

3. Instantiate the **WordDocument** class. This class represents the Word document that will be created in the memory. This is the memory representation of the Word document that will be written to the disk.

[C#]

```
// Create a new Word document.  
// This document has no section and no paragraph by default.  
WordDocument document = new WordDocument();  
  
// Add a new section to the document.  
IWSection section = document.AddSection();  
  
// Adding a new paragraph to the section.  
IWParagraph paragraph = section.AddParagraph();  
  
// Insert Text into the paragraph  
paragraph.AppendText("Hello World!");  
  
// Save the file to stream.  
SaveFileDialog sfd = new SaveFileDialog();  
sfd.DefaultExt = ".xls";  
sfd.Filter = "Files(*.xls)|*.xls";  
if (sfd.ShowDialog() == true)  
{  
    using (Stream stream = sfd.OpenFile())  
    {  
        document.Save(stream, FormatType.Doc);  
    }  
}
```

[VB .NET]

```
' Create a new Word document.  
' This document has no section and no paragraph by default.  
Dim document As WordDocument = New WordDocument()  
  
' Add a new section to the document.  
Dim section As IWSection = document.AddSection()  
  
' Adding a new paragraph to the section.  
Dim paragraph As IWParagraph = section.AddParagraph()  
  
' Insert Text into the paragraph  
paragraph.AppendText("Hello World!")
```

```
' Save the file to stream.  
Dim sfd As New SaveFileDialog()  
sfd.DefaultExt = ".xls"  
sfd.Filter = "Files(*.xls)|*.xls"  
If sfd.ShowDialog() = True Then  
    Using stream As Stream = sfd.OpenFile()  
        document.Save(stream, FormatType.Doc)  
    End Using  
End If
```



Note: Here, initially Word document has no section and no paragraph. You should add sections and paragraphs to write text on it. Save method of the WordDocument is used to save the created document to disk or stream to browser.

The following screen shot shows the Word document that is generated.

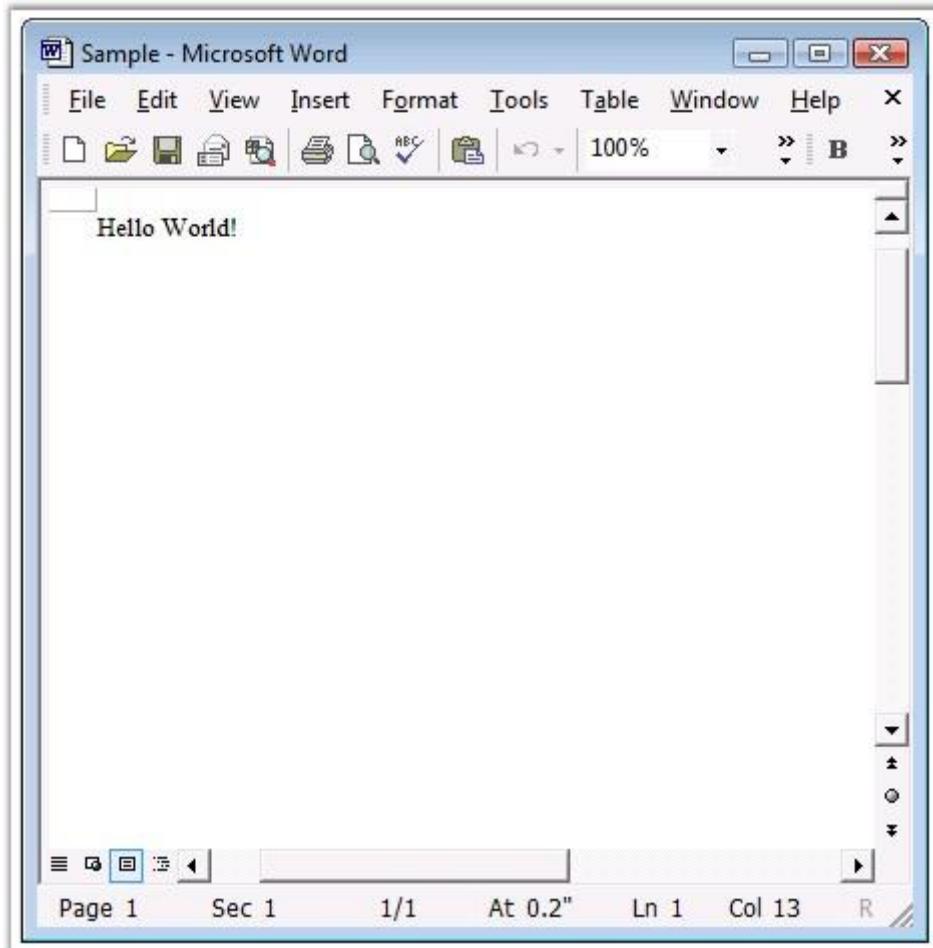


Figure 20: Word Document

A Word document is created in the Silverlight application.

3.3.5 ASP.NET MVC

Now, you have created an ASP.NET MVC application (refer [Creating Platform Application](#)). This section covers the following:

- Deploying Essential DocIO in an ASP.NET MVC Application
- Creating a Word document in an ASP.NET MVC Application

Deploying Essential DocIO in an ASP.NET MVC Application

The following steps will guide you to deploy Essential DocIO:

1. Add the Syncfusion.DocIO.Base assembly into the **Reference** folder to deploy Essential DocIO to the application.
2. Add the following assembly reference under **<compilation>** tag of Web.config file.

[ASPX]

```
<compilation debug="true">
  <assemblies>
    ...
    <add assembly="Syncfusion.DocIO.Base, Version=X.X.X.X, Culture=neutral,
      PublicKeyToken=3D67ED1F87D44C89"/>
    ...
  </assemblies>
</compilation>
```



Note: X.X.X.X in the above code corresponds to the version number of the Essential Studio version that you are currently using.

3. Add the following under **<namespaces>** tag of Web.config file.

[ASPX]

```
<namespaces>
...
<add namespace="Syncfusion.DocIO.Base"/>
</namespaces>
```

Essential DocIO is now deployed in your ASP.NET MVC application.

Creating a Word document in an ASP.NET MVC Application

The following steps illustrate how to create a Word document in an MVC application.

Step 1: View

Add a **Button** to the aspx page in View as follows.

[ASPX]

```
<%Html.BeginForm("HelloWorld", "GettingStarted", FormMethod.Post);%>


|                                                                                                                                 |                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| <input name="Browser" style="font-size:12px; font: Trebuchet MS" type="checkbox" value="Browser"/> Open Document inside Browser | <input style="margin-right: 3px; height:27px; font-size:12px; font: Trebuchet MS" type="submit" value="Create PDF"/> |
|---------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|


<% Html.EndForm();%>
```

- **Html.BeginForm("HelloWorld", "GettingStarted", FormMethod.Post)** represents the Form tag. When the form is submitted, the "HelloWorld" action in the "GettingStartedController" gets invoked.
- **HelloWorld** represents an Action.
- **GettingStarted** represents a controller.

Step 2: Create Custom ActionResult

In an MVC application, a controller action returns an action result. In particular, it returns an action that derives from the base ActionResult class.

But, if you want to return some other type of content to a browser, such as an image, a PDF file, or a Microsoft Excel document, you need to create your own action result. The following code example illustrates how to create a custom action result.

[C#]

```
public class DocResult : ActionResult
{
    private string m_filename;
    private IWordDocument m_document;
    private FormatType m_formatType;
    private HttpResponse m_response;
    private HttpContentDisposition m_contentDisposition;

    public string FileName
    {
        get
        {
            return m_filename;
        }
        set
        {
            m_filename = value;
        }
    }

    public IWordDocument Document
    {
        get
        {
            if(m_document!=null)
                return m_document;
            return null;
        }
    }

    public FormatType formatType
    {
        get
        {
            return m_formatType;
        }
    }
}
```

```

        set
    {
        m_formatType = value;
    }
}
public HttpContentDisposition ContentDisposition
{
    get
    {
        return m_contentDisposition;
    }
    set
    {
        m_contentDisposition = value;
    }
}
public HttpResponse Response
{
    get
    {
        return m_response;
    }
}

public DocResult(IWordDocument document, string filename, FormatType
formattype, HttpResponse response, HttpContentDisposition
contentDisposition)
{
    FileName = filename;
    m_document = document;
    this.formatType = formattype;
    m_response = response;
    this.ContentDisposition = contentDisposition;
}
public override void ExecuteResult(ControllerContext context)
{
    if (context == null)
        throw new ArgumentNullException("Context");
    this.Document.Save(FileName, formatType, Response, ContentDisposition);
}
}

```

Every action result must inherit from the base ActionResult class. The base ActionResult class has one method that you must implement: the **ExecuteResult** method. The ExecuteResult method is called to generate the content created by the action result. This method streams the output Word document file to the browser.

A controller action cannot return an action result directly. For example, if you want to return a view from a controller action, you don't return a ViewResult. Instead, you call the **View** method. The View method instantiates a new ViewResult and returns the new ViewResult to the browser.

Extension Methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. An extension method is a special kind of static method, but is called as if it were an instance method on the extended type.

Extension methods are defined as static methods, but are called by using the instance method syntax. The first parameter of this method specifies the type on which the method operates, and is preceded by the "this" modifier. Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.

An extension method named **ExportAsActionResult** has to be created to add the Controller class. The ExportAsActionResult method returns a DocResult.

[C#]

```
public static DocResult ExportAsActionResult(this IWordDocument document,
string filename, FormatType formattype, HttpResponse response,
HttpContentDisposition contentDisposition)
{
    return new DocResult(document, filename, formattype, response,
    contentDisposition);
}
```

In the preceding code example, one extension is defined for the type, IWordDocument.

Step 3: Controller

Add the following code in the Controller's action result.

[C#]

```
public ActionResult HelloWorld()
{
    return View();
}

[AcceptVerbs(HttpVerbs.Post)]
```

```
public ActionResult HelloWorld(string SaveOption, string Browser)
{
    // A new document is created.
    IWordDocument document = new WordDocument();

    // Add a new section to the document.
    IWSection section = document.AddSection();

    // Adding a new paragraph to the section.
    IWParagraph paragraph = section.AddParagraph();

    // Insert text into the paragraph.
    paragraph.AppendText("Hello World!");
    return document.ExportAsActionResult("Sample.doc", FormatType.Doc,
HttpContext.ApplicationInstance.Response,
HttpContentDisposition.InBrowser);
}
```

Step 4: Run the Application

Run the application. The following dialog box appears. It provides options to download and launch the generated file.



Figure 21: Word document being opened by using MVC Application

A Word document is created in the ASP.NET MVC application.

3.4 Saving the Word Document

This topic illustrates how to save the Word document created in an application.

Essential DocIO provides support to save the Word document to the following formats:

- Doc
- Docx
- Dot
- Rtf
- Html
- Text
- Xml
- Docm
- Dotm
- Dotx

Saving the Word document in Windows Forms and WPF Applications

To use DocIO in Windows Forms and WPF applications, you must save the created document to disk. The following code example illustrates this.

[C#]

```
// Saving the document to disk.  
doc.Save("Sample.doc", FormatType.Doc);
```

[VB .NET]

```
' Saving the document to disk.  
doc.Save("Sample.doc", FormatType.Doc)
```



Note: Essential DocIO also provides support to save the document into the Stream. For details, see [Stream Support](#).

Saving the Word document in ASP.NET Application

Essential DocIO is a Non-UI component that is used in Web applications. To use DocIO in an ASP.NET application, you must stream the created document to the client browser.

The following code example illustrates how to stream the document to the browser.

[C#]

```
// Streaming the document to the Browser.  
doc.Save("Sample.doc", FormatType.Doc, Response,  
HttpContentDisposition.InBrowser);  
  
// Streaming the document to the Browser.  
doc.Save("Sample.docx", FormatType.Docx, Response,  
HttpContentDisposition.InBrowser);
```

[VB.NET]

```
' Streaming the document to the Browser.  
doc.Save("Sample.doc", FormatType.Doc, Response,  
HttpContentDisposition.InBrowser)  
  
' For .docx format  
doc.Save("Sample.docx", FormatType.Docx, Response,  
HttpContentDisposition.InBrowser)
```

Saving the Word document in Silverlight Application

To use DocIO in a Silverlight application, you must save the created document to the disk. The following code example illustrates how to do this.

[C#]

```
SaveFileDialog sfd = new SaveFileDialog()  
{  
    DefaultExt = "doc",  
    FilterIndex = 1  
};  
if (sfd.ShowDialog() == true)  
{  
    using (Stream stream = sfd.OpenFile())  
    {  
        document.Save(stream, FormatType.Doc);  
    }
```

```
}
```

[VB.NET]

```
Dim sfd As New SaveFileDialog() With {.DefaultExt = "doc", .FilterIndex = 1}
If sfd.ShowDialog() = True Then
    Using stream As Stream = sfd.OpenFile()
        document.Save(stream, FormatType.Doc)
    End Using
End If
```

3.5 Feature Summary

Essential DocIO provides support for the following features:

General

- Creating and editing a new document.
- Modify existing MS Word documents.
- Open or Save a document as a Stream.
- Save a document as a Text file.
- Ability to create a docx Document.
- Ability to read docx file.
- Essential DocIO will support almost all the features in .docx which are available in .doc format.
- Ability to insert Ole objects from existing document to another document.
- Ability to Clone multiple documents and merge it into a single document.
- Ability to read and write Built-In and Custom Document properties.
- API to manipulate Page Setup settings.
- Ability to insert Vector Images.
- Ability to Watermark document with pictures and text.
- Ability to set the Background for a document.
- Ability to add or edit different types of Hyperlinks like Files, Images, Bookmarks and Email Hyperlinks.
- Ability to deploy applications with DocIO in medium trust.

Formatting Support

- Ability to create Formatted Tables in the document.
- Format text in the document.
- Ability to format content using Custom Styles.

Import and Export

- Ability to import HTML to Document
- Ability to export Document to HTML
- Ability to export Word Document to the Rich Text Format (.rtf)
- Ability to import RTF document into Word document
- Ability to export Word document into Pdf document
- Ability to export Word document into EPub document
- Ability to export Word document into text document
- Ability to import text document into Word document



Note: The above specified import and export features are not supported in Silverlight application.

Add/Remove Comments

- Ability to insert Comments.
- Ability to replace or remove the Commented Document Content.
- Ability to remove existing specific Comments or all the comments from the documents.

Page Header and Footer

- Insert document page Header and Footer.
- Ability to insert FootNotes and EndNotes.

Page Breaks

- Insert Page Breaks, Column Breaks, Section Breaks and Paragraph Breaks.

Mail Merge Support

- Advanced Mail Merge with different data sources like Data Table, Data View, Data Reader, and so on.
- Ability to perform Nested Mail Merge for Regions and also for Tables.

Find and Replace

- Ability to Find and Replace first occurrence of text.
- Added feature to Find and Replace control characters like Carriage Return, Newline, and so on.
- Advanced ability to Find and Replace text with its original formatting.

Security

- Ability to Protect the document.

- DocIO lets you Protect the Word document with password.

Bookmarks

- Ability to insert Bookmarks.
- Ability to navigate through bookmarks and replace it with paragraph items.

Bullets and Numbering

- Ability to insert Bullets and Numbering.
- Image Bullet preservation during doc to docx and docx to doc conversion.

Built-in Styles

- Ability to modify the Built-In Styles.

Form Fields

- Ability to create and fill Form Fields.

3.6 API Changes

The following lists the API changes for the DocIO.

- **OleObjectType** has been changed to '**OleLinkType**'.
- The values of the **OutlineLevel** property are of type '**Enumeration**' (no need to assign numeric values).

4 Concepts and Features

This tutorial illustrates how to get started with Essential DocIO. It will give you a basic introduction to the concepts you need to know before getting started with the product, and some tips and ideas on how to implement DocIO in your projects. Each category under this section explains about the various DLS entities of DocIO.

Note: Refer to the [Class Reference documentation](#) for comprehensive information on the classes.

This section covers information on the following:

4.1 Basic Concepts

Object model of Microsoft Word document is dendritic. WordDocument is the root of such a tree. Base, simplified hierarchy of document content can be shown as follows.

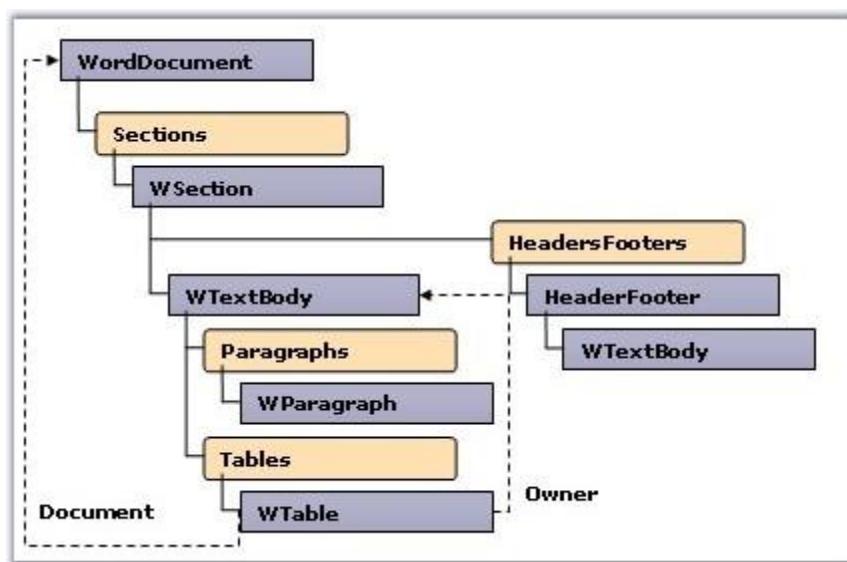


Figure 22: Object Model of Word Document

WordDocument and the rest of the nodes in such a tree (which are containers for content: text or graphics), are inherited from the abstract class, Entity. This class has the **EntityType** property, which defines the type of the node. Such an approach gives an opportunity to generalize the work with the nodes of the WordDocument tree.

Every element of a tree has a reference to the document it refers. Elements also have the **Owner** property, which shows whether the current element is attached to the document. If an element is not attached to the document, it won't be available in the resultant document after saving the document.

Composite Design Patterns

Overview

Object model of Word document in DocIO uses the idea of "Composite Design" pattern. The following screen shot illustrates the classic structure of the Composite Design pattern.

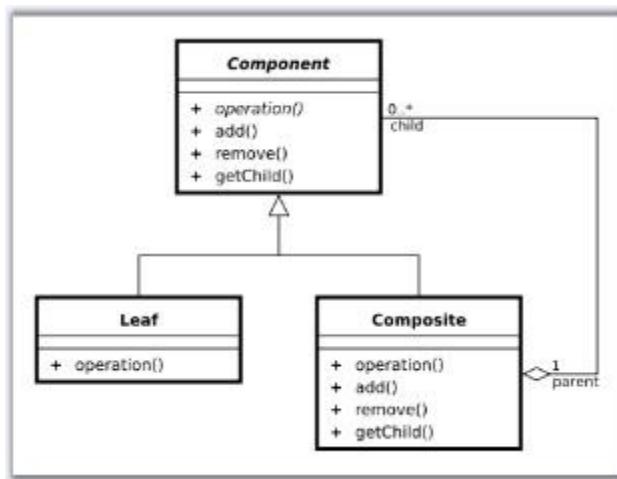


Figure 23: Composite Design Pattern

IEntity

IEntity interface represents the "Component" block in DocIO. **IEntity** interface supports all the elements which have content. Composite block is represented by the **ICompositeEntity** interface. Composite block has child nodes. Classes which implement only the **IEntity** (which don't have child nodes) are "leafs" for DocIO.

IEntity interface has some specific properties.

- **IsComposite**: defines whether the current element is composite (elements which have child nodes are composite).
- **NextSibling**: returns the next sibling. For example, in the following screen shot, the **NextSibling** property for Child node 2 will return Child node 3 element.
- **PreviousSibling**: returns the previous sibling. For example, in the following screen shot, the **PreviousSibling** property for Child node 2 will return Child node 1 element.

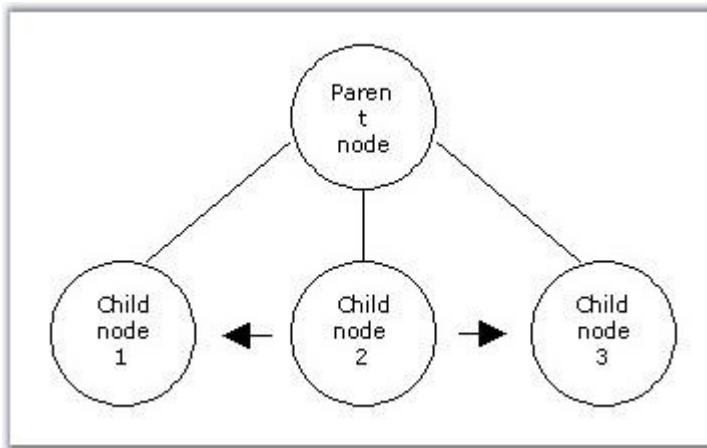


Figure 24: IEntity Interface

IEntity Public Properties

Name	Description
Document	Gets document of this entity.
EntityType	Gets the type of the entity.
IsComposite	Gets a value indicating whether this instance is composite.
NextSibling	Gets the next sibling.
Owner	Gets owner of this entity.
PreviousSibling	Gets the previous sibling.

IEntity Public Methods

Name	Description
Clone	Creates a duplicate of the entity.

ICompositeEntity

ICompositeEntity interface represents an element which has "children". A child element is an element that supports the ICompositeEntity (and also has children) or a "leaf" (element without children).

ICompositeEntity Public Property

Name	Description
ChildEntities	Gets the child entities.

4.2 Word Document

You can open, modify and create Microsoft Word documents by using the **WordDocument** class. WordDocument class models the structure of a Microsoft Word document.

Creating a Word Document

To create a new document, use the **EnsureMinimal** method. This method creates a document with an empty section, and adds empty paragraphs to the newly created section.

Opening a Word Document

DocIO also gives you an opportunity to open existing documents, or read data from streams saved in the following FormatType variants.

- **Doc**: Microsoft Word File Format
- **Txt**: Text File Format
- **Docx**: Word 2007 File Format
- **Dot**: Word Template Format
- **HTML**: HTML Format
- **RTF**: Rich Text Format
- **Docm** – Word Macro-enabled Document Format
- **Dotm** – Word Macro-enabled Template Format
- **Dotx** – Word Open xml Template Format

To open a document, use the **Open** method. There are several overloads for this method.

- **Open(string fileName)**: opens Word document.
- **Open(string fileName, FormatType formatType)**: opens the document with specified format type (.doc, .xml or .txt file).
- **Open(string fileName, FormatType formatType, string password)**: opens the encrypted document with a specified format type and password.
- **Open(string fileName, FormatType formatType, XHTMLValidationType validationType)**: opens the document with a specified format type and XHTMLValidationType. Ignores validation type if the input document is not XHTML.
- **Open(string fileName, FormatType formatType, XHTMLValidationType validationType, string baseUrl)**: opens the document with specified format type, XHTMLValidationType and the baseUrl of the input document. Ignores validation type and baseUrl if the input document is not HTML.

- **Open(Stream stream, FormatType formatType)**: opens the document from the stream which has the specified format type.
- **Open(Stream stream, FormatType formatType, string password)**: opens the encrypted document from the stream that has the specified format type and the password.
- **Open(Stream stream, FormatType formatType, XHTMLValidationType validationType)**: opens the document from the stream that has the specified format type and XHTMLValidationType. Ignores validation type if the input document is not XHTML.
- **Open(Stream stream, FormatType formatType, XHTMLValidationType validationType, string baseUrl)**: opens the document from the stream that has the specified format type, XHTMLValidationType, and the baseUrl of the input document. Ignores validation type and baseUrl if the input document is not HTML.

The following asynchronous open methods are specific to Windows Store applications.

- **OpenAsync(StorageFile file)**: opens the document asynchronously from the storage file.
- **OpenAsync (StorageFile file, FormatType formatType)**: opens the document asynchronously from the storage file that has the specified format type.
- **OpenAsync(Stream stream, FormatType formatType)**: opens the document asynchronously from the stream that has the specified format type.
- **OpenAsync(StorageFile file, FormatType formatType, string password)**: opens the document asynchronously from the storage file that has the specified format type and the password.
- **OpenAsync(Stream stream, FormatType formatType, string password)**: opens the document asynchronously from the stream that has the specified format type and the password.

To save a document back to the Word document format use the **Save** method. There are several overloads for this method.

- **Save(string fileName)**: saves to file in Microsoft Word format.
- **Save(string fileName, FormatType formatType)**: saves the document to file in .xml or Microsoft Word format.
- **Save(Stream stream, FormatType formatType)**: saves the document into stream in .xml or Microsoft Word format.
- **Save(string fileName, FormatType formatType, HttpResponse response, HttpContentDisposition contentDisposition)**: saves the document into the client browser.

The following asynchronous save methods are specific to Windows Store applications.

- **SaveAsync(StorageFile file, FormatType formatType)**: saves the document asynchronously into the storage file in the specified format type.

- **SaveAsync(Stream stream, FormatType formatType)**: saves the document asynchronously into the stream in the specified format type.

To open the document which is already opened in Word, use the **OpenReadOnly** method to open the document in the read-only mode.

- **OpenReadOnly(string fileName, FormatType formatType)**: opens the document in the read-only mode.

Public Properties

Property	Description
Background	Gets document's background.
Bookmarks	Gets document bookmarks.
Built-inDocumentProperties	Gets document built-in properties object.
ChildEntities	Gets the child entities.
CustomDocumentProperties	Gets document custom properties object.
EndnoteNumberFormat	Gets or sets endnote numbering format.
EndnotePosition	Gets or sets endnote position in the document.
EntityType	Gets the type of the entity.
FootnoteNumberFormat	Gets or sets footnote numbering format.
FootnotePosition	Gets or sets footnote position in the document.
InitialEndnoteNumber	Gets or sets the initial endnote number.
InitialFootnoteNumber	Gets or sets the initial footnote number.
LastParagraph	Gets last section object.
LastSection	Gets last section of the document.
ListStyles	Gets document list styles.
MailMerge	Gets mail merge engine.
ProtectionType	Gets or sets the type of protection of the

	document.
RestartIndexForEndnote	Gets or sets the restart index for endnote.
RestartIndexForFootnotes	Gets or sets the restart index for footnotes.
Sections	Gets document sections.
Styles	Gets document styles.
TextBoxes	Gets or sets textbox items of main document.
ThrowExceptionsForUnsupportedElements	Gets or sets whether to throw exceptions for unsupported elements.
ViewSetup	Gets view setup options in MSWord.
Watermark	Gets or sets document's watermark.
FontSubstitutionTable	Gets or sets a dictionary object which represents the font substitution table. The key should be the font name and the value should be its corresponding alternate font name.
HasMacros	Determines whether the document has Macros.
AttachedTemplate	Gets the attached template.
UpdateStylesOnOpen	Gets or sets a value indicating whether to automatically update the styles of a document from the attached template each time the document is opened.
Footnotes	Gets or sets the footnote separators.
Endnotes	Gets or sets the endnote separators.

Public Constructors

Name	Description
WordDocument.WordDocument ()	Initializes a new instance of the WordDocument class.

WordDocument.WordDocument (Stream)	Initializes a new instance of the WordDocument class from the stream.
WordDocument.WordDocument (Stream, FormatType, string)	Initializes a new instance of the WordDocument class from the stream of specified type, protected with password.
WordDocument.WordDocument (Stream, string)	Initializes a new instance of the WordDocument class from the Word document's stream, which is protected with password.
WordDocument.WordDocument (string)	Initializes a new instance of the WordDocument class from Word document.
WordDocument.WordDocument (string, FormatType, string)	Initializes a new instance of the WordDocument class from existing file of specified type protected with password.
WordDocument.WordDocument (string, string)	Initializes a new instance of the WordDocument class from existing Word document, which is protected with password.

Public Methods

Name	Description
AddListStyle	Adds new list style to document.
AddParagraphStyle	Adds new paragraph style to the document.
AddSection	Adds new section to document.
Clone	Clones itself.
CreateParagraph	Creates the paragraph.
CreateParagraphItem	Creates new paragraph item instance.
EnsureMinimal	Adds one empty section to the document and one empty paragraph to created section.
Find	Finds the first entry of specified string.
FindAll	Finds all entries of specified string.

GetText	Gets the document's text.
ImportContent	Imports all content into document.
ImportSection	Imports section into document.
Open	Opens doc file.
OpenTxt	Opens the document in text format.
OpenXml	Opens the document in xml format.
Replace	Replaces all entries of given string.
Save	Saves WordDocument instance to the specified file format.
OpenReadOnly	Open the document in ReadOnly mode.
UpdateDocumentFields()	Updates the fields present in the document.
RemoveMacros	Removes the macros in the document

The following example illustrates how to use the Open and Save methods.

[C#]

```
//Open the Word document
WordDocument sourceDoc = new WordDocument();
sourceDoc.Open("SourceDocument.doc", FormatType.Doc);

//Create a new word document with one section and one paragraph
WordDocument doc = new WordDocument();
doc.EnsureMinimal();

//Clone the content of source document to the newly created document
doc = sourceDoc.Clone();

//Save the document as xml
doc.Save("Document.doc", FormatType.Doc);
```

[VB.NET]

```
'Open the Word document
```

```
Dim sourceDoc As WordDocument = New WordDocument()
sourceDoc.Open("SourceDocument.doc", FormatType.Doc)

'Create a new word document with one section and one paragraph
Dim doc As WordDocument = New WordDocument()
doc.EnsureMinimal()

'Clone the content of source document to the newly created document
doc = sourceDoc.Clone()

'Save the document as xml
doc.Save("Document.doc", FormatType.Doc)
```

The following code illustrates how to use the **Open** and **Save Async** methods in Windows Store applications.

[C#]

```
//Open the Word document from the memory stream.
WordDocument doc = new WordDocument();
doc.OpenAsync(memoryStream, FormatType.Doc);

//Open the Word document from the storage file.
WordDocument doc = new WordDocument();
doc.OpenAsync(storageFile, FormatType.Doc);

//Save the document in the memory stream.
doc.SaveAsync(memoryStream, FormatType.Doc);

//Save the document in the storage file.
doc.SaveAsync(storageFile, FormatType.Doc);
```

[VB .NET]

```
'Open the Word document from the memory stream.
Dim doc As WordDocument = New WordDocument()
sourceDoc.OpenAsync(memoryStream, FormatType.Doc);

'Open the Word document from a storage file.
Dim doc As WordDocument = New WordDocument()
sourceDoc.OpenAsync(storageFile, FormatType.Doc)

'Save the document in a memory stream.
doc.SaveAsync(memoryStream, FormatType.Doc)
```

```
'Save the document in a storage file.  
doc.SaveAsync(storageFile, FormatType.Doc)
```

For more information, refer to the following:

[Document Properties](#), [Document Background](#), [Docx Support](#), [Stream Support](#)

4.2.1 Document Properties

Document Properties contain general information about the document like author of the document, subject, character count, word count, page count, creation date and so on.

To view or edit the document properties, go to the **File** menu and click **Properties**. The Document Properties dialog box will appear as follows.



Figure 25: Document Properties Dialog Box

The Document Properties dialog box contains the following tabs.

- **General** (not editable): contains general document information such as file name, type, size, location, creation, modification and access date
- **Summary** (editable): set or modify the document properties such as author name, title, company, subject, keywords, and so on
- **Statistics** (not editable): displays the statistics of the document such as creation, modification, access and print dates, revision number, characters, words, pages number, and so on
- **Contents** (not editable): displays the content of the document
- **Custom** (editable): enables you to create your own properties combining their name, type and value. For example, you may want to mark some document as "Checked". You can do it by creating a custom property with the Name as "Checked", Type as "Text", and Value as "your name".

Built-in Properties

BuiltinDocumentProperties class represents all document properties, excluding custom properties.

Class Hierarchy

SummaryDocumentProperties

|

BuiltinDocumentProperties

Public Properties

Name	Description
BytesCount	Represents the number of bytes in the document.
Category	Gets or sets the category of the document.
Company	Gets or sets Company property.
HiddenCount	Gets or sets hidden count.
LinesCount	Gets or sets the number of lines in the document.
Manager	Gets or sets Manager property.
NoteCount	Gets or sets Note count.
ParagraphCount	Gets or sets the number of paragraphs in the document.
SlideCount	Gets or sets slide count.

Public Methods

Name	Description
Clone	Clones itself.

The following example illustrates how to get, set and modify the document properties.

[C#]

```
WordDocument doc = new WordDocument();
doc.Open( "DocumentProperties.doc" );

string author = doc.BuiltinDocumentProperties.Author;
int bytesCount = doc.BuiltinDocumentProperties.BytesCount;
doc.BuiltinDocumentProperties.Keywords += "document properties";

doc.BuiltinDocumentProperties.Author = "Author's name";
doc.BuiltinDocumentProperties.Comments = "Document comments";
```

[VB .NET]

```
Dim doc As WordDocument = New WordDocument()
doc.Open("DocumentProperties.doc")

Dim author As String = doc.BuiltinDocumentProperties.Author
Dim bytesCount As Integer = doc.BuiltinDocumentProperties.BytesCount
doc.BuiltinDocumentProperties.Keywords &= "document properties"

doc.BuiltinDocumentProperties.Author = "Author's name"
doc.BuiltinDocumentProperties.Comments = "Document comments"
```

Custom Properties

CustomDocumentProperties class enables you to create and save custom properties. It contains a collection of DocumentProperty class instances. You can access a document property by indexing, i.e., by specifying the property name or index.

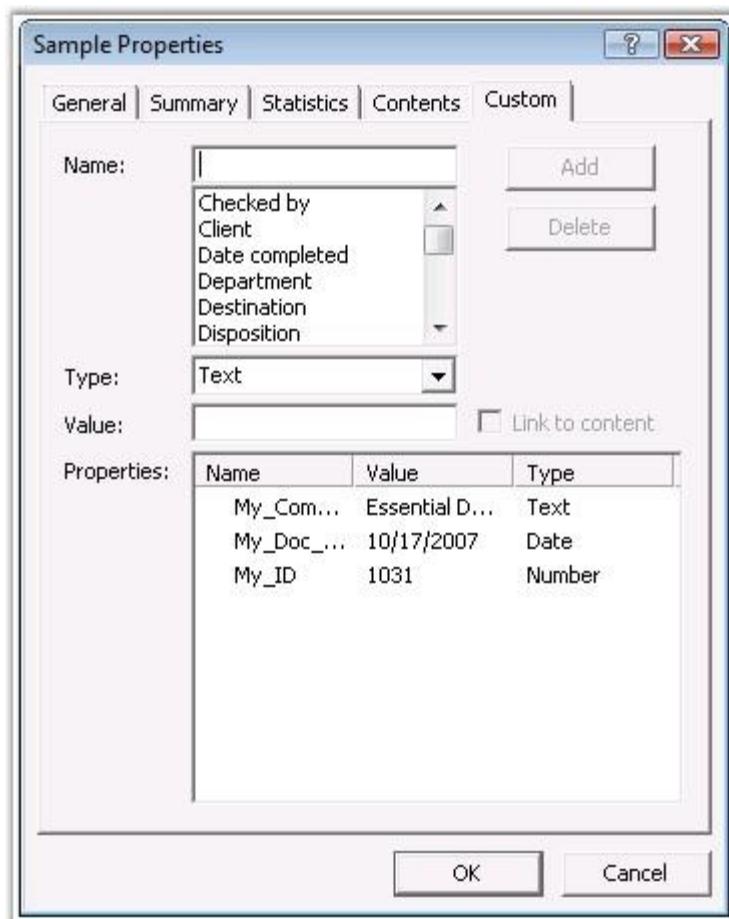


Figure 26: Custom Tab in Document Properties Dialog Box

Public Properties

Name	Description
Count	Gets count of the properties.

Public Methods

Name	Description
Add	Adds a new custom property.
Clone	Clones itself.
Remove	Remove property specified by name.

DocumentProperty

DocumentProperty class represents each custom document property.

Public Methods

Name	Description
Clone	Clones itself.
ToBool	Converts value to Boolean.
ToByteArray	Converts value to byte array.
ToDateTime	Converts value to DateTime.
ToFloat	Converts value to float.
ToInt	Converts value to integer.
ToString	Converts value to string.

The following example illustrates how to get or set the existing custom document properties, and also how to add new custom document properties.

[C#]

```
WordDocument doc = new WordDocument();
doc.Open( "DocumentProperties.doc" );

// Getting custom property value
int phoneNumber = doc.CustomDocumentProperties["Telephone number"].ToInt();

// Setting existent custom property value
doc.CustomDocumentProperties["Check by"].Value = "user name";

// Adding new custom property
DateTime completedDate = DateTime.Now;
doc.CustomDocumentProperties.Add( "Date completed", completedDate );
```

[VB.NET]

```

Dim doc As WordDocument = New WordDocument()
doc.Open("DocumentProperties.doc")

' Getting custom property value
Dim phoneNumber As Integer = doc.CustomDocumentProperties("Telephone number")
.ToInt()

' Setting existent custom property value
doc.CustomDocumentProperties("Check by").Value = "user name"

' Adding new custom property
Dim completedDate As DateTime = DateTime.Now
doc.CustomDocumentProperties.Add("Date completed", completedDate)

```

4.2.2 Document Background

Background and **Watermark** classes represent the background effects in the Word document.

To set the background effects in Word, open the **Format** menu and click **Background**. The following are the background effects available in Word.

- Fill Color
- Fill Effects (Gradient, Texture, Pattern, Picture)
- Printed Watermark

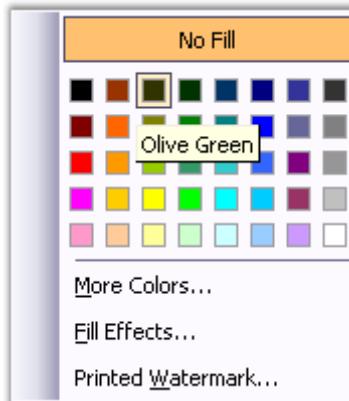


Figure 27: Background and Watermark classes in the Word document

Note: A document can also have a printed watermark and background effect (fill color or one of fill effects) at the same time.

4.2.2.1 Document Color

Background class represents the background color and fills effects in the Word document. The type of the background effect is defined by using the Type property. This property returns the value of `BackgroundType` type, and it can take the following variants.

- **NoBackground**: no background fill effect
- **Gradient**: gradient fill effect
- **Picture**: background picture
- **Texture**: texture fill effect
- **Color**: color fill effect



Note: *Pattern fill effect is not supported.*

Picture property defines the picture to be reflected as the document background (in case background type is set to `BackgroundType.Picture`). If the background type is `BackgroundType.Texture`, the background picture will be used as a picture for texture. So the background picture must be present in such cases.

Color property defines the color to be reflected as the document background (in case background type is set to `BackgroundType.Color`).

Gradient property defines the gradient to be reflected as the document background (in case background type is set to `BackgroundType.Gradient`). Gradient property returns the object of the **BackgroundGradient** class. For more details on `BackgroundGradient` class, refer the `BackgroundGradient` documentation.

WordDocument.Background property is used to access the DocIO document background. Background property of `WordDocument` is automatically initialized. If there is no background in the default DocIO document, it means that the Type property of the `Background` is set to `NoBackground`.

Public Properties

Name	Description
Color	Gets or sets the background color.
Gradient	Gets or sets the background gradient.
Picture	Gets or sets the background picture.

Type	Gets the type of the background effect for the document.
------	--

The following example illustrates how to use the Background class.

[C#]

```
IWordDocument doc1 = new WordDocument();
doc1.Open("Background.doc");
IWordDocument doc2 = new WordDocument();
doc2.EnsureMinimal();

switch (doc1.Background.Type)
{
    case BackgroundType.Gradient:
        doc2.Background.Gradient = doc1.Background.Gradient.Clone();
        break;

    case BackgroundType.Picture:
    case BackgroundType.Texture:
        doc2.Background.Picture = doc1.Background.Picture;
        break;

    case BackgroundType.Color:
        doc2.Background.Color = doc1.Background.Color;
        break;

    default: break;
}

doc2.Background.Type = doc1.Background.Type;
doc1.Background.Type = BackgroundType.Color;
doc1.Background.Color = Color.Red;

doc1.Save("Background.doc");
doc2.Save("BackgroundNew.doc");
```

[VB.NET]

```
Dim doc1 As IWordDocument = New WordDocument()
doc1.Open("Background.doc")
Dim doc2 As IWordDocument = New WordDocument()
doc2.EnsureMinimal()

Select Case doc1.Background.Type
    Case BackgroundType.Gradient
        doc2.Background.Gradient = doc1.Background.Gradient.Clone()

    Case BackgroundType.Picture, BackgroundType.Texture
        doc2.Background.Picture = doc1.Background.Picture

    Case BackgroundType.Color
        doc2.Background.Color = doc1.Background.Color

    Case Else
End Select

doc2.Background.Type = doc1.Background.Type
doc1.Background.Type = BackgroundType.Color
doc1.Background.Color = Color.Red

doc1.Save("Background.doc")
doc2.Save("BackgroundNew.doc")
```

Background Gradient

Background Gradient class represents the background gradient fill effect in the Word document. To set the gradient by using Word menu, open the **Format** menu, point to **Background, Fill Effects**, and then click **Gradient**.

The following screen shot shows the **Fill Effects** dialog box.

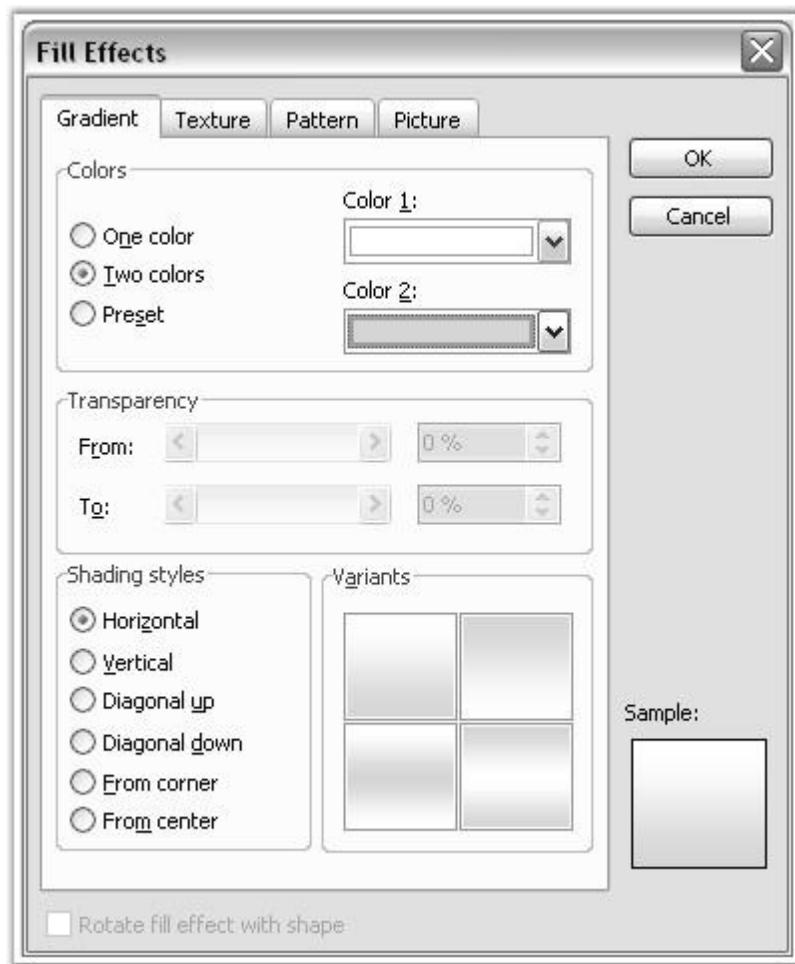


Figure 28: Fill Effects Dialog Box

Using DocIO, you can access background gradient options through the **WordDocument.Background.Gradient** option. Background Gradient will be set as the background fill effect when the **WordDocument.Background.Type** option is set to **BackgroundType.Gradient**.

Color1 and **Color2** properties of **Background Gradient** define the gradient colors. **GradientShadingStyle** and **GradientShadingVariant** properties define the type of the different variants of the gradient.

Public Properties

Name	Description
Color1	Gets or sets first color for gradient.

Color2	Gets or sets second color for gradient (used when TwoColors set to true).
ShadingStyle	Gets or sets shading style for gradient.
ShadingVariant	Gets or sets shading variants.

Public Methods

Name	Description
Clone	Clones current Gradient object.

The following example illustrates how to use the Background Gradient class.

[C#]

```
IWordDocument doc = new WordDocument(true);
doc.Background.Type = BackgroundType.Gradient;
doc.Background.Gradient.Color1 = Color.White;
doc.Background.Gradient.Color2 = Color.Black;
doc.Background.Gradient.ShadingStyle = GradientShadingStyle.FromCenter;
doc.Background.Gradient.ShadingVariant = GradientShadingVariant.ShadingDown;
```

[VB]

```
Dim doc As IWordDocument = New WordDocument(True)
doc.Background.Type = BackgroundType.Gradient
doc.Background.Gradient.Color1 = Color.White
doc.Background.Gradient.Color2 = Color.Black
doc.Background.Gradient.ShadingStyle = GradientShadingStyle.FromCenter
doc.Background.Gradient.ShadingVariant = GradientShadingVariant.ShadingDown
```

4.2.2.2 Watermark

Watermark class represents printed watermark in the Word document.

To insert a watermark into the Word document, open the **Format** menu, point to **Background**, and then click **Printed Watermark**. There are two types of watermarks.

- Picture Watermark

- Text Watermark

The type of watermark is defined by using the **Type** property. This property returns the object of type, WatermarkType. It includes the following options.

- **NoWatermark**: document does not have watermark
- **PictureWatermark**: document has picture watermark
- **TextWatermark**: document has text watermark

You can create a Picture Watermark or Text Watermark, but you cannot create an object of the Watermark class.

Watermark is a paragraph item. It is found in the first paragraph of the header / footer subdocument. DocIO Watermark is accessible through the **WordDocument.Watermark** property.

Class Hierarchy

ParagraphItem

|

Watermark

Public Properties

Name	Description
EntityType	Gets the type of the entity.
Type	Gets or sets the watermark type.

Picture Watermark

PictureWatermark class represents the picture watermark in the Word document.



Figure 29: Selecting Picture Watermark in Printed Watermark Dialog Box

Picture property defines the picture to be used as the watermark. **Scaling** property defines the watermark scaling (in percents). **Washout** property defines whether the washout effect is to be applied to the watermark. Default value for **Washout** property is set to **True**.

Class Hierarchy

ParagraphItem

|

Watermark

|

PictureWatermark

Public Constructors

Name	Description
PictureWatermark.PictureWatermark()	Initializes a new instance of the PictureWatermark class.
PictureWatermark.PictureWatermark(Image, bool)	Initializes a new instance of the PictureWatermark

	class.
--	--------

Public Properties

Name	Description
EntityType	Gets the type of the entity.
Picture	Gets or sets picture for Picture watermark.
Scaling	Gets or sets picture scaling in percents.
Washout	Gets or sets Washout property for Picture watermark.

The following example illustrates how to use the PictureWatermark class.

[C#]

```
IWordDocument doc = new WordDocument();
doc.EnsureMinimal();
PictureWatermark picWatermark = new PictureWatermark();
picWatermark.Scaling = 120f;
picWatermark.Washout = true;
doc.Watermark = picWatermark;
picWatermark.Picture = Image.FromFile(ImagesPath + "Water lilies.jpg");
```

[VB .NET]

```
Dim doc As IWordDocument = New WordDocument()
doc.EnsureMinimal()
Dim picWatermark As PictureWatermark = New PictureWatermark()
picWatermark.Scaling = 120f
picWatermark.Washout = True
doc.Watermark = picWatermark
picWatermark.Picture = Image.FromFile(ImagesPath & "Water lilies.jpg")
```

Text Watermark

TextWatermark class represents the text watermark in the Word document. The following screen shot illustrates the Text Watermark settings.



Figure 30: Selecting Text Watermark in Printed Watermark Dialog Box

Text property defines the text of the Text Watermark. **FontName** property defines the name of the text font. Default font name is **Times New Roman**. **Size** property defines the font size. Default font size is **36**. **Color** property defines the color of the text (font). Default font color is **Color.Gray**. **Semitransparent** property defines whether the text watermark is semi-transparent. Default value for this property is set to **True**.

Layout property defines the layout for the watermark.

- **Diagonal:** diagonal watermark layout
- **Horizontal:** horizontal watermark layout

Layout property returns the value of the **WatermarkLayout** type. Default layout is **Diagonal**.

Class Hierarchy

ParagraphItem

|

Watermark

|

TextWatermark

Public Properties

Name	Description
TextWatermark.TextWatermark ()	Initializes a new instance of the TextWatermark class.
TextWatermark.TextWatermark (string)	Initializes a new instance of the TextWatermark class.
TextWatermark.TextWatermark (string, string, int, WatermarkLayout)	Initializes a new instance of the TextWatermark class.

Public Properties

Name	Description
Color	Gets or sets text watermark color.
EntityType	Gets the type of the entity.
FontName	Gets or sets watermark text's font name.
Layout	Gets or sets layout for Text watermark.
Semitransparent	Gets or sets semitransparent property for Text watermark.
Size	Gets or sets the text watermark size (in points).
Text	Gets or sets watermark text.

The following example illustrates how to use the TextWatermark class.

[C#]

```
IWordDocument doc = new WordDocument();
doc.EnsureMinimal();
TextWatermark textWatermark = new TextWatermark();
doc.Watermark = textWatermark;
textWatermark.Size = 96;
textWatermark.Layout = WatermarkLayout.Horizontal;
```

```
textWatermark.Semitransparent = false;
textWatermark.Color = Color.Black;
textWatermark.Text = "TextWatermark";
```

[VB.NET]

```
Dim doc As IWordDocument = New WordDocument()
doc.EnsureMinimal()
Dim textWatermark As TextWatermark = New TextWatermark()
doc.Watermark = textWatermark
textWatermark.Size = 96
textWatermark.Layout = WatermarkLayout.Horizontal
textWatermark.Semitransparent = False
textWatermark.Color = Color.Black
textWatermark.Text = "TextWatermark"
```

4.2.3 Docx Support

Essential DocIO provides support for creating Microsoft Word 2007, Microsoft Word 2010, and Microsoft Word 2013 format files from scratch by using the DocIO API. Word 2007, Word 2010, and Word 2013 format files are created with the same API as that of .doc files; the only difference is the format type in which they are saved and opened.

Note: The format type option “Docx” specifies the Microsoft Word 2013 format and its usage is deprecated. We recommend using Word 2007, Word 2010, or Word 2013 .

The following code demonstrates how to create a Word 2007 format document.

[C#]

```
WordDocument doc = new WordDocument();
//Add a new section to the document.
IWSection section = document.AddSection();
//Adding a new paragraph to the section.
IWPParagraph paragraph = section.AddParagraph()
//Insert text into the paragraph.
paragraph.AppendText( "Hello World!" );
//Saves the document in Word 2007 format type.
doc.Save("OutDocument.docx", FormatType.Word2007);
```

[VB.NET]

```
Dim doc As New WordDocument()
'Add a new section to the document.
Dim section As IWSection = document.AddSection()
'Adding a new paragraph to the section.
Dim paragraph As IWParagraph = section.AddParagraph()
'Insert text into the paragraph.
paragraph.AppendText("Hello World!")
'Saves the document in Word 2007 format type.
doc.Save("OutDocument.docx", FormatType.Word2007)
```

The following code illustrates how to open a Word 2010 format document and save it.

[C#]

```
WordDocument doc = new WordDocument();
//Opens the Word 2010 format document.
document.Open(filename, FormatType.Automatic);
//Adding a new paragraph to the last section.
IWParagraph paragraph = document.LastSection.AddParagraph()
//Insert text into the paragraph.
paragraph.AppendText( "Creating Word 2010 format document!" );
//Saves the document in Word 2010 format type.
doc.Save("Sample.docx", FormatType.Word2010);
```

[VB.NET]

```
Dim doc As New WordDocument()
'Opens the Word 2010 format document.
document.Open(filename, FormatType.Automatic);
'Adding a new paragraph to the last section.
Dim paragraph As IWParagraph = document.LastSection.AddParagraph()
'Insert text into the paragraph.
paragraph.AppendText("Creating Word 2010 format document!")
```

```
'Saves the document in Word 2007 format type.  
doc.Save("Sample.docx", FormatType.Word2010)
```

The following code illustrates how to open a Word 2013 format document and save it.

[C#]

```
WordDocument doc = new WordDocument();  
//Opens the Word 2013 format document.  
document.Open(filename, FormatType.Automatic);  
//Adding a new paragraph to the last section.  
IWParagraph paragraph = document.LastSection.AddParagraph()  
//Insert text into the paragraph.  
paragraph.AppendText( "Creating Word 2013 format document!" );  
//Saves the document in Word 2013 format type.  
doc.Save("Sample.docx", FormatType.Word2013);
```

[VB.NET]

```
Dim doc As New WordDocument()  
'Opens the Word 2013 format document.  
document.Open(filename, FormatType.Automatic);  
'Adding a new paragraph to the last section.  
Dim paragraph As IWParagraph = document.LastSection.AddParagraph()  
'Insert text into the paragraph.  
paragraph.AppendText("Creating Word 2013 format document!")  
'Saves the document in Word 2013 format type.  
doc.Save("Sample.docx", FormatType.Word2013)
```

DocIO also has the ability to save .doc files into the .docx format (i.e, Microsoft Word 2007, Microsoft Word 2010, and Microsoft Word 2013 format files). All the elements supported by .doc are supported in .docx.

[C#]

```
WordDocument doc = new WordDocument();
doc.Open("SourceDocument.doc", FormatType.Doc);
doc.Save("OutDocument.docx", FormatType.Word2007);
```

[VB .NET]

```
Dim doc As New WordDocument()
doc.Open("SourceDocument.doc", FormatType.Doc)
doc.Save("OutDocument.docx", FormatType.Word2007)
```

Essential DocIO provides support for reading Word 2007/Word 2010/Word 2013 files. Currently we have only a limited support. Docx files can be read with the same API as that of .doc files. The only difference is the format it is opened in.

The following code illustrates how to read a .docx file.

[C#]

```
WordDocument doc = new WordDocument("C:\\sample.docx", FormatType.Docx);
```

[VB]

```
Dim doc As New WordDocument("c:\\sample.docx", FormatType.Docx)
```

4.2.3.1 Word 2010 Support

Microsoft Word 2010 creates the document in ISO/IEC 29500 standard, whereas the older version (MS Word 2007) generates the document in ECMA-376 standard. ISO/IEC 29500 has two types of schema, Transitional and Strict. Office 2010 generates the word document of type ISO/IEC 29500 transitional schema. This feature enables Essential DocIO to create and modify the word document of ISO/IEC standard. Using Essential DocIO, you can create a document with ISO/IEC 29500 standard by specifying its format type as **Word 2010**.

Use Case Scenarios

Essential DocIO can be used to read and write Microsoft Word document of Word 97-2003, Word 2007 and Word 2010 format. It is compatible with both 32 and 64-bit machines, and has no external dependencies.

Application

Essential DocIO provides a rich set of APIs to create and manipulate the word document and its elements. Word 2010 format documents can be created using the same APIs that was used for Word 97-2003 and Word 2007 format documents, except the format type.

To open/save the word 2010 format document, you have to specify the format type as “Automatic” or “Word2010”.

4.2.4 Stream Support

Essential DocIO provides support to open or save the created Word document into the **Memory Stream** and **File Stream**. Using this, you can open or save the document in the database or make changes to the document without saving to disk.

[C#]

```
//Create a Memory Stream.  
MemoryStream memStream = new MemoryStream();  
  
//Save the document into memory stream.  
document.Save ( memStream );  
  
//Move the pointer to the first position  
memStream.Seek ( 0 , SeekOrigin.Begin );  
  
//Create a file Stream  
FileStream fs = new FileStream("Sample.doc", FileMode.Create);  
memStream.WriteTo(fs);  
  
//Open the Word document from stream  
WordDocument sourceDoc = new WordDocument(memStream, FormatType.Doc);  
  
//Close the Streams.  
memStream.Close();  
  
fs.Close();
```

[VB]

```
'Create a Memory Stream.
Dim memStream As New MemoryStream()

'Save the document into memory stream.
document.Save(memStream)

'Move the pointer to the first position
memStream.Seek(0, SeekOrigin.Begin)

'Create a file Stream
Dim fs As New FileStream("Sample.doc", FileMode.Create)
memStream.WriteTo(fs)

'Open the Word document from stream
Dim sourceDoc As New WordDocument(memStream, FormatType.Doc)

'Close the Streams.
memStream.Close()

fs.Close()
```

4.2.5 Macro-enabled Document Support

A macro is a piece of Visual Basic (VB) programming code that is embedded in a word file to automate repetitive tasks. Essential DocIO provides support for manipulating Microsoft Word macro-enabled documents (*.docm) and Microsoft Word macro-enabled templates (*.dotm) of Word 2007 and Word 2010 formats.

Microsoft Word macro-enabled format files can be created with the following format type enumerations while saving the WordDocument object:

- Word2007Docm - Microsoft Word 2007 macro enabled document format.
- Word2007Dotm - Microsoft Word 2007 macro enabled template format.
- Word2010Docm - Microsoft Word 2010 macro enabled document format.
- Word2010Dotm - Microsoft Word 2010 macro enabled template format.
- Word2013Docm - Microsoft Word 2013 macro enabled document format.
- Word2013Dotm - Microsoft Word 2013 macro enabled template format.

 **Note:** The macros present in the input macro-enabled document will be preserved as it is in the output macro-enabled document. However, Essential DocIO does not have support to create or edit macro commands in the macro-enabled documents.

Use Case Scenario

Using this feature you can preserve the macros present in the input macro-enabled document during conversion to another macro-enabled document (*.docm to *.docm, *.dotm to *.dotm, *.docm to *.dotm, and vice versa).

The following code illustrates how to open and save a Word macro-enabled document.

[C#]

```
WordDocument doc = new WordDocument();
//Opens the Word macro-enabled document.

doc.Open("SourceDocument.docm", FormatType.Automatic);
//Saves as the Word macro-enabled document.

doc.Save("OutDocument.docm", FormatType.Word2007Docm);
```

[VB.NET]

```
Dim doc As New WordDocument()
'Opens the Word macro-enabled document.

doc.Open("SourceDocument.docm", FormatType.Automatic)
'Saves as the Word macro-enabled document.

doc.Save("OutDocument.docm", FormatType.Word2007Docm)
```

The following code illustrates how to open a Word macro-enabled document and save it as macro free document.

[C#]

```
WordDocument doc = new WordDocument();
//Opens the Word macro-enabled document.

doc.Open("SourceDocument.docm", FormatType.Automatic);
//Determines whether the document has Macros.

if (doc.HasMacros)
{
    //Removes the macro commands present in the macro-enabled document.
    doc.RemoveMacros();
}

//Saves as the macro free Word document.

doc.Save("OutDocument.docx", FormatType.Word2007);
```

[VB.NET]

```

Dim doc As New WordDocument()
'Opens the Word macro-enabled document.
doc.Open("SourceDocument.docm", FormatType.Automatic)
'Determines whether the document has Macros.

If doc.HasMacros Then
    'Removes the macro commands present in the macro-enabled document.
    doc.RemoveMacros()

End If
'Saves as the macro free Word document.

doc.Save("OutDocument.docx", FormatType.Word2007)

```



Note: Now macros will not be cloned/imported to the destination macro-enabled document while cloning/importing a macro-enabled document. If the macros are present in the destination macro-enabled document, then it will be preserved as it is in the macro-enabled document.

Samples Link

To view samples:

1. Open the Syncfusion's Dashboard.
2. Select **Reporting** Edition, and then click **ASP.NET**.



- Note:** You can select required platform under Reporting Edition.
3. Click **Run Samples**, and then click **DocIO** at the bottom.
 4. Navigate to **View > Macro Preservation**.

4.2.6 Track Changes

Track Changes feature enables you to keep track of the changes made to a document in Microsoft Word. Using this feature, you can maintain a record of every insertion, deletion and modification in the document, including who made the change and when it was made. The objects that carry such information are called "Tracking Changes" or "Revisions".

Revision Tracking is normally meant for use in a shared environment, so you can track how other people may have changed a document for which you are responsible. However, it can also be a valuable tool even if you are the only one using a document, as you can view your own changes in the document.

Accessing Revisions

You can choose to accept or reject the changes made to a document in Microsoft Word. ParagraphItem and TextBodyItem (WParagraph and WTable are TextBodyItems) objects have the **WordDocument.AcceptChanges** and **WordDocument.RejectChanges** properties, to detect whether an object was inserted or deleted in Microsoft Word (revision tracking being enabled).

WordDocument.HasChanges specifies whether the document has any revisions (changes). It returns **True**, if the document has atleast one revision.

WordDocument.TrackChanges property is used to enable revision tracking in Microsoft Word. Note that this setting does not affect the changes made to the document by using DocIO. Also, the changes made to the document by using DocIO, are never tracked as revisions.

[C#]

```
Syncfusion.DocIO.DLS.WordDocument doc = new WordDocument(@"../../Essential
DocIO.doc", FormatType.Doc);
foreach (WSection section in doc.Sections)
{
    for (int i = 0; i <= section.Paragraphs.Count - 1; i++)
    {
        para = section.Paragraphs[i];

        // Check each paragraph items revisions.
        foreach (ParagraphItem item in para.Items)
        {
            Console.WriteLine(item.EntityType.ToString() + " Inserted: " +
item.IsInsertRevision.ToString());
            Console.WriteLine(item.EntityType.ToString() + " Deleted: " +
item.IsDeleteRevision.ToString());
        }
    }
}

// Accept tracking changes of the document.
doc.AcceptChanges();
doc.Save("sample.doc");
```

[VB]

```

Dim doc As Syncfusion.DocIO.DLS.WordDocument = New
WordDocument("../Essential_DocIO.doc", FormatType.Doc)
For Each section As WSection In doc.Sections
    For i As Integer = 0 To section.Paragraphs.Count - 1
        para = section.Paragraphs(i)

        ' Check each paragraph items revisions.
        For Each item As ParagraphItem In para.Items
            Console.WriteLine((item.EntityType.ToString() & " Inserted: ") +
                item.IsInsertRevision.ToString())
            Console.WriteLine((item.EntityType.ToString() & "Deleted:") +
                item.IsDeleteRevision.ToString())
        Next
    Next
Next

' Accept tracking changes of the document.
doc.AcceptChanges()
doc.Save("sample.doc")

```

4.2.7 Font Substitution for Word Documents

Font substitution is the process of using an alternate font for fonts that are not installed in the system. MS Word renders the text based on alternate font defined, if the actual font is not installed in the system. MS Word displays the actual font name of the text in the font dialog box, but the text will be rendered based on the alternate font. Below screen shot illustrates the “Arial (W1)” font is substituted by the alternate font “Gabriola” in MS Word document.

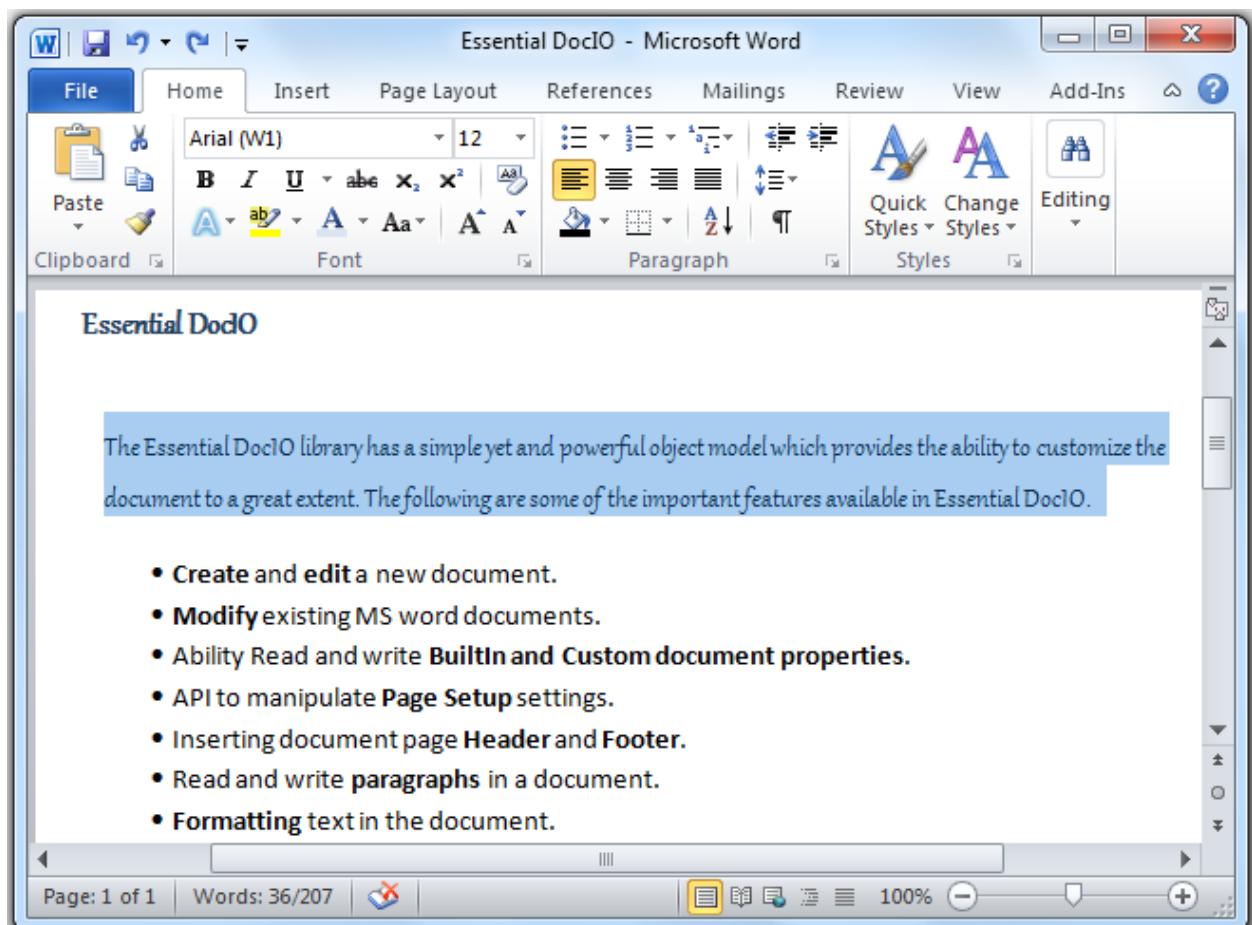


Figure 31: MS Word document with “Gabriola” as alternate font for “Arial (W1)”.

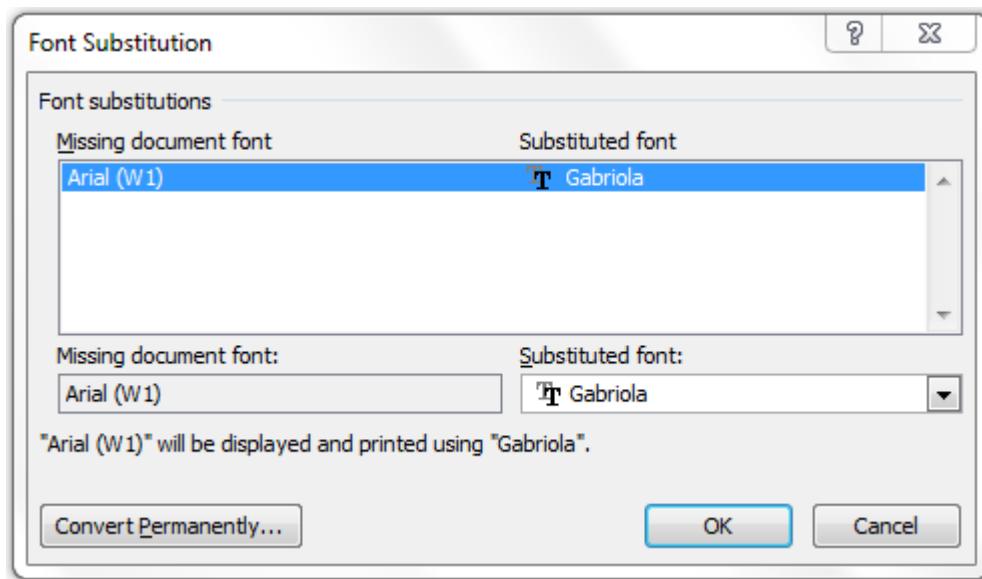


Figure 32: MS Word Font Substitution Table



Note: In some cases MS Word preserves the alternate font based the information stored in the application cache. To overcome this case close all the open instances of MS Word and re-instantiate the document.

The following code snippets illustrate how to access the font substitution table.

[C#]

```
// Updating alternate font name in font substitution table
if (document.FontSubstitutionTable.ContainsKey("Arial (W1)"))
    document.FontSubstitutionTable["Arial (W1)"] = "Gabriola";
else
    document.FontSubstitutionTable.Add("Arial (W1)", "Gabriola");
```

[VB .NET]

```
' Updating alternate font name in font substitution table
If document.FontSubstitutionTable.ContainsKey("Arial (W1)") Then
    document.FontSubstitutionTable("Arial (W1)") = "Gabriola"
Else
    document.FontSubstitutionTable.Add("Arial (W1)", "Gabriola")
End If
```

4.3 Section

WSection class represents a single section in a document. Every section is a region with its own **PageSetup Options, HeadersFooters** and **Paragraphs Collection**.

A valid section must contain at least one empty paragraph. Each section can have its own page setup. Page setup of DocIO section is accessible through the **PageSetup** property. This property enables user to set page size, orientation, margins, and so on.

DocIO section holds four different collections.

- **Paragraphs:** collection of section paragraphs
- **Tables:** collection of section tables
- **ChildEntities:** collection of child entities (general collection which includes paragraphs and tables)
- **Columns:** collection of columns, which logically divides a page on many printing / publishing areas

Each section has its own header and footer which is set by using the **HeadersFooters** property. For more details, see [Headers and Footers](#).

Document sections are divided by section breaks that define where the sections start. This is specified by using the **BreakCode** property.

- **NewColumn**: section starts from a new column
- **NewPage**: section starts from a new page
- **EvenPage**: section starts on a new even page
- **OddPage**: section starts on a new odd page

The following screen shot illustrates the breaks accessible through the **Insert** menu in the MS Word Break dialog box.

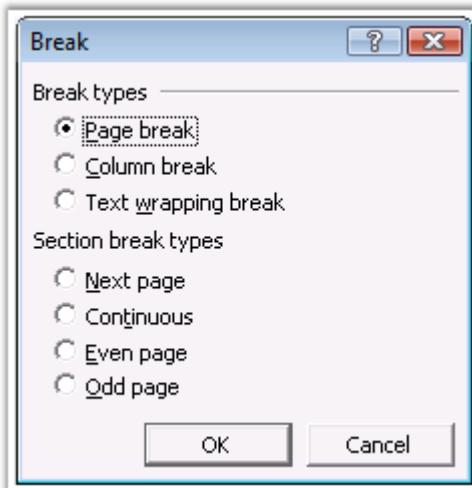


Figure 33: Break Dialog Box

The following screen shot illustrates the various page setup options accessible through the **File** menu in MS Word.

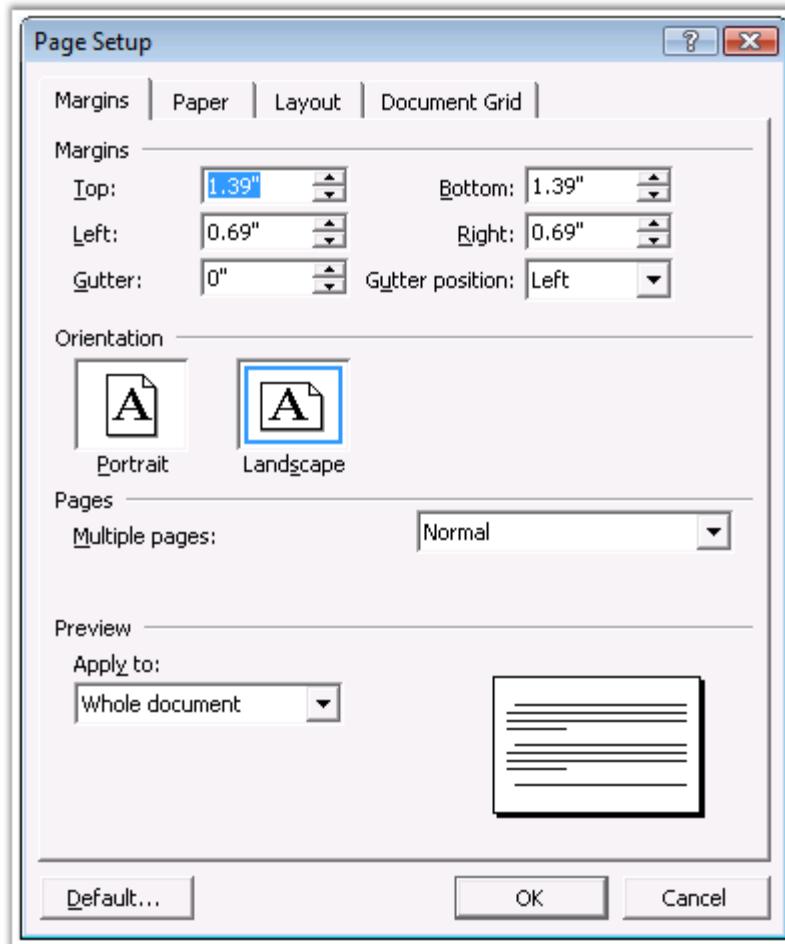


Figure 34: Page Setup Dialog Box

PageSetup properties are listed in the following table.

Name	Description
Bidi	Gets or sets whether section contains right-to-left text.
Borders	Gets page borders collection.
ClientWidth	Gets width of client area.
DefaultTabWidth	Gets or sets the length of the auto tab.
DifferentFirstPage	Setting to specify that the current section has a different header / footer for first page.

DifferentOddAndEvenPages	True if the document has different headers and footers for odd-numbered and even-numbered pages.
FooterDistance	Gets or sets footer height in points.
HeaderDistance	Gets or sets height of header in points.
IsFrontPageBorder	Gets or sets a value indicating whether this instance is front page border.
LineNumberingDistanceFromText	Gets or sets distance from text in lines numbering.
LineNumberingMode	Gets or sets line numbering mode.
LineNumberingStartValue	Gets or sets line numbering start value.
LineNumberingStep	Gets or sets line numbering step.
Margins	Gets or sets page margins in points.
Orientation	Gets or sets orientation of a page.
PageBorderApply	Gets or sets the value that determine on which pages border is applied.
PageBorderOffsetFrom	Gets or sets the position of page border.
PageBordersApplyType	Gets or sets the value that determine on which pages border is applied.
PageSize	Gets or sets page size in points.
VerticalAlignment	Gets or sets vertical alignment.

Public Constructors

Name	Description
WSection.WSection (IWordDocument)	Initializes a new instance of the WSection class.

Public Properties

Name	Description
Body	Gets the section body.

BreakCode	Gets / sets break code.
ChildEntitiesx	Gets the child entities.
Columns	Gets collection of columns, which logically divide page on many printing/publishing areas.
EntityType	Gets the type of the entity.
HeadersFooters	Gets headers/footers of current section.
PageSetup	Gets page Setup of current section.
Paragraphs	Gets the paragraphs.
Tables	Gets the tables.

Public Methods

Name	Description
AddColumn	Adds new column to the section.
AddParagraph	Adds the paragraph.
AddTable	Adds the table.
Clone	Clones itself.
MakeColumnsEqual	Makes all columns in current section to be of equal width.

The following example illustrates how to create a simple Word document, and add sections and breaks to it.

```
[C#]

//Create a new Word document
IWordDocument doc = new WordDocument();

IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();
paragraph.AppendText("Text Body_Text");
```

```
//Set Page break
paragraph.ParagraphFormat.PageBreakAfter = true;

paragraph = section.AddParagraph();
paragraph.AppendText("[ After PAGE BREAK ] \rText Body_Text");

section = doc.AddSection();

//Set Section break
section.BreakCode = SectionBreakCode.NewPage;
paragraph = section.AddParagraph();
paragraph.AppendText("[ After SECTION BREAK ( New page ) ] \rText
Body_Text");

section = doc.AddSection();

//Set page setup Options
section.PageSetup.Borders.BorderType = BorderStyle.DashLargeGap;
section.PageSetup.Borders.Color = Color.DeepPink;
section.PageSetup.PageBorderOffsetFrom = PageBorderOffsetFrom.PageEdge;
section.PageSetup.Borders.LineWidth = 2;
section.BreakCode = SectionBreakCode.NoBreak;
paragraph = section.AddParagraph();
paragraph.AppendText("[ After SECTION BREAK ( continuous page ) ] \rText
Body_Text");
```

[VB.NET]

```
'Create a new Word document
Dim doc As IWordDocument = New WordDocument()

Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph = section.AddParagraph()
paragraph.AppendText("Text Body_Text")

'Set Page break
paragraph.ParagraphFormat.PageBreakAfter = True

paragraph = section.AddParagraph()
paragraph.AppendText("[ After PAGE BREAK ] " & Constants.vbCr & "Text
Body_Text")

section = doc.AddSection()

'Set Section break
section.BreakCode = SectionBreakCode.NewPage
```

```

paragraph = section.AddParagraph()
paragraph.AppendText("[ After SECTION BREAK ( New page ) ] " & Constants.vbCr
& "Text Body_Text")

section = doc.AddSection()

'Set page setup Options
section.PageSetup.Borders.BorderType = BorderStyle.DashLargeGap
section.PageSetup.Borders.Color = Color.DeepPink
section.PageSetup.PageBorderOffsetFrom = PageBorderOffsetFrom.PageEdge
section.PageSetup.Borders.LineWidth = 2
section.BreakCode = SectionBreakCode.NoBreak
paragraph = section.AddParagraph()
paragraph.AppendText("[ After SECTION BREAK ( continuous page ) ] " &
Constants.vbCr & "Text Body_Text")

```

For More Information Refer:

[Cloning and Merging](#), [Headers and Footers](#), [Table](#)

4.3.1 Cloning and Merging

DocIO has an ability to clone the whole Word document or a part of it.

Use the **Clone** method for "deep" document cloning. This method returns the new object of the WordDocument class along with the content of the cloned document which invoked the Clone method. You can use the Clone method to clone any document entity.



Note: If source and destination documents have styles with the same names, then the styles of the imported document will be renamed after importing.

The following example illustrates how to merge two documents by using the Clone method.

[C#]

```

// Create the first document.
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();
IWTextRange text1 = section.AddParagraph().AppendText( "First document
section..." );
text1.CharacterFormat.TextColor = Color.Red;
section.AddParagraph().AppendText( "Some Text..." );

```

```
section.AddParagraph().AppendText( "New Paragraph" );
section.AddParagraph().AppendText( "Third Paragraph" );

// Create the second document.
IWordDocument doc2 = new WordDocument();
IWSection section2 = doc2.AddSection();
IWTextRange text2 = section2.AddParagraph().AppendText( "Second document
section..." );
text2.CharacterFormat.TextColor = Color.Blue;
section2.AddParagraph().AppendText( "Some Text..." );
section2.AddParagraph().AppendText( "New Paragraph More Text" );
section2.AddParagraph().AppendText( "Third Paragraph More Text" );

// Merge the second document with the first.
doc.Sections.Add( section2.Clone() );
```

[VB.NET]

```
' Create the first document.
Dim doc As IWordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()
Dim text1 As IWTextRange = section.AddParagraph().AppendText("First document
section...")
text1.CharacterFormat.TextColor = Color.Red
section.AddParagraph().AppendText("Some Text...")
section.AddParagraph().AppendText("New Paragraph")
section.AddParagraph().AppendText("Third Paragraph")

' Create the second document.
Dim doc2 As IWordDocument = New WordDocument()
Dim section2 As IWSection = doc2.AddSection()
Dim text2 As IWTextRange = section2.AddParagraph().AppendText("Second
document section...")
text2.CharacterFormat.TextColor = Color.Blue
section2.AddParagraph().AppendText("Some Text...")
section2.AddParagraph().AppendText("New Paragraph More Text")
section2.AddParagraph().AppendText("Third Paragraph More Text")

' Merge the second document with the first.
doc.Sections.Add(section2.Clone())
```

Import Contents

Import content functionality is used to copy/merge the contents from one document to another. Compatibility options of the source document will not be imported to the destination document.

Use the **ImportContent(WordDocument doc)** method to import contents and styles from the source document to the destination document.

Use the **ImportContent(WordDocument doc, bool importStyles)** method to import contents from the source document to the destination document by specifying whether to import styles which have the same name between the source and destination document.

- If **importStyles** is true, all the contents and styles of the source document will be imported to the destination document. In cases where a style in the source document has the same name as a style in the destination document, “Guid” is added as a suffix to the name of the imported style in order to preserve unique style name.
- If **importStyles** is false, all the contents will be imported, but only the styles that are not present in the destination document will be preserved. In cases where a style with the same name exists in the destination document, the destination style is applied to the imported contents.



Note: If source and destination documents have styles with the same names, then Guid is added as a suffix to the name of the imported styles in the destination document.

Use the **ImportContent(WordDocument doc, ImportOptions importOptions)** method to import contents from the source document to the destination document with various import options similar to MS Word copy and paste options. Below are the import options supported by Essential DocIO.

- **KeepSourceFormatting**—Imports all the contents of the source document to the destination document and preserves the entire source document formatting of the content as direct formatting. Header and footer contents will be imported similar to the **UseDestinationStyles** option.
- **MergeFormatting**—Imports all the contents of the source document to the destination document and applies the formatting of surrounding content in destination document. Merges the formatting of the contents surrounding it by preserving some of the source formatting (such as bold, italic, underline, etc.). Header and footer contents will be imported similar to the **UseDestinationStyles** option.
- **KeepTextOnly**—Imports only the text from the source document to the destination document (tables, textboxes, pictures, headers, footers, etc., will be removed), similar to content copied from a text file (.txt).
- **UseDestinationStyles**—Imports all the content of the source document to the destination document and applies the styles present in the destination document, or imports the source style to the destination document if no style with same name in destination document.

The following example illustrates how to import contents from one document to another with various import options.

[C#]

```
// Open the destination word document.  
WordDocument destination = new WordDocument("Destination.doc");  
  
// Open the source word document.  
WordDocument source = new WordDocument("Source.doc");  
  
// Imports the contents with import option keep source formatting.  
destination.ImportContent(source, ImportOptions.KeepSourceFormatting);  
  
// Save the document.  
destination.Save("Sample.doc", FormatType.Doc);
```

[VB.NET]

```
' Open the destination word document.  
Dim destination As New WordDocument("Destination.doc")  
  
' Open the source word document.  
Dim source As New WordDocument("Source.doc")  
  
' Imports the contents with import option keep source formatting.  
destination.ImportContent(source, ImportOptions.KeepSourceFormatting)  
  
' Save the document.  
destination.Save("Sample.doc", FormatType.Doc)
```

4.3.2 Headers and Footers

Headers and Footers are displayed at the top and bottom of the document pages respectively. Headers and Footers can include text, graphics, and nearly any other information that can be contained by a document.

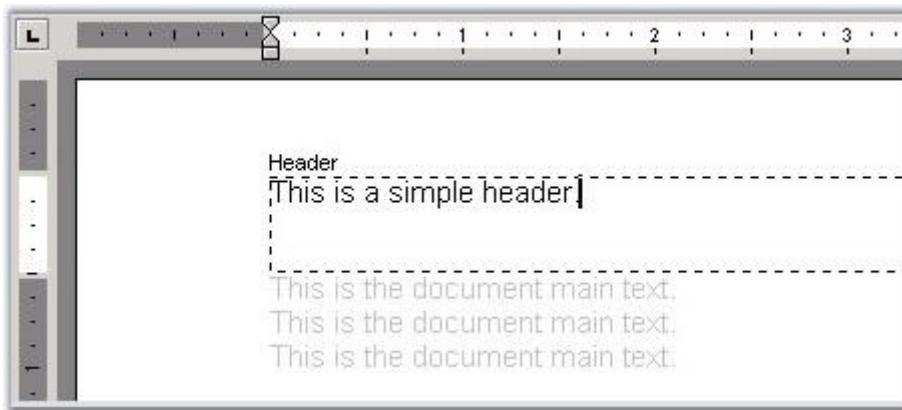


Figure 35: Header added to the Document

Headers and Footers are characteristics (properties) of the document section. Each document section can have its own set of headers/footers. Each section can also have different headers on the first, odd and even pages.

You can set the header and footer by using the **HeadersFooters** property of the Word Document's section. **HeadersFooters** property returns the object of the **WHeadersFooters** type. To access a particular header/footer, you can use the following properties of the **WHeadersFooters** class.

- FirstPageHeader
- FirstPageFooter
- OddHeader
- OddFooter
- EvenHeader
- EvenFooter

The following properties return the object of the **HeaderFooter** type.

Public Properties

Name	Description
EvenFooter	Gets even footer.
EvenHeader	Gets even header.
FirstPageFooter	Gets first page footer.
FirstPageHeader	Gets first page header.
Footer	Gets default footer.

Header	Gets default header.
IsEmpty	Detects whether all headers/footers are empty.
OddFooter	Gets odd footer (This is also the default footer).
OddHeader	Gets odd header (This is also the default header).
LinkToPrevious	Links to previous section's header and footer.

Public Methods

Name	Description
GetEnumerator	Returns an enumerator that iterates through a collection.

HeaderFooter Class

The HeaderFooter class represents the page header or footer. It is inherited from the **WTextBody**, and hence can hold other paragraphs inside.

Public Properties

Name	Description
EntityType	Gets the type of the entity.

The following example illustrates how to add text to different types of headers and footers.

[C#]

```
// A new document is created
WordDocument document = new WordDocument();

// Adding the first section to the document.
IWSection section = document.AddSection();

// Adding a paragraph to the section.
IWParagraph paragraph = section.AddParagraph();

// Setting DifferentFirstPage and DifferentOddEvenPages as true for inserting
```

```

Header and Footer text.
section.PageSetup.DifferentFirstPage = true;
section.PageSetup.DifferentOddAndEvenPages = true;

// Appending some text to the first page in document.
paragraph.AppendText( "\r\r[ First Page ] \r\rText Body_Text Body_Text
Body_Text Body_Text Body_Text Body" );
paragraph.ParagraphFormat.PageBreakAfter = true;

// Appending some text to the second page in document.
paragraph = section.AddParagraph();
paragraph.AppendText( "\r\r[ Second Page ] \r\rText Body_Text Body_Text
Body_Text Body_Text Body_Text Body" );
paragraph.ParagraphFormat.PageBreakAfter = true;

// Appending some text to the third page in document.
paragraph = section.AddParagraph();
paragraph.AppendText( "\r\r[ Third Page ] \r\rText Body_Text Body_Text
Body_Text Body_Text Body_Text Body" );

// Inserting First Page Header
paragraph = new WParagraph( document );
paragraph.AppendText( "[ FIRST PAGE Header ]" );
section.HeadersFooters.FirstPageHeader.Paragraphs.Add( paragraph );

// Inserting First Page Footer
paragraph = new WParagraph( document );
paragraph.AppendText( "[ FIRST PAGE Footer ]" );
section.HeadersFooters.FirstPageFooter.Paragraphs.Add( paragraph );

// Inserting Odd Pages Header
paragraph = new WParagraph( document );
paragraph.AppendText( "[ ODD Page Header Text goes here ]" );
section.HeadersFooters.OddHeader.Paragraphs.Add( paragraph );

// Inserting Odd Pages Footer
paragraph = new WParagraph( document );
paragraph.AppendText( "[ ODD Page Footer Text goes here ]" );
section.HeadersFooters.OddFooter.Paragraphs.Add( paragraph );

// Inserting Even Pages Header
paragraph = new WParagraph( document );
paragraph.AppendText( "[ EVEN Page Header Text goes here ]" );
section.HeadersFooters.EvenHeader.Paragraphs.Add( paragraph );

// Inserting Even Pages Footer

```

```

paragraph = new WParagraph( document );
paragraph.AppendText( "[ EVEN Page Footer Text goes here ]" );
section.HeadersFooters.EvenFooter.Paragraphs.Add( paragraph );

// Adding the second section to the document.
section = document.AddSection();
section.PageSetup.DifferentFirstPage = true;

// Appending some text to the Second Sections's first page in the document.
paragraph = section.AddParagraph();
paragraph.AppendText( "\r\r[ First Page for SECOND SECTION ]\r[ ON DIFFERENT
FIRTS PAGE ]\r\rText Body_Text Body_Text Body_Text Body_Text Body_Text Body"
);
paragraph.ParagraphFormat.PageBreakAfter = true;

// Appending some text to the Second Sections's second page in the document.
paragraph = section.AddParagraph();
paragraph.AppendText( "\r\r[ Second Page for SECOND SECTION ]\rText Body_Text
Body_Text Body_Text Body_Text Body_Text Body" );

// Inserting Second Sections's First Header
paragraph = new WParagraph( document );
paragraph.AppendText( "[ SECOND SECTION FIRST PAGE Header ]" );
section.HeadersFooters.FirstPageHeader.Paragraphs.Add( paragraph );

// Inserting Second Sections's First Footer
paragraph = new WParagraph( document );
paragraph.AppendText( "[ SECOND SECTION FIRST PAGE Footer ]" );
section.HeadersFooters.FirstPageFooter.Paragraphs.Add( paragraph );

// Inserting Second Sections's Header
paragraph = new WParagraph( document );
paragraph.AppendText( "SECOND SECTION Header Text goes here" );
section.HeadersFooters.OddHeader.Paragraphs.Add( paragraph );

// Inserting Second Sections's Footer
paragraph = new WParagraph( document );
paragraph.AppendText( "SECOND SECTION Footer Text goes here" );
section.HeadersFooters.OddFooter.Paragraphs.Add( paragraph );

// Saving the document to disk.
document.Save( "Sample.doc" , FormatType.Doc );

```

[VB.NET]

```
' A new document is created.
```

```

Dim document As WordDocument = New WordDocument()

' Adding the first section to the document.
Dim section As IWSection = document.AddSection()

' Adding a paragraph to the section.
Dim paragraph As IWPParagraph = section.AddParagraph()

' Setting DifferentFirstPage and DifferentOddEvenPages as true for inserting
Header and Footer text.
section.PageSetup.DifferentFirstPage = True
section.PageSetup.DifferentOddAndEvenPages = True

' Appending some text to the first page in document.
paragraph.AppendText(Constants.vbCr + Constants.vbCr & "[ First Page ] " &
Constants.vbCr + Constants.vbCr & "Text Body_Text Body_Text Body_Text
Body_Text Body_Text Body")

paragraph.ParagraphFormat.PageBreakAfter = True

' Appending some text to the second page in document.
paragraph = section.AddParagraph()
paragraph.AppendText(Constants.vbCr + Constants.vbCr & "[ Second Page ] " &
Constants.vbCr + Constants.vbCr & "Text Body_Text Body_Text Body_Text
Body_Text Body_Text Body")

paragraph.ParagraphFormat.PageBreakAfter = True

' Appending some text to the third page in document.
paragraph = section.AddParagraph()
paragraph.AppendText(Constants.vbCr + Constants.vbCr & "[ Third Page ] " &
Constants.vbCr + Constants.vbCr & "Text Body_Text Body_Text Body_Text
Body_Text Body_Text Body")

' Inserting First Page Header
paragraph = New WParagraph(document)
paragraph.AppendText("[ FIRST PAGE Header ]")
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph)

' Inserting First Page Footer
paragraph = New WParagraph(document)
paragraph.AppendText("[ FIRST PAGE Footer ]")
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph)

' Inserting Odd Pages Header
paragraph = New WParagraph(document)
paragraph.AppendText("[ ODD Page Header Text goes here ]")
section.HeadersFooters.OddHeader.Paragraphs.Add(paragraph)

```

```

' Inserting Odd Pages Footer
paragraph = New WParagraph(document)
paragraph.AppendText("[ ODD Page Footer Text goes here ]")
section.HeadersFooters.OddFooter.Paragraphs.Add(paragraph)

' Inserting Even Pages Header
paragraph = New WParagraph(document)
paragraph.AppendText("[ EVEN Page Header Text goes here ]")
section.HeadersFooters.EvenHeader.Paragraphs.Add(paragraph)

' Inserting Even Pages Footer
paragraph = New WParagraph(document)
paragraph.AppendText("[ EVEN Page Footer Text goes here ]")
section.HeadersFooters.EvenFooter.Paragraphs.Add(paragraph)

' Adding the second section to the document.
section = document.AddSection()
section.PageSetup.DifferentFirstPage = True

' Appending some text to the Second Sections's first page in the document.
paragraph = section.AddParagraph()
paragraph.AppendText(Constants.vbCr + Constants.vbCr & "[ First Page for
SECOND SECTION ]" & Constants.vbCr & "[ ON DIFFERENT FIRST PAGE ]" &
Constants.vbCr + Constants.vbCr & "Text Body_Text Body_Text Body_Text
Body_Text Body_Text Body")
paragraph.ParagraphFormat.PageBreakAfter = True

' Appending some text to the Second Sections's second page in the document.
paragraph = section.AddParagraph()
paragraph.AppendText(Constants.vbCr + Constants.vbCr & "[ Second Page for
SECOND SECTION ]" & Constants.vbCr & "Text Body_Text Body_Text Body_Text
Body_Text Body_Text Body")

' Inserting Second Sections's First Header
paragraph = New WParagraph(document)
paragraph.AppendText("[ SECOND SECTION FIRST PAGE Header ]")
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph)

' Inserting Second Sections's First Footer
paragraph = New WParagraph(document)
paragraph.AppendText("[ SECOND SECTION FIRST PAGE Footer ]")
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph)

' Inserting Second Sections's Header
paragraph = New WParagraph(document)
paragraph.AppendText("SECOND SECTION Header Text goes here")
section.HeadersFooters.OddHeader.Paragraphs.Add(paragraph)

```

```
' Inserting Second Section's Footer
paragraph = New WParagraph(document)
paragraph.AppendText("SECOND SECTION Footer Text goes here")
section.HeadersFooters.OddFooter.Paragraphs.Add(paragraph)

' Saving the document to disk.
document.Save("Sample.doc", FormatType.Doc)
```

DocIO provides options to link the header or footer of a section to the corresponding header or footer in the previous section by using the **LinkToPrevious** property. This option is available in the Header/Footer toolbar in MS Word. By default this property is set to **True**.

The following code illustrates how to enable this option by using DocIO.

[C#]

```
doc.AddSection().HeadersFooters.LinkToPrevious = true;
```

[VB .NET]

```
doc.AddSection().HeadersFooters.LinkToPrevious = True
```

Page Number Format

You can insert page numbers of different formats such as arabic numbers, roman numbers, and so on, to the pages in the document. It is also possible to restart the page numbers from any section, and change the starting number of the page number for each section. This is equivalent to the Insert -> Page Numbers -> Format option of MS Word.

[C#]

```
section.PageSetup.PageStartingNumber = 3;
section.PageSetup.RestartPageNumbering = true;
sections.PageSetup.PageNumberStyle = PageNumberStyle.Arabic;
```

[VB .NET]

```
section.PageSetup.PageStartingNumber = 3
section.PageSetup.RestartPageNumbering = True
sections.PageSetup.PageNumberStyle = PageNumberStyle.Arabic
```

The following screen shot illustrates the options provided by DocIO for setting the Page Number.

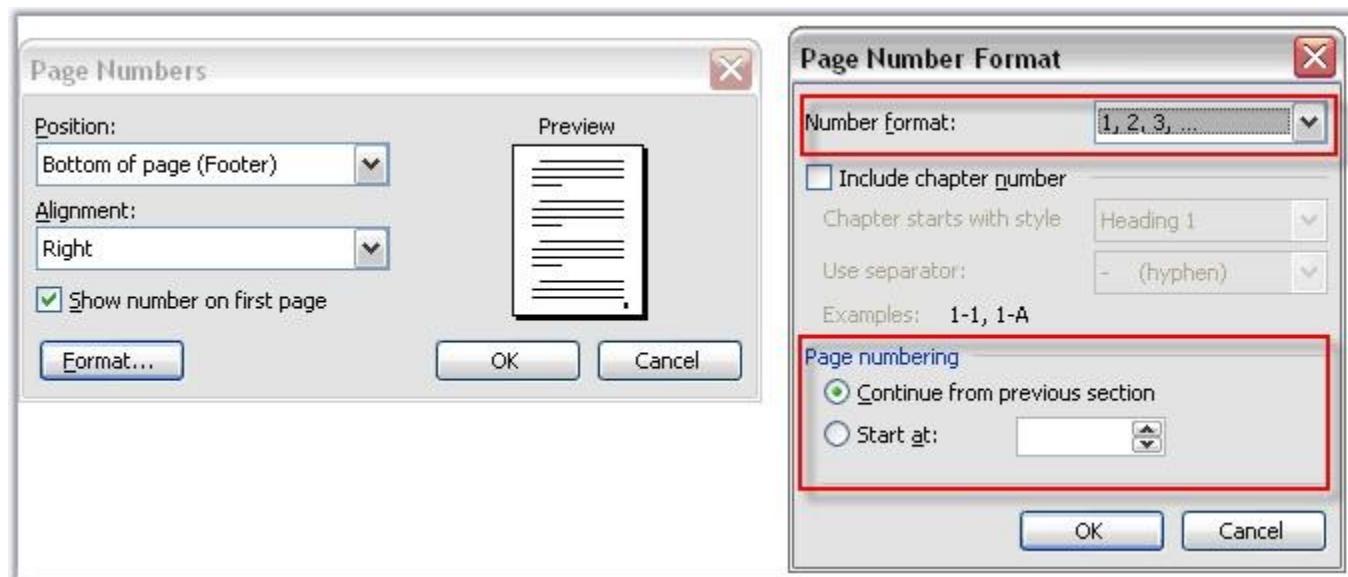


Figure 36: Page Number Settings

4.3.3 Table

WTable class represents a table in a Word document. Every table in the Word document consists of table rows (one or more). Every table row consists of table cells (one or more).

WTable class has similar architecture. It contains a collection of table rows. This collection is accessible through the **Rows** property, which returns the object of the WRowCollection type. Collection of rows consists of **WTableRow** objects. For more details about WTableRow, refer the WTableRow documentation. This class also contains the **TableFormat** property which defines formatting for the whole table. This property returns the object of the RowFormat type. For more details about RowFormat, see [RowFormat](#).

- **ResetCells:** enables you to create tables with the specified number of rows and specified number of cells in each row.
- **WTableRow AddRow(bool isCopyFormat):** enables you to add a new row to the existing table. The **isCopyFormat** parameter defines whether to copy the row format from the previous row.
- **WTableRow AddRow(bool isCopyFormat, bool autoPopulateCells):** enables the user to add a new row to the existing table, but the second parameter, **autoPopulateCells**,

defines whether to create the same number of cells as in the previous row of the table and copy its formatting.

Class Hierarchy

```
TextBodyItem
|
WTable
```

Public Constructors

Name	Description
WTable.WTable (IWordDocument)	Initializes a new instance of the WTable class.
WTable.WTable (IWordDocument, bool)	Initializes a new instance of the WTable class.

Public Properties

Name	Description
ChildEntities	Gets the child entities.
EntityType	Gets the type of the entity.
FirstRow	Get first row of the table.
LastCell	Get last cell of the table.
LastRow	Get last row of the table.
Rows	Get the table rows.
TableFormat	Sets table formatting after ResetCells call.
Width	Gets the width of the table (in points).
StyleName	Gets the table style name.
Title	Gets or sets the table title (Microsoft Word 2010 specific property)
Description	Gets or sets the table description (Microsoft Word 2010 specific property)

Public Methods

Name	Description
AddRow	Adds new row to the table.
Clone	Clones this instance.
Find	Finds text by specified pattern.
Replace	Replaces the text by specified pattern.
ResetCells	Resets rows / columns number.
ApplyStyle	Applies built-in table style to the table.

The following example illustrates how to create an empty table with two rows. Each row has two cells (two columns).

[C#]

```
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();

paragraph.AppendText("Tiny table");
paragraph = section.AddParagraph();
IWTable table = section.AddTable();
table.ResetCells(2, 2);

doc.Save("Table.doc");
```

[VB .NET]

```
Dim doc As IWordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph = section.AddParagraph()

paragraph.AppendText("Tiny table")
paragraph = section.AddParagraph()
Dim table As IWTable = section.AddTable()
table.ResetCells(2, 2)

doc.Save("Table.doc")
```

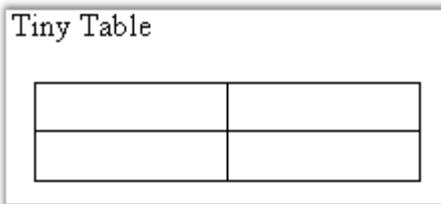


Figure 37: Table with Two Rows and two Columns

Nested Table

You can create nested tables by creating a table in the cell of the parent table by using DocIO. The following code illustrates this.

[C#]

```
// Adding a nested Table to the cell (2,2)of the parent table.
IWTable nestTable = table[2, 2].AddTable();
nestTable.ResetCells(3, 3);
```

[VB .NET]

```
' Adding a new Table to the cell (2,2) of the parent table.
Dim nestTable as IWTable = table[2, 2].AddTable()
nestTable.ResetCells(3, 3)
```

Cell Image

You can also insert images in the table cells by appending a picture to the cell paragraph. The following code illustrates how to insert a picture in the first cell.

[C#]

```
IWTable table = section.Body.AddTable();
table.ResetCells(1, 1);

WTableRow row = table.Rows[0];
paragraph = (IWParagraph)row.Cells[0].AddParagraph();
paragraph.AppendPicture(new Bitmap("image.png"));
```

[VB .NET]

```
Dim table As IWTable = section.Body.AddTable()
table.ResetCells(1, 1)

Dim row As WTableRow = table.Rows(0)
paragraph = DirectCast(row.Cells(0).AddParagraph(), IWParagraph)
paragraph.AppendPicture(New Bitmap("image.png"))
```

4.3.3.1 Table Row

WTableRow class represents a table row in the Word document. WTableRow class has a collection of table cells (WTableCell objects). Formatting for a table row is defined by the **RowFormat** property. This property returns the object of the RowFormat type.

Table Row Height

Table row height is of two types.

- AtLeast
- Exactly

The height type for the table row is defined by the **HeightType** property. When the height type is set to AtLeast, the height of the table row is just enough to fit the text. When the height type is set to Exactly, the row height is specified by the **Height** property, in terms of points.

The following screen shot illustrates how to set the above properties by using the Table Properties dialog box in MS Word.

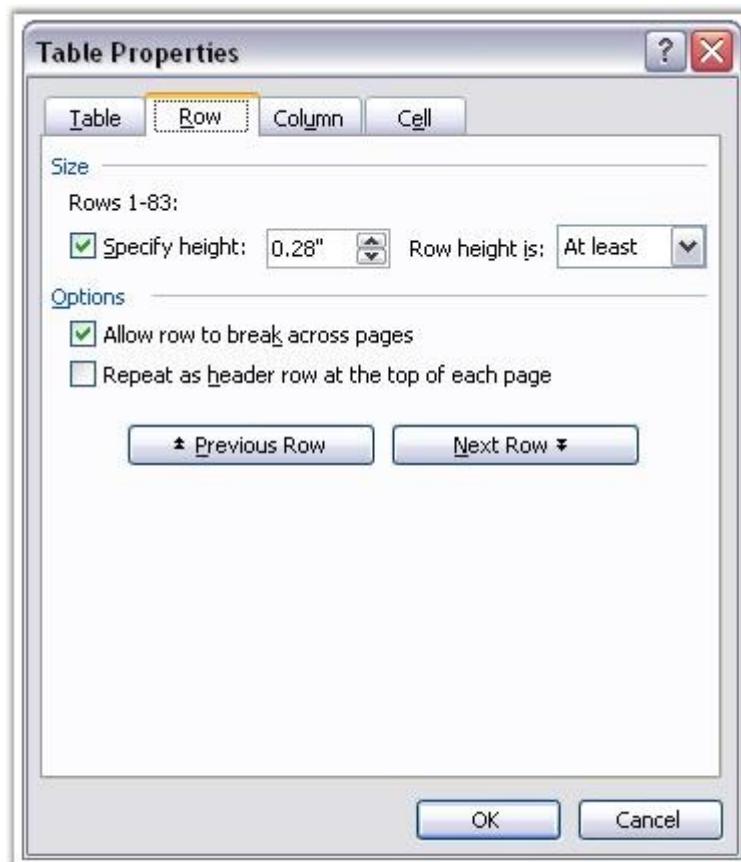


Figure 38: Table Properties Dialog Box

Adding Cells

You can use the **AddCell** and **AddCell(bool isCopyFormat)** functions to add new cells to the table row, where **isCopyFormat** parameter defines whether to apply the formatting of the upper cell to the added cell.

AddCell function without parameters is the same as **AddCell(true)**.

Public Constructors

Name	Description
WTableRow.WTableRow (IWordDocument)	Initializes a new instance of the WTableRow class.

Public Properties

Name	Description
Cells	Gets or sets cell collection.
ChildEntities	Gets the child entities.
EntityType	Gets the type of the entity.
Height	Gets or sets the height of the row (in points)
HeightType	Get or set table row height type.
IsHeader	Gets or sets whether the row is a table header.
RowFormat	Gets table format.

Public Methods

Name	Description
AddCell	Adds a cell.
Clone	Clones this instance.
GetRowIndex	Returns index of the row in owner table.
WTableRow	Initializes a new instance of the WTableRow class.

4.3.3.2 Table Cell

WTableCell class represents a table cell in the Word document. **WTextBody** is the base class of WTableCell, which means that WTableCell object can hold paragraphs. You can format the table cell by using the **CellFormat** property. This property returns the value of the CellFormat type.

The following screen shot illustrates how to set the Cell Format in MS Word.

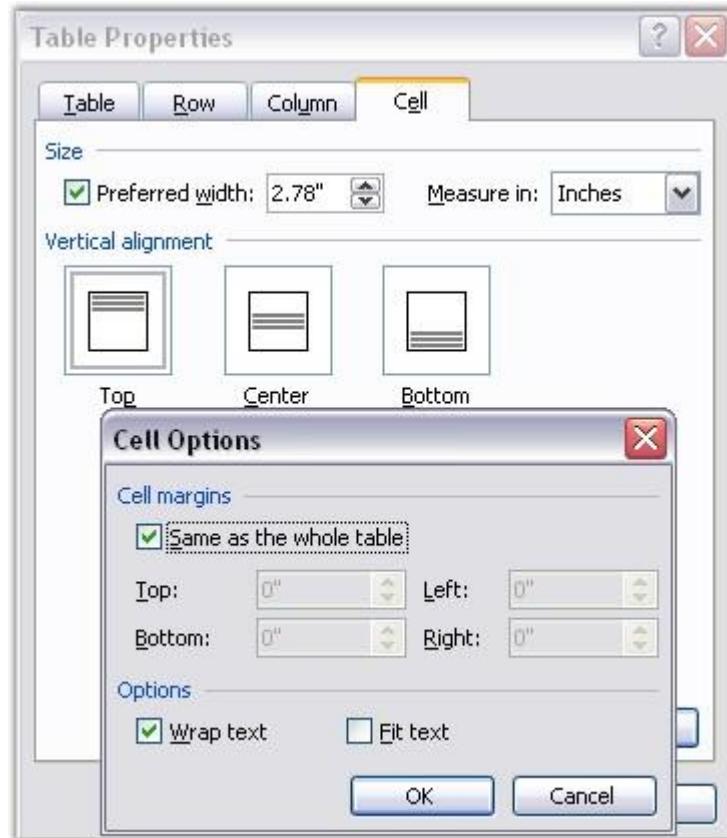


Figure 39: Cell Format Options in Table Properties Dialog Box

Cell Format Public Properties

Name	Description
BackColor	Gets or sets background color.
Borders	Gets borders.
FitText	Gets or sets fit text option.
HorizontalMerge	Gets or sets the way of horizontal merging of the cell.
Paddings	Gets paddings.
TextDirection	Gets or sets cell text direction.
TextWrap	Gets or sets a value indicating whether [text wrap].

VerticalAlignment	Gets or sets vertical alignment.
VerticalMerge	Gets or sets the way of vertical merging of the cell.

Public Constructor

Name	Description
WTableCell.WTableCell (IWordDocument)	Initializes a new instance of the WTableCell class.

Public Properties

Name	Description
CellFormat	Gets cell format.
EntityType	Gets the type of the entity.
OwnerRow	Gets owner row of the cell.
Width	Gets or sets the cell width (in points).

Public Methods

Name	Description
Clone	Clones this instance.
GetCellIndex	Get cell index in the table row.

The following example illustrates how to create a table with non-default formatting.

[C#]

```
paragraph = section.AddParagraph();
paragraph.AppendText("Table with different formatting");
paragraph = section.AddParagraph();

//Add a table
table = section.AddTable();

//Set number of rows and columns
table.ResetCells(3, 3);
table.TableFormat.Borders.LineWidth = 2f;
```

```

table.TableFormat.Borders.Color = Color.Green;

//Select the first row and append text in each cell
WTableRow row0 = table.Rows[0];
row0.Cells[0].AddParagraph().AppendText("1");
row0.Cells[0].CellFormat.Borders.LineWidth = 2f;
row0.Cells[0].CellFormat.Borders.Color = Color.Magenta;

WTableRow row = table.Rows[0];
row0.Cells[1].AddParagraph().AppendText("2");
row0.Cells[2].AddParagraph().AppendText("3");

WTableRow row1 = table.Rows[1];
row = table.Rows[1];
row1.Cells[0].AddParagraph().AppendText("4");
row1.Cells[1].AddParagraph().AppendText("5");
row1.Cells[1].CellFormat.Borders.LineWidth = 2f;
row1.Cells[1].CellFormat.Borders.Color = Color.Brown;

row1.Cells[2].AddParagraph().AppendText("6");
WTableRow row2 = table.Rows[2];
row2.Cells[0].AddParagraph().AppendText("7");
row2.Cells[1].AddParagraph().AppendText("8");
row2.Cells[2].AddParagraph().AppendText("9");
row2.Cells[2].CellFormat.Borders.LineWidth = 2f;
row2.Cells[2].CellFormat.Borders.Color = Color.Cyan;
row2.Cells[2].CellFormat.Borders.Shadow = true;

```

[VB.NET]

```

paragraph = section.AddParagraph()
paragraph.AppendText("Table with different formatting")
paragraph = section.AddParagraph()

'Add a table
table = section.AddTable()

'Set number of rows and columns
table.ResetCells(3, 3)
table.TableFormat.Borders.LineWidth = 2f
table.TableFormat.Borders.Color = Color.Green

'Select the first row and append text in each cell
Dim row0 As WTableRow = table.Rows(0)
row0.Cells(0).AddParagraph().AppendText("1")

```

```

row0.Cells(0).CellFormat.Borders.LineWidth = 2f
row0.Cells(0).CellFormat.Borders.Color = Color.Magenta

Dim row As WTableRow = table.Rows(0)
row0.Cells(1).AddParagraph().AppendText("2")
row0.Cells(2).AddParagraph().AppendText("3")

Dim row1 As WTableRow = table.Rows(1)
row = table.Rows(1)
row1.Cells(0).AddParagraph().AppendText("4")
row1.Cells(1).AddParagraph().AppendText("5")
row1.Cells(1).CellFormat.Borders.LineWidth = 2f
row1.Cells(1).CellFormat.Borders.Color = Color.Brown

row1.Cells(2).AddParagraph().AppendText("6")
Dim row2 As WTableRow = table.Rows(2)
row2.Cells(0).AddParagraph().AppendText("7")
row2.Cells(1).AddParagraph().AppendText("8")
row2.Cells(2).AddParagraph().AppendText("9")
row2.Cells(2).CellFormat.Borders.LineWidth = 2f
row2.Cells(2).CellFormat.Borders.Color = Color.Cyan
row2.Cells(2).CellFormat.Borders.Shadow = True

```

Cell Content Formatting

The **AddParagraph** method of the **WTableCell** class is used to add paragraphs to the table cell. The **TextRange** property of this method is used to format the contents of the cell. For details, see [Text Range](#).

[C#]

```

// Adding a new Table to the text body.
IWTable table = sec.body.AddTable();

// Inserting rows to the table. This will apply the format to whole table.
table.ResetCells(6, 6, format, 80);
WTableCell cell = table.Rows[0].Cells[0] as WTableCell;
WTextRange range = cell.AddParagraph().AppendText("aaaaaaaaaaaaaaaaaa") as
WTextRange ;

//Format first cell first paragraph.
range.CharacterFormat.FontName = "Arial";
range.CharacterFormat.FontSize = 10;

```

```
// Format first cell second paragraph.
range = cell.AddParagraph().AppendText("bbbbbbbbbbbbbbbb") as WTextRange;
range.CharacterFormat.Italic = true;
range.CharacterFormat.FontSize = 12;
```

[C#]

```
' Adding a new Table to the text body.
Dim table As IWTable = sec.body.AddTable()

' Inserting rows to the table. This will apply the format to whole table.
table.ResetCells(6, 6, Format, 80)
Dim cell As WTableCell = TryCast(table.Rows(0).Cells(0), WTableCell)
Dim range As WTextRange =
TryCast(cell.AddParagraph().AppendText("aaaaaaaaaaaaaaaaaa"), WTextRange)

'Format first cell first paragraph.
range.CharacterFormat.FontName = "Arial"
range.CharacterFormat.FontSize = 10

' Format first cell second paragraph.
range = TryCast(cell.AddParagraph().AppendText("bbbbbbbbbbbbbbbb"), WTextRange)
range.CharacterFormat.Italic = True
range.CharacterFormat.FontSize = 12
```

4.3.3.3 Row Format

RowFormat class represents **table or table row** formatting in the Word document.

Properties

- **Borders:** defines format of row borders (width of line, line color and so on)
- **Paddings:** defines margins for the cells in the row or table
- **CellSpacing:** defines spacing between cells
- **LeftIndent:** defines left indent of table or table row
- **IsAutoResized:** defines if table or row auto resizes to fit the text

The following screen shot illustrates how to set the Row Format in MS Word.

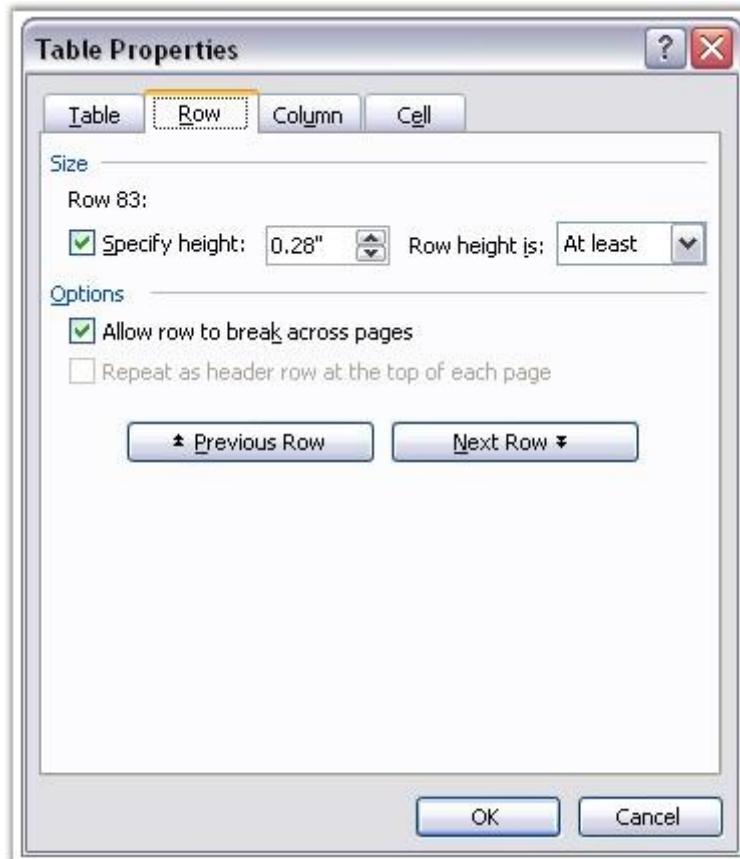


Figure 40: Row Format Options in Table Properties Dialog Box

Public Properties

Name	Description
Bidi	Gets or sets whether table is right-to-left.
Borders	Gets borders.
CellSpacing	Gets or sets spacing between cells (in points).
HorizontalAlignment	Gets or sets horizontal alignment for the paragraph.
IsAutoResized	Gets or sets the boolean value indicating if the table is auto resized.
IsBreakAcrossPages	Gets or sets the boolean value indicating if there is

	a break across pages.
LeftIndent	Gets or sets table indent (in points).
Paddings	Gets paddings.

[C#]

```
// Adding a new Table to the textbody.
IWTable table = sec.body.AddTable();

RowFormat format = new RowFormat();
format.Paddings.All = 5;
format.Borders.BorderType = Syncfusion.DocIO.DLS.BorderStyle.Dot;
format.Borders.LineWidth = 2;

// Inserting rows to the table.This will apply the format to whole table
table.ResetCells(6, 6, format, 80);
```

[VB.NET]

```
' Adding a new Table to the textbody.
Dim table As IWTable = sec.body.AddTable()

Dim format As New RowFormat()
format.Paddings.All = 5
format.Borders.BorderType = Syncfusion.DocIO.DLS.BorderStyle.Dot
format.Borders.LineWidth = 2

' Inserting rows to the table.This will apply the format to whole table
table.ResetCells(6, 6, format, 80)
```

4.3.3.4 Table Styles

WTableStyle class represents table style in the Word document. Table style defines a set of formatting characteristics that can be applied to the table.



Figure 41: Table Styles

You can apply the Word built-in table styles by using the **WTable.ApplyStyle** method with **Built-inTableStyle** enumeration parameter which specifies the built-in table style.

 **Note:** Essential DocIO currently provides support for built-in table styles in Word2007, Word2010, and Word2013 formats.

4.3.4 Content Control

The Content control helps you to design Word documents (.docx) and templates (.dotx) to have the following features:

- **User Interface**—Allows end users to enter data only in a controlled section of the Word document.
- **Data Protection**—Restricts users from editing the protected sections of a Word document or template.
- **Data Binding**—Binds the Content control to a data source.

Limited support has been added to the Essential DocIO to preserve the Content control with custom XML mapping (data binding).

The following are the list of Content controls for which the preservation support is added:

- Rich Text Content control
- Plain Text Content control
- Picture Content control
- Combo Box Content control
- Drop-down List Content control
- Date Picker Content control
- Check Box Content control (Word 2010 only)

 **Note:** Currently Essential DocIO does not provide support to edit/modify the Content controls. Track changes information will not be preserved using DocIO.

4.4 Paragraph

WParagraph class represents a single paragraph in a document. DocIO paragraph contains paragraph items inside. You can add paragraph items by using the Items property. This property returns the collection of paragraph items (object of ParagraphItemCollection type).

Each paragraph has a paragraph format. The format of the paragraph is set by using the **ParagraphFormat** property. This property is used to define the paragraph border, style of texture, foreground and background color, paragraph spacing, and so on. For more details on Paragraph Formatting, see [WParagraphFormat](#).

IsInCell: defines whether current paragraph belongs to the table cell (is in the table cell)

- **IsEndOfSection:** defines whether the current paragraph is the last paragraph in the section
- **IsEndOfDocument:** defines whether the current paragraph is the last paragraph in the document

Formatting Break Symbol

BreakCharacterFormat property is used to set the character formatting for the break symbol.

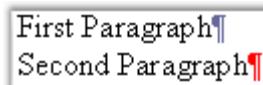


Figure 42: Formatted Break Symbol

DocIO List

DocIO paragraphs can also be displayed as a list by using the ListFormat property. This property returns the object of the WListFormat type. The WListFormat class defines the formatting for the list (applied list style, list level number and so on). For more details on WListFormat class, see [WListFormat](#).

Adding Paragraph Items

DocIO paragraph enables to add paragraph items to the end of the current paragraph by using the Append function. For example, the AppendText method, AppendBreak method, and so on, are used for this purpose.

Class Hierarchy

TextBodyItem

|

WParagraph

Public Constructor

Name	Description
WParagraph.WParagraph (IWordDocument)	Initializes a new instance of the WParagraph class.

Public Properties

Name	Description
BreakCharacterFormat	Gets character format for the break symbol.
ChildEntities	Gets the child entities.
EntityType	Gets the type of the entity.
IsEndOfDocument	Gets a value indicating whether this paragraph is the end of document.
IsEndOfSection	Gets a value indicating whether this paragraph is the end of the section.
IsInCell	Gets a value indicating whether this paragraph is in cell.
Items	Gets paragraph items.
ListFormat	Gets format of the list for the paragraph.
ParagraphFormat	Gets paragraph format.
StyleName	Gets paragraph style name.
Text	Gets or sets paragraph text.

Public Methods

Name	Description
AppendBookmarkEnd	Appends end of the bookmark with specified name into paragraph.
AppendBookmarkStart	Appends start of the bookmark with specified name into paragraph.
AppendBreak	Appends break to end of the paragraph.
AppendCheckBox	Appends checkbox to end of paragraph.
AppendComment	Appends comment to end of paragraph.
AppendDropDownFormField	Appends DropDown form field to end of paragraph.
AppendField	Appends field to end of paragraph.
AppendFootnote	Appends footnote to end of paragraph.
AppendPicture	Appends picture to end of paragraph.
AppendSymbol	Appends special symbol to end of paragraph.
AppendTable	Append Table.
AppendText	Appends text to end of document.
AppendTextBox	Append Text box to the end of the paragraph.
AppendTextFormField	Appends text form field to end of paragraph.

AppendTOC	Appends the TOC.
ApplyStyle	Applies style to the paragraph.
Find	Finds text inside the paragraph.
GetStyle	Gets related style.
Replace	Replaced text inside the paragraph.
InsertSectionBreak	Inserts a section break.

The following example illustrates how to add various formats to paragraphs.

[C#]

```
// Add paragraph and apply formatting.
paragraph = section.AddParagraph();
paragraph.ParagraphFormat.Borders.Bottom.BorderType =
BorderStyle.ThinThickSmallGap;
paragraph.ParagraphFormat.HorizontalAlignment = HorizontalAlignment.Center;
paragraph.ParagraphFormat.BeforeSpacing = 18;
textRange = paragraph.AppendText("Windows Forms. ");

paragraph = section.AddParagraph();
paragraph.ParagraphFormat.PageBreakBefore = true;
paragraph.ParagraphFormat.BackColor = Color.FromArgb(102, 102, 153);
paragraph.ParagraphFormat.BeforeSpacing = 18;
paragraph.ParagraphFormat.AfterSpacing = 6;
paragraph.ParagraphFormat.FirstLineIndent = 45;
```

[VB .NET]

```
' Add paragraph and apply formatting.
paragraph = section.AddParagraph()
paragraph.ParagraphFormat.Borders.Bottom.BorderType =
BorderStyle.ThinThickSmallGap
paragraph.ParagraphFormat.HorizontalAlignment = HorizontalAlignment.Center
paragraph.ParagraphFormat.BeforeSpacing = 18
textRange = paragraph.AppendText("Windows Forms. ")

paragraph = section.AddParagraph()
paragraph.ParagraphFormat.PageBreakBefore = True
paragraph.ParagraphFormat.BackColor = Color.FromArgb(102, 102, 153)
paragraph.ParagraphFormat.BeforeSpacing = 18
paragraph.ParagraphFormat.AfterSpacing = 6
paragraph.ParagraphFormat.FirstLineIndent = 45
```

For More Information Refer:

[Paragraph Items, Styles and Formatting](#)

4.4.1 Paragraph Item

ParagraphItem class is the base class for all paragraph items (pictures, text ranges, comments, watermarks, etc).

Class Hierarchy

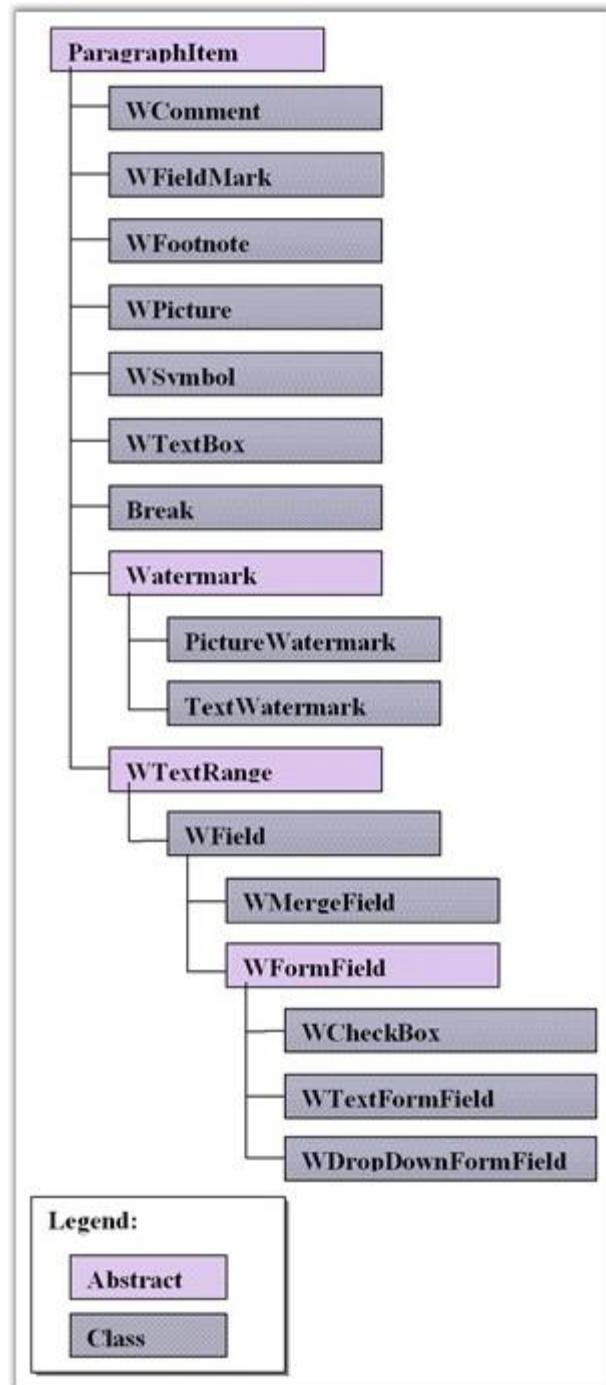


Figure 43: ParagraphItem Class Hierarchy

Public Property

Name	Description
------	-------------

OwnerParagraph	Gets owner paragraph.
----------------	-----------------------

For More Information Refer:

[Text Range](#), [Fields](#), [Bookmark](#), [Shapes](#), [Footnote and Endnote](#), [Symbol](#), [Break](#), [Table Of Contents](#), [OLE Object](#)

4.4.1.1 Text Range

WTextRange class represents a piece of text which has separate formatting. You can format the text by using the CharacterFormat property. For more details on CharacterFormat property, refer the CharacterFormat documentation.

You can also use the AppendText function of the WParagraph class to append text to the paragraph.



Note: *AppendText function appends a new text range to the paragraph with the default character formatting.*

Class Hierarchy**ParagraphItem**

```

  |

```

```
WextRange
```

Public Constructor

Name	Description
WTextRange	Initializes a new instance of the WTextRange class.

Public Properties

Name	Description

CharacterFormat	Gets character format(font properties).
EntityType	Gets the type of the entity.
Text	Gets or sets text.

The following example illustrates how to use the WTextRange class.

[C#]

```
IWordDocument doc = new WordDocument();

// Write different font name/font size.
string[] fontNames = new string[] { "Times New Roman", "Verdana", "Symbol",
};

IWSection section = doc.AddSection();
IWParagraph paragraph;
IWTextRange textRange;

for (int j = 0, len = fontNames.Length; j < len; j++)
{
    paragraph = section.AddParagraph();
    string fontName = fontNames[j];

    for (int i = 9; i < 40; i++)
    {
        textRange = paragraph.AppendText(fontName + " " + i.ToString() + "
");
        textRange.CharacterFormat.FontSize = i;
        textRange.CharacterFormat.FontName = fontName;
        if (i > 15)
        {
            i += 5;
        }
        IWTextRange txtRange2 = paragraph.AppendText(fontName + "34,5 ");
        txtRange2.CharacterFormat.FontSize = (float)34.5;
        txtRange2.CharacterFormat.FontName = fontName;
    }
}

// Write bold/italic/underline/strike text.
paragraph = section.AddParagraph();
textRange = paragraph.AppendText("Bold Text_Bold Text      ");
```

```

textRange.CharacterFormat.Bold = true;
textRange = paragraph.AppendText("Italic Text_Italic Text    ");
textRange.CharacterFormat.Italic = true;
textRange = paragraph.AppendText("Underline Text_Underline Text    ");
textRange.CharacterFormat.UnderlineStyle = UnderlineStyle.Dash;
textRange = paragraph.AppendText("Strike Text_Strike Text    ");
textRange.CharacterFormat.Strikeout = true;

textRange = paragraph.AppendText("Shadow Text_Shadow Text    ");
textRange.CharacterFormat.Shadow = true;

paragraph = section.AddParagraph();
paragraph = section.AddParagraph();

textRange = paragraph.AppendText("Merged Font Style Text_Merged Font Style
Text");
textRange.CharacterFormat.Bold = true;
textRange.CharacterFormat.Italic = true;
textRange.CharacterFormat.Strikeout = true;
textRange.CharacterFormat.UnderlineStyle = UnderlineStyle.Dash;

```

[VB.NET]

```

Dim doc As IWordDocument = New WordDocument()

' Write different font name/font size.
Dim fontNames As String() = New String() { "Times New Roman", "Verdana",
"Symbol", }

Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph
Dim textRange As IWTextRange

Dim j As Integer = 0
len = fontNames.Length

Do While j < len
    paragraph = section.AddParagraph()
    Dim fontName As String = fontNames(j)

    For i As Integer = 9 To 39
        textRange = paragraph.AppendText(fontName & " " & i.ToString() &
" ")
        textRange.CharacterFormat.FontSize = i
        textRange.CharacterFormat.FontName = fontName
        If i > 15 Then

```

```

        i += 5
    End If
Next i
Dim txtRange2 As IWTextRange = paragraph.AppendText(fontName & "34,5 ")
txtRange2.CharacterFormat.FontSize = CSng(34.5)
txtRange2.CharacterFormat.FontName = fontName
j += 1
Loop

// Write bold/italic/underline/strike text.
paragraph = section.AddParagraph()
textRange = paragraph.AppendText("Bold Text_Bold Text    ")
textRange.CharacterFormat.Bold = True
textRange = paragraph.AppendText("Italic Text_Italic Text    ")
textRange.CharacterFormat.Italic = True
textRange = paragraph.AppendText("Underline Text_Underline Text    ")
textRange.CharacterFormat.UnderlineStyle = UnderlineStyle.Dash
textRange = paragraph.AppendText("Strike Text_Strike Text    ")
textRange.CharacterFormat.Strikeout = True

textRange = paragraph.AppendText("Shadow Text_Shadow Text    ")
textRange.CharacterFormat.Shadow = True

paragraph = section.AddParagraph()
paragraph = section.AddParagraph()

textRange = paragraph.AppendText("Merged Font Style Text Merged Font Style
Text")
textRange.CharacterFormat.Bold = True
textRange.CharacterFormat.Italic = True
textRange.CharacterFormat.Strikeout = True
textRange.CharacterFormat.UnderlineStyle = UnderlineStyle.Dash

```

4.4.1.2 Fields

Fields are special elements of the Word document. To insert a field, open the Insert menu and click Field option in Microsoft Word.

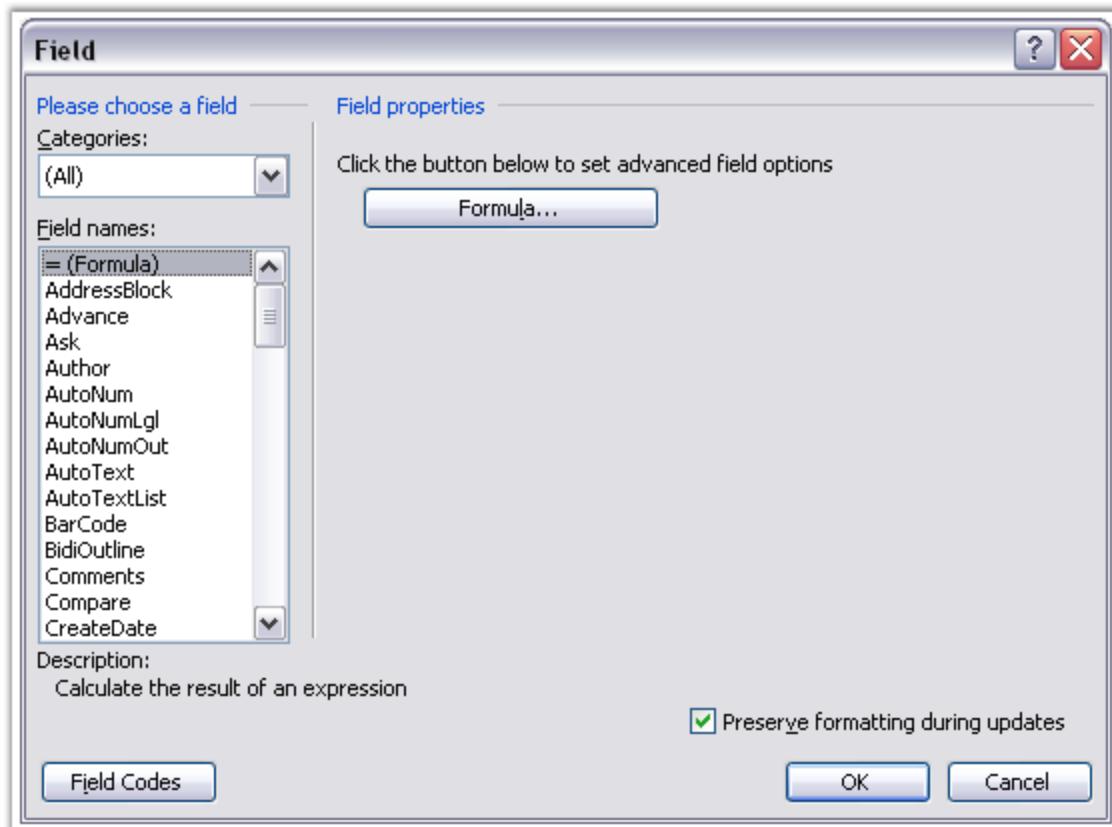
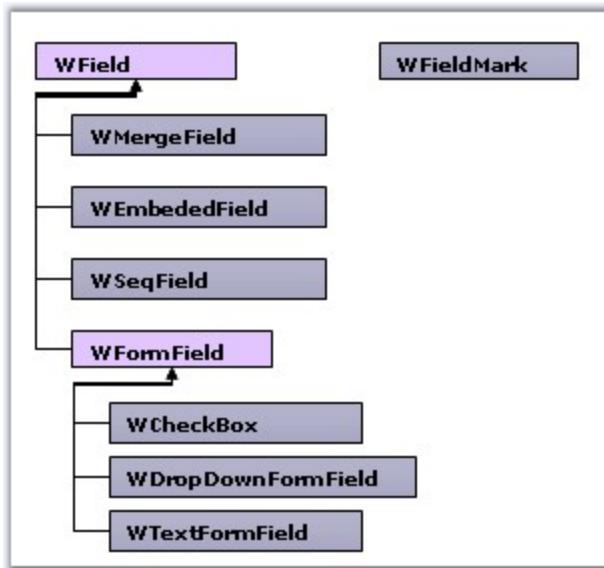


Figure 44: Field Dialog Box

Fields are widely used for Mail Merge. In Word document, almost every field consists of the field start marker text, which defines the type of the field, field separator marker, field value, and field end marker.

WField class represents a field in the Word document. There are many types of fields.

Figure 45: *WField Class Hierarchy*

You can get or set the type of field by using the **FieldType** property. **WField** class also has the **TextFormat** property, which defines the text format for the field. There are four different types of text formats for fields.

- None – no text formatting
- Uppercase – uppercase text formatting
- Lowercase – lowercase text formatting
- FirstCapital – first capital text formatting
- Titlecase – title case text formatting

Adding Field to Paragraph

You can use the **AppendField** function of the **WParagraph** class to add new fields to a paragraph. When you add a field to a paragraph, all the field markers are automatically added to the paragraph. For details, refer **WFieldMark** class description.

There are special fields like Form Field, Merge Field, Embed Field and Seq Field. For details, refer the **WFormField**, **WMergeField**, **WEmbeddedField** and **WSeqField** documentation.

Class Hierarchy

WTextRange

|

WField

Public Constructor

Name	Description
WField.WField (IWordDocument)	Initializes a new instance of the WField class.

Public Properties

Name	Description
EntityType	Gets the type of the entity.
FieldPattern	Gets / sets field pattern.
FieldType	Gets / sets field type
FieldValue	Gets the field value.
TextFormat	Gets/ sets regular text format.

[C#]

```
IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();
paragraph.AppendText("Testing writing Merge Fields into Header");

section.PageSetup.DifferentFirstPage = true;
section.PageSetup.DifferentOddAndEvenPages = true;

paragraph = new WParagraph(doc);
paragraph.AppendText("[ FIRST PAGE Header ]");
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph);
paragraph = new WParagraph(doc);

//Appends field
paragraph.AppendField("Field's Name", FieldType.FieldMergeField);
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph);

paragraph = new WParagraph(doc);
paragraph.AppendText("[ FIRST PAGE Footer ]\r");
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph);
```

[VB.NET]

```

Dim section As IWSection = doc.AddSection()
Dim paragraph As IWPParagraph = section.AddParagraph()
paragraph.AppendText("Testing writing Merge Fields into Header")

section.PageSetup.DifferentFirstPage = True
section.PageSetup.DifferentOddAndEvenPages = True

paragraph = New WParagraph(doc)
paragraph.AppendText("[ FIRST PAGE Header ]")
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph)
paragraph = New WParagraph(doc)

'Appends field
paragraph.AppendField("Field's Name", FieldType.FieldMergeField)
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph)

paragraph = New WParagraph(doc)
paragraph.AppendText("[ FIRST PAGE Footer ]" & Constants.vbCr)
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph)

```

4.4.1.2.1 Merge Field

WMergeField class represents a merge field in the Word document. To add a merge field in Microsoft Word, open the **Insert** menu, click **Field**, and then click **MergeField**. Merge field is suitable for mail merge because it is easy to set the user's data inside it.

- **FieldName**: defines the name of the field
- **TextBefore** and **TextAfter**: define the text that is displayed before and after the merge field
- **NumberFormat** and **DateFormat**: define the number and date format respectively

These properties are used during mail merge. NumberFormat and DateFormat properties do not have an equivalent in Word MergeField.

Class Hierarchy

WTextRange

|

```

WField
|
WMergeField

```

Public Constructor

Name	Description
WMergeField.WMergeField (IWordDocument)	Initializes a new instance of the WMergeField class.

Public Properties

Name	Description
DateFormat	Gets the date format.
EntityType	Gets the type of the entity.
FieldName	Gets or sets field name.
NumberFormat	Gets the number format.
Prefix	Gets the prefix of merge field.
TextAfter	Gets or sets the text after merge field.
TextBefore	Gets or sets the text before merge field.

The following example illustrates how to add the a merge field to the header and footer of the document.

[C#]

```

IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();
paragraph.AppendText("Testing writing Merge Fields into Header");

section.PageSetup.DifferentFirstPage = true;
section.PageSetup.DifferentOddAndEvenPages = true;
paragraph = new WParagraph(doc);
paragraph.AppendText("[ FIRST PAGE Header ]");

```

```

section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph);
paragraph = new WParagraph(doc);
paragraph.AppendField("Field's Name", FieldType.FieldMergeField);
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph);

paragraph = new WParagraph(doc);
paragraph.AppendText("[ FIRST PAGE Footer ]\r");
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph);

```

[VB.NET]

```

Dim section As IWSection = doc.AddSection()
Dim paragraph As IWPParagraph = section.AddParagraph()
paragraph.AppendText("Testing writing Merge Fields into Header")

section.PageSetup.DifferentFirstPage = True
section.PageSetup.DifferentOddAndEvenPages = True
paragraph = New WParagraph(doc)
paragraph.AppendText("[ FIRST PAGE Header ]")
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph)
paragraph = New WParagraph(doc)
paragraph.AppendField("Field's Name", FieldType.FieldMergeField)
section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph)

paragraph = New WParagraph(doc)
paragraph.AppendText("[ FIRST PAGE Footer ]" & Constants.vbCr)
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph)

```

4.4.1.2.2 Embed Field

WEmbedField class represents an embed field type in the Word document. Word does not allow to create an embed field type manually (using Microsoft Word interface). This field is used when the document has embedded objects. This field usually points to the container in the document which encloses the embedded object.



Note: Modification of **WEmbedField** properties can cause document corruption or incorrect document preservation. DocIO preserves only fields of this type.

Class Hierarchy

WTextRange

|

```

WField
|
WEEmbedField

```

Public Property

Name	Description
EntityType	Gets the type of the entity.

4.4.1.2.3 Seq Field

WSeqField class represents a sequence field type in the Word document. To add a sequence field in Microsoft Word, open **Insert** menu, click **Field**, and then click **Seq**. You can find information about the sequence field in the following location: <http://office.microsoft.com/en-us/word/HP051861901033.aspx>.

You can use the **NumberFormat** property to set the numbering format for the fields, and the **CaptionName** property to set the name of the caption.

Public Constructor

Name	Description
WSeqField.WSeqField (IWordDocument)	Initializes a new instance of the WSeqField class.

Public Properties

Name	Description
CaptionName	Gets or sets caption name.
EntityType	Gets the type of the entity.
FormattingString	Gets the formatting string.
NumberFormat	Gets or sets the type of caption numbering. It includes the following options. Number

	Roman
	Alphabetic

[C#]

```
WSeqField field = ( WSeqField )paragraph.AppendField("Sequence Field",
FieldType.FieldSequence );
field.CaptionName = "Sequence Field";
field.NumberFormat = CaptionNumberingFormat.Alphabetic;
```

[VB .NET]

```
Dim field As WSeqField = CType(paragraph.AppendField("Sequence Field",
FieldType.FieldSequence), WSeqField)
field.CaptionName = "Sequence Field"
field.NumberFormat = CaptionNumberingFormat.Alphabetic
```

4.4.1.2.4 Form Field

WFormField is the base abstract class for all form fields in DocIO (WCheckBox, WDropDownFormField and WTextFormField). There are three types of form fields.

- **Text:** text form field
- **CheckBox:** check box form field
- **DropDown:** drop-down form field

The WFormField class holds all the common properties for the form fields. The following are the common properties of the form fields.

- CalculateOnExit
- Enabled: defines whether the status bar form field is enabled
- FormFieldType: defines type of form field
- Help: represents form field help
- MacroOnEnd: defines the name of the macro to run on entry of form field
- MacroOnStart: defines the name of the macro to run on start of form field
- Name: represents the name of the form field
- StatusBarHelp: represents the help to display in the status bar

You can also insert these fields through the **Forms** toolbar in MS Word. The following screen shot illustrates a Forms toolbar in MS Word.



Figure 46: Forms Toolbar

Class Hierarchy

```

WTextRange
  WField
    WFormField

```

Public Properties

Name	Description
CalculateOnExit	Gets or sets calculate on exit property.
Enabled	Gets or sets Enabled property (true if form field enabled).
EntityType	Gets the type of the entity.
FieldPattern	Gets or sets field pattern.
FieldType	Gets or sets field type.
FieldValue	Gets the field value.
FormFieldType	Gets type of this form field.
Help	Gets or sets form field help.
MacroOnEnd	Gets or sets the name of macros on end.
MacroOnStart	Gets or sets the name of macros on start.
Name	Gets form field title name (bookmark name).
StatusBarHelp	Gets or sets the status bar help.
TextFormat	Gets or sets regular text format.

Note that the Form Field ActiveX controls shown in the following screenshot can only be preserved in doc and docx formats and cannot be inserted by using DocIO.



Figure 47: ActiveX Controls

4.4.1.2.4.1 CheckBox

WCheckBox class represents a check box form field in the Word document. To add a check box to the Word document, click **Check Box Form Field** on the **Forms** toolbar.



Figure 48: Forms Panel



Figure 49: CheckBox Properties in MS Word

CheckBoxSize property defines the size of the check box. When the **SizeType** property is set to **Auto**, the size of the check box will be set automatically. You can also set custom size for the check box. This is achieved by setting the **SizeType** property to **Exactly**.

DefaultCheckBoxValue defines the default value of the check box. You can use the **Checked** property to set the value of the check box.

You can use the **AppendCheckBox** function of **WParagraph** to append a check box to the end of the paragraph.

Class Hierarchy

WTextRange



Public Methods

Name	Description
WCheckBox.WCheckBox (IWordDocument)	Initializes a new instance of the WCheckBox class.

Public Properties

Name	Description
CheckBoxSize	Gets or sets size of checkbox (in integer).
Checked	Gets or sets Checked property.
DefaultCheckBoxValue	Gets or sets default checkbox value.

EntityType	Gets the type of the entity.
SizeType	Gets or sets check box size type.

The following example illustrates how to use the WCheckBox class.

[C#]

```
IWordDocument doc = new WordDocument();
doc.EnsureMinimal();
IWParagraph par = doc.LastParagraph;
WCheckBox checkBox = par.AppendCheckBox();
checkBox.Enabled = false;
checkBox.StatusBarHelp = "Help1";
checkBox.Help = "Help2";
checkBox.DefaultCheckBoxValue = true;
checkBox.SizeType = CheckBoxSizeType.Auto;
checkBox.CalculateOnExit = true;
par.AppendText(" CheckBox2: ");
WCheckBox checkBox1 = par.AppendCheckBox();
checkBox1.CheckBoxSize = 30;
checkBox1.SizeType = CheckBoxSizeType.Exactly;
checkBox1.CalculateOnExit = false;
checkBox1.Checked = true;
doc.Save("TestDoc.doc");
```

[VB .NET]

```
Dim doc As IWordDocument = New WordDocument()
doc.EnsureMinimal()
Dim par As IWParagraph = doc.LastParagraph
Dim checkBox As WCheckBox = par.AppendCheckBox()
checkBox.Enabled = False
checkBox.StatusBarHelp = "Help1"
checkBox.Help = "Help2"
checkBox.DefaultCheckBoxValue = True
checkBox.SizeType = CheckBoxSizeType.Auto
checkBox.CalculateOnExit = True
par.AppendText(" CheckBox2: ")
checkBox1 As WCheckBox = par.AppendCheckBox()
checkBox1.CheckBoxSize = 30
checkBox1.SizeType = CheckBoxSizeType.Exactly
checkBox1.CalculateOnExit = False
```

```
checkBox1.Checked = True
doc.Save("TestDoc.doc")
```

4.4.1.2.4.2 DropDown

WDropDownListFormField class represents a drop-down form field in the Word document. To add a drop-down form field to the Word document, click **DropDown Form Field** on the Forms toolbar.



Figure 50: Forms Panel

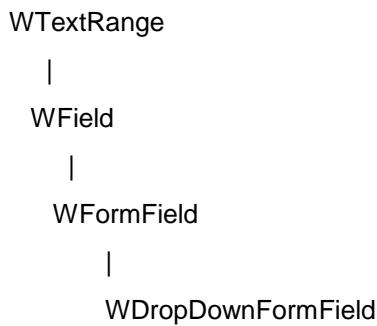


Figure 51: DropDown Form Field Properties

DropDownSelectedIndex property is used to define the index of the record to be displayed in the drop-down form field. The record is chosen among the collection of the drop-down records. This collection is accessible through the **DropDownItems** property.

You can use the **AppendDropDownFormField** function of WParagraph to append drop-down form fields to the end of the paragraph.

Class Hierarchy



Public Constructor

Name	Description
WdropDownForm.FieldWDropDownFormField(IWordDocument)	Initializes a new instance of the WDropDownFormField class.

Public Properties

Name	Description
DropDownItems	Gets drop down items.
DropDownSelectedIndex	Gets or sets the selected drop-down index.

[C#]

```

IWordDocument doc = new WordDocument();
doc.EnsureMinimal();

IWParagraph par = doc.LastParagraph;
WDropDownFormField dropDown = par.AppendDropDownFormField();
dropDown.DropDownItems.Add("One");
dropDown.DropDownItems.Add("Two");
dropDown.DropDownSelectedIndex = 1;
dropDown.CalculateOnExit = true;
dropDown.Enabled = false;

```

```
dropDown.Help = "Help2";
dropDown.StatusBarHelp = "Help1";

doc.Save("TestDoc.doc");
```

[VB .NET]

```
Dim doc As IWordDocument = New WordDocument()
doc.EnsureMinimal()

Dim par As IWParagraph = doc.LastParagraph
Dim dropDown As WDropDownFormField = par.AppendDropDownFormField()
dropDown.DropDownItems.Add("One")
dropDown.DropDownItems.Add("Two")
dropDown.DropDownSelectedIndex = 1
dropDown.CalculateOnExit = True
dropDown.Enabled = False
dropDown.Help = "Help2"
dropDown.StatusBarHelp = "Help1"

doc.Save("TestDoc.doc")
```

4.4.1.2.4.3 Text

WTextFormField class represents a text form field in Word document. To add a text form field to the Word document, click Text Form Field on the Forms toolbar.



Figure 52: Forms Panel

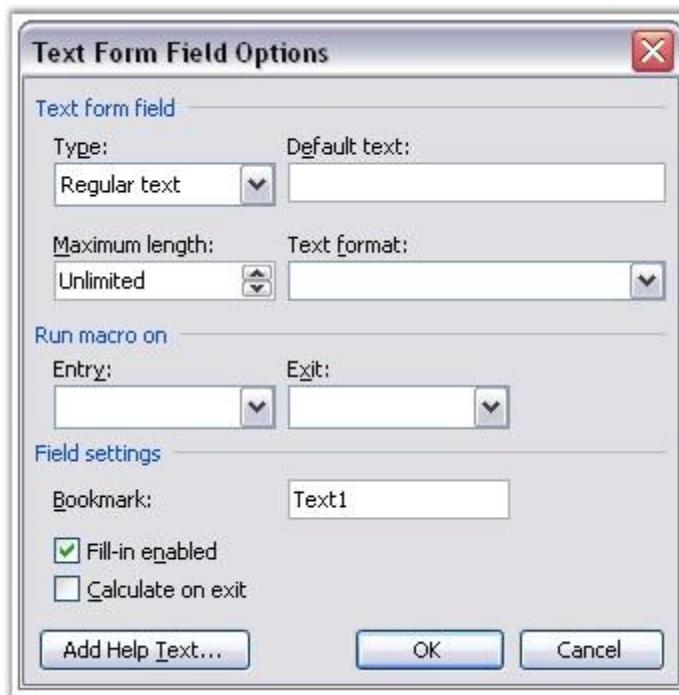


Figure 53: Text Form Field Properties

To set the format of the DocIO text directly from the field, you can use the **StringFormat** property. To get or set the default text for the text form field, you can use the **DefaultText** property.



Note: Text form field will display the default text only when the text of the form field has no value, i.e., when the **TextRange** property (**TextRange.Text**) has no value.

TextRange property is used to set the text of the text form field. **Type** property specifies the type of the text form field. The following are the types of the text form field.

- RegularText
- NumberText
- DateText

Class Hierarchy

WTextRange

|

WField

```

|
WFormField
|
WTextFormField

```

Public Constructor

Name	Description
WTextFormField.WTextFormField(IWordDocument)	Initializes a new instance of the WTextFormField class.

Public Properties

Name	Description
DefaultText	Gets/sets default text for text form field.
MaximumLength	Gets/sets maximum text length.
StringFormat	Gets/sets string text format (text, date/time, number) directly.
TextRange	Gets/sets form field text range.
Type	Get/sets text form field type.

The following three examples illustrate the different variants of text form field usage.

[C#]

Example 1

```

IWordDocument doc = new WordDocument(true);
IWParagraph par = doc.LastParagraph;

// Append text form field to the paragraph.
WTextFormField textField = par.AppendTextFormField("Hello");
textField.TextFormat = TextFormat.Uppercase;
textField.Enabled = false;
textField.Help = "Help2";

```

```

textFormField.StatusBarHelp = "Help1";
textFormField.MacroOnStart = "Test1";
textFormField.CalculateOnExit = true;

Example 2

// In this sample we modify all text form field in document.
foreach (WSection sec in doc.Sections)
{
    foreach (WTextBody body in sec.ChildEntities)
    {
        // Every WTextBody object has a collection of form fields.
        foreach (WFormField ffield in body.FormFields)
        {
            switch (ffield.FormFieldType)
            {
                case FormFieldType.TextInput:
                    WTextFormField textField = (WTextFormField)ffield;
                    textField.Type = TextFormFieldType.DateText;

                    // Setting of default text of form field.
                    textField.DefaultText = "01/01/2007";
                    textField.StringFormat = "MM/dd/yyyy ";

                    // Setting character format of field (not text of form field).
                    // This formatting you can see, when you press Alt+F9 on
                    // document, which has form field.
                    textField.CharacterFormat.FontName = "Comic Sans MS";
                    textField.CharacterFormat.Shadow = true;
                    textField.CharacterFormat.FontSize = 20f;

                    // Setting text of text form field and it's character format.
                    // If textField.TextRange.Text value is not equal to
string.Empty
                    // form field's text will be textField.TextRange.Text, in
other
                    // case textField.DefaultText.
                    textField.TextRange.Text = string.Empty;
                    textField.TextRange.CharacterFormat.FontName = "Comic Sans
MS";
                    textField.TextRange.CharacterFormat.Shadow = true;
                    textField.TextRange.CharacterFormat.FontSize = 20f;
                    textField.TextRange.CharacterFormat.TextColor = Color.Blue;
                    break;

                default:
            }
        }
    }
}

```

```

        break;
    }
}
}
}
```

[VB.NET]**Example 1**

```

Dim doc As IWordDocument = New WordDocument(True)
Dim par As IWParagraph = doc.LastParagraph

' Append text form field to the paragraph.
Dim textField As WTextFormField = par.AppendTextFormField("Hello")
textField.TextFormat = TextFormat.Uppercase
textField.Enabled = False
textField.Help = "Help2"
textField.StatusBarHelp = "Help1"
textField.MacroOnStart = "Test1"
textField.CalculateOnExit = True
```

Example 2

```

' In this sample we modify all text form field in document.
For Each sec As WSection In doc.Sections
    For Each body As WTextBody In sec.ChildEntities

        'Every WTextBody object has a collection of form fields.
        For Each ffield As WFormField In body.FormFields
            Select Case ffield.FormFieldType
                Case FormFieldType.TextInput
                    Dim textField As WTextFormField = CType(ffield,
WTextFormField)
                    textField.Type = TextFormFieldType.DateText

                'Setting of default text of form field.
                    textField.DefaultText = "01/01/2007"
                    textField.StringFormat = "MM/dd/yyyy "

                'Setting character format of field (not text of form field).
                'This formatting you can see, when you press Alt+F9 on
                'document, which has form field.
                    textField.CharacterFormat.FontName = "Comic Sans MS"
                    textField.CharacterFormat.Shadow = True
```

```

textField.CharacterFormat.FontSize = 20.0F

'Setting text of text form field and it's character format.
'If textField.TextRange.Text value is not equal to
string.Empty
'form field's text will be textField.TextRange.Text, in other
'case textField.DefaultText.
textField.TextRange.Text = String.Empty
textField.TextRange.CharacterFormat.FontName = "Comic Sans MS"
textField.TextRange.CharacterFormat.Shadow = True
textField.TextRange.CharacterFormat.FontSize = 20.0F
textField.TextRange.CharacterFormat.TextColor = Color.Blue

Case Else
End Select
Next ffield
Next body
Next sec

```

DocIO also provides option to add/remove form field shading. It specifies whether to turn on the gray shading on form fields using the below code snippet.

[C#]

```
document.PropertiesFormFieldShading = false;
```

[VB .NET]

```
document.Properties.FormFieldShading = False
```

4.4.1.2.5 Hyperlink

A hyperlink is a colored and underlined text or a graphic, which when clicked, directs to a file, or a location in a file, or an HTML page on the World Wide Web, or an HTML page on an Intranet. It includes the path information to another object. The object can be a target on the same document, a file on the same computer, or a uniform resource locator, giving the location of a web page halfway around the world. The process is exactly the same in all cases. Some point on the document is turned into an active spot, which includes the path information.

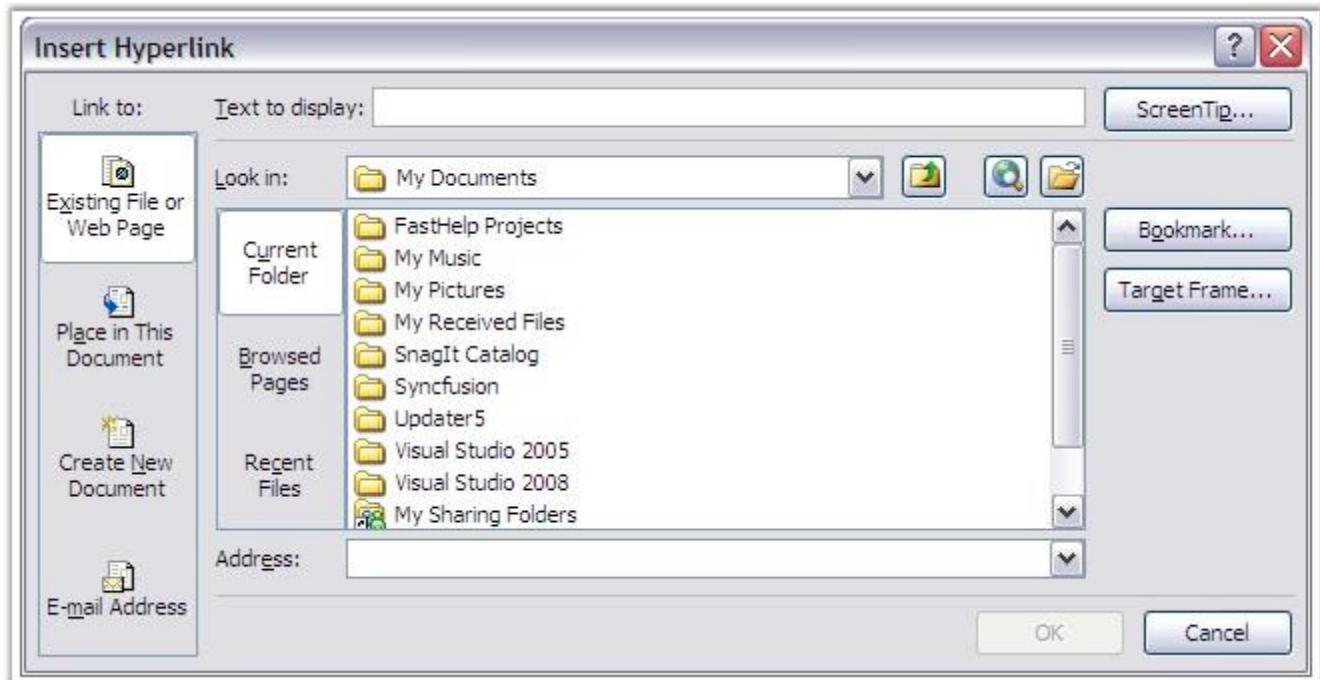


Figure 54: Insert Hyperlink Dialog Box in MS Word

Essential DocIO allows to insert, edit and replace hyperlinks as fields by using the **Hyperlink** class. **HyperlinkType** enumerator specifies the type of the link in use.

Public Constructor

Name	Description
Hyperlink(WField hyperlink)	Initializes a new instance of the Hyperlink class.

Public Properties

Name	Description
FilePath	Gets / sets file path.
Uri	Gets / sets url link.
BookmarkName	Get/sets Bookmark.
HyperlinkType	Gets / sets a HyperlinkType object that indicates the link type. Below are the Hyperlink types supported by DocIO. WebLink - Sets the URI

	EMailLink - Sets the URI Bookmark - Sets the name of the bookmark FileLink - Sets the file path
TextToDisplay	Gets or sets the text, which will be displayed on the place of hyperlink.

The following code illustrates how to find and replace the web hyperlinks.

[C#]

```
WordDocument doc = new WordDocument();
doc.Open("WebLink_1.doc");
Hyperlink hlink = null;

foreach (ParagraphItem item in doc.LastParagraph.Items)
{
    if (item is WField && (item as WField).FieldType ==
    FieldType.FieldHyperlink)
    {
        hlink = new Hyperlink(item as WField);
        if (hlink.Type == HyperlinkType.EMailLink)
        {
            hlink.Type = HyperlinkType.WebLink;
            hlink.TextToDisplay = "Football";
            hlink.Uri = @"\http://www.football.ua\"";
        }
    }
}
doc.Save("WebLink_modified.doc");
```

[VB .NET]

```
Dim doc As New WordDocument()
doc.Open("WebLink_1.doc")
Dim hlink As Hyperlink = Nothing
For Each item As ParagraphItem In doc.LastParagraph.Items
    If TypeOf item Is WField AndAlso TryCast(item, WField).FieldType =
    FieldType.FieldHyperlink Then
        hlink = New Hyperlink(TryCast(item, WField))
        If hlink.Type = HyperlinkType.EMailLink Then
            hlink.Type = HyperlinkType.WebLink
            hlink.TextToDisplay = "Football"
            hlink.Uri = """http://www.football.ua/"""
        End If
    End If
Next
```

```

End If
Next
doc.Save("WebLink_modified.doc")

```

Hyperlink for Images

The following code illustrates how to set hyperlinks for images.

[C#]

```

IWParagraph para = doc.Sections[0].AddParagraph();
WPicture mImage = new WPicture(doc);
mImage.LoadImage(Image.FromFile(@"..\..\Nature.jpg"));

// Scaling Image.
mImage.HeightScale = 50f;
mImage.WidthScale = 50f;
IWField field = para.AppendField("Hyperlink", FieldType.FieldHyperlink);
Hyperlink hlink = new Hyperlink(field as WField);
hlink.Type = HyperlinkType.WebLink;
hlink.Uri = "http://www.syncfusion.com";
hlink.PictureToDisplay = mImage;

```

[VB .NET]

```

Dim para As IWParagraph = doc.Sections(0).AddParagraph()
Dim mImage As New WPicture(doc)
mImage.LoadImage(Image.FromFile("../..\Nature.jpg"))

' Scaling Image.
mImage.HeightScale = 50F
mImage.WidthScale = 50F
Dim field As IWFfield = para.AppendField("Hyperlink",
FieldType.FieldHyperlink)
Dim hlink As New Hyperlink(TryCast(field, WField))
hlink.Type = HyperlinkType.WebLink
hlink.Uri = "http://www.syncfusion.com"
hlink.PictureToDisplay = mImage

```

4.4.1.2.6 Document Variable

A document variable is stored as part of a document or template. Document variables store information about the document. These are useful for document automation because they allow the programmer to store information for future use. For example, some firms use document variables to store Client and Author information, for use in the footer or field code purposes.

Essential DocIO provides support to work with these document variables. You can get or set the variables by using the variable name or index.

Document variables are accessible through the **IDocument.Variables** property.

Variables are added to the document by using the **IDocument.Variables.Add(string variableName, string variableValue)** method. Variables are removed from the document by using the **IDocument.Variables.Remove(string variableName)** method.

These fields could be referred to, in other parts of the document easily. For example, use the "IWParagraph.AppendField(string fieldName, FieldType type)" method, where the 1st argument is the name of the document field and the 2nd argument is `FieldType.FieldDocVariable`.

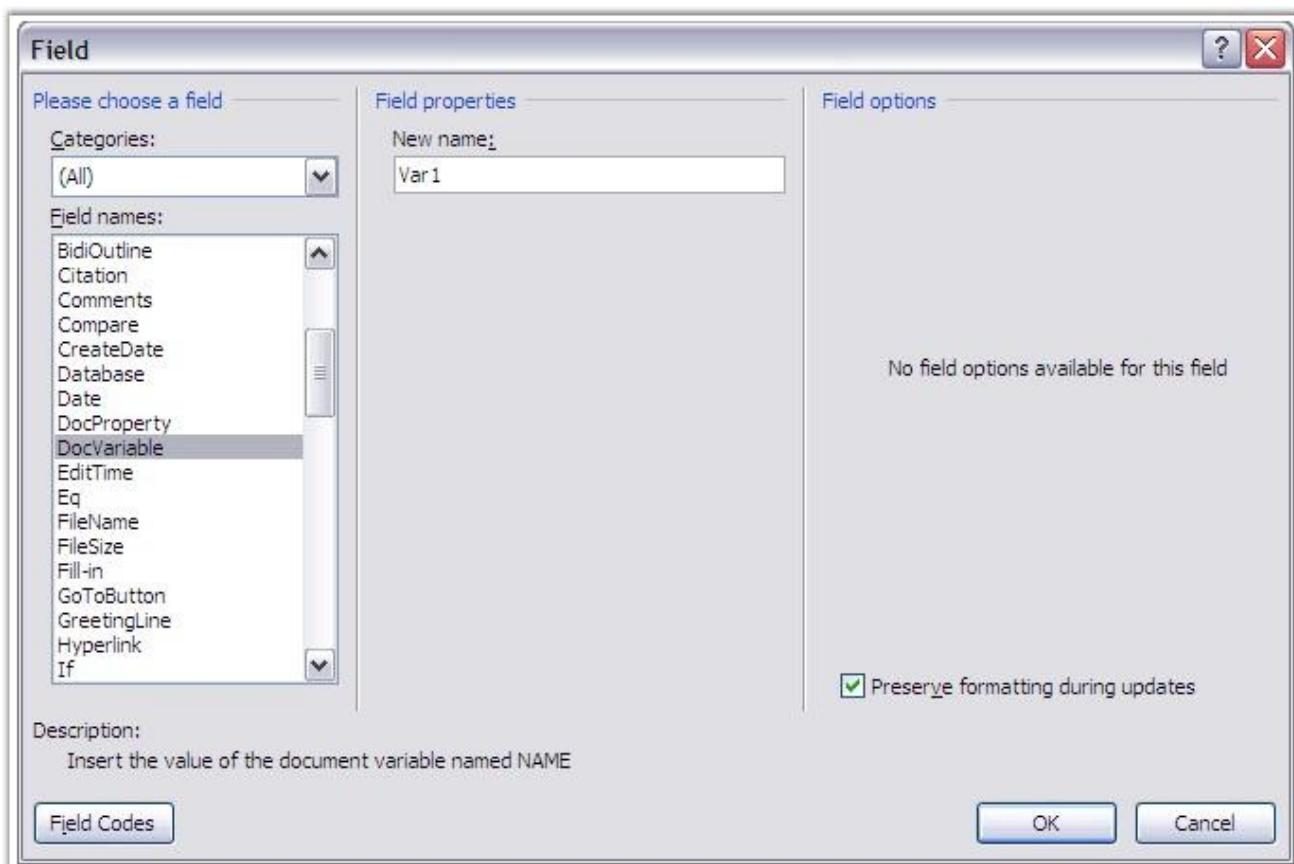


Figure 55: Document Variable Field

[C#]

```
WordDocument doc = new WordDocument();
doc.Open("Sample.doc");
DocVariables v = document1.Variables;

//Add a variable
v.Add("var1", "Author Name");

//Add / modify variables:
v["var2"] = "change name";

Console.WriteLine("Number of Variables:" +
document1.Variables.Count.ToString());
Console.WriteLine("Variable Name:" + document1.Variables.GetNameByIndex(0));
Console.WriteLine("Varaible Value:" +
document1.Variables.GetValueByIndex(0));

doc.Save("SampleModified.doc");
```

[VB.NET]

```
Dim doc As WordDocument = New WordDocument()
doc.Open("Sample.doc")
Dim v As DocVariables = document1.Variables

'Add a variable
v.Add("var1", "Author Name")

'Add / modify variables:
v("var2") = "change name"

Console.WriteLine("Number of Variables:" +
document1.Variables.Count.ToString())
Console.WriteLine("Variable Name:" + document1.Variables.GetNameByIndex(0))
Console.WriteLine("Varaible Value:" + document1.Variables.GetValueByIndex(0))

doc.Save("SampleModified.doc")
```

4.4.1.2.7 Fields Updating Engine

Field in a Word document is a placeholder of data that might change on field update. A field contains field code, field separator, field result and field end. The field code contains information about how the field result is to be calculated and updated. The field result specifies the resultant value of the field. Field updating engine calculates the resultant value based on the field code information and updates the field result with the new value.

The supported fields are:

- = (formula field)
- DATE
- TIME
- DOCVARIABLE
- DOCPROPERTY
- COMPARE
- IF
- NEXTIF
- MERGEREC
- MERGESEQ
- SECTION

= (formula field)

This field is an expression that contains any combination of numbers, bookmarks that refer to numbers, fields resulting in numbers, and the available operators and functions. The expression can refer to values in a table and values returned by functions.

DATE and TIME

This field displays the current date-time, in the format specified by date-time picture switch.

Syntax

```
{ DATE [\@ "Date-Time Picture"] }  
{ TIME [\@ "Date-Time Picture"] }
```

DOCVARIABLE

This field displays the value of the specified document variable.

Syntax

```
{DOCVARIABLE "Name"}
```

DOCPROPERTY

This field displays the value of the specified document property.

Syntax

```
{DOCPROPERTY "Name"}
```

COMPARE

This field compares the two expressions and displays the result "1" if the result of comparison is true or "0" (zero) if the comparison is false.

Syntax

```
{ COMPARE Expression1 Operator Expression2 }
```

IF

This field compares the two expressions and displays the true text if the result of comparison is true or displays the false text if the result of comparison is false.

Syntax

```
{ IF Expression1 Operator Expression2 TrueText FalseText }
```

NEXTIF

This field compares two expressions. If the comparison result is true, the next data record is merged into the current merge document. If the comparison result is false, the next data record is merged into a new merge document.

Syntax

```
{ NEXTIF Expression1 Operator Expression2 }
```

MERGESEQ

This field numbers all the merged records of a mail merge. The number may be different from the value that is inserted by the MERGEREC field.

Syntax

```
{ MERGESEQ }
```

MERGEREC

This field displays the ordinal position of the current data record in a merged document. The number reflects any sorting or filtering that you applied to the data source before the merge.

Syntax

```
{ MERGEREC }
```

SECTION

This field displays the number of the current section.

Syntax

```
{ SECTION }
```

The following example illustrates how to update the fields present in the document.

```
[C#]
// Opening the word document.
WordDocument document = new WordDocument("Input.doc");

// Updating the fields present in the document.
document.UpdateDocumentFields();

// Saving the document.
document.Save("Sample.doc", FormatType.Doc);
```

```
[VB.NET]
' Opening the Word document.
Dim document As WordDocument = New WordDocument("Input.doc")

' Updating the fields present in the document.
document.UpdateDocumentFields()

' Saving the document.
document.Save("Sample.doc", FormatType.Doc)
```

4.4.1.3 Bookmark

A bookmark identifies a location or selection of text that you name, and identifies them for future reference. For example, you might use a bookmark to identify the text that you want to revise later. Instead of scrolling through the document to locate the text, you can access it by using the **Bookmark** dialog box.

The following steps illustrate how to add a bookmark in the Word.

1. Select a text or item that you want to assign as the bookmark.
2. Open the **Insert** menu and click **Bookmark**. This will open the **Bookmark** dialog box.

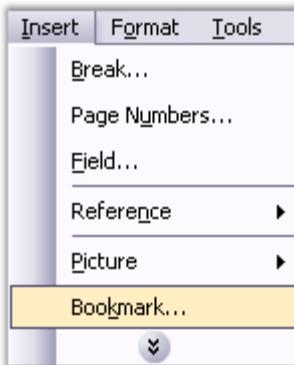


Figure 56: Insert Menu

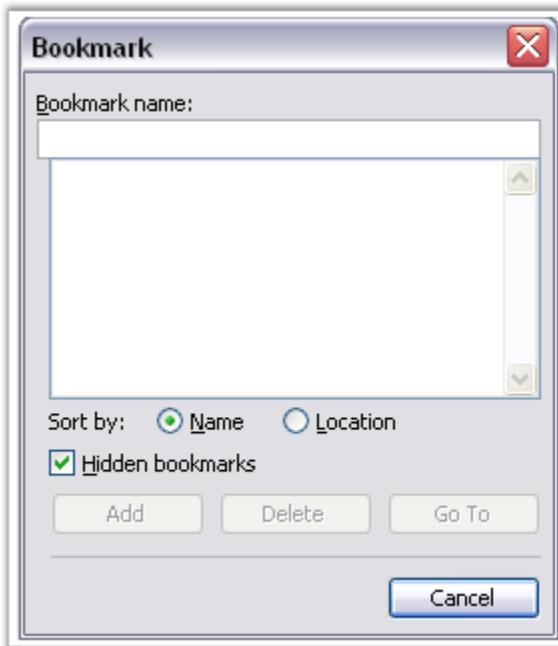


Figure 57: Bookmark Dialog Box

3. Type the name of the bookmark.
4. Click **Add** button.

Note: *Bookmark names must begin with letter and can contain numbers. You cannot include spaces in a bookmark name. However, you can use the underscore character to separate words.*

DocIO gives you a simple mechanism of adding bookmarks to a document, and managing the bookmarks in the document. Every Word document contains a collection of bookmarks. This collection is accessible through the **Bookmarks** property of the Word document. This collection contains objects of the Bookmark type, and enables you to find and delete bookmarks.

You can find a bookmark in the Bookmarks Collection, by specifying its name, by using the **FindByName** procedure. You can also remove a bookmark from the Bookmarks Collection, by specifying its index, by using the **RemoveAt** procedure, or remove a specified bookmark by using the **Remove** procedure.

Every DocIO bookmark consists of the **Bookmark Start** and **Bookmark End**. A **BookmarkStart** class represents a part of a bookmark which identifies the start of a specific bookmark. A **BookmarkEnd** class represents a part of a bookmark which identifies the end of a specific bookmark. BookmarkStart and BookmarkEnd have a common property, **Name**. This property defines the name of the DocIO bookmark.

Class Hierarchy:

ParagraphItem

|

WBookmarkStart

ParagraphItem

|

WBookmarkEnd

BookmarkStart Public Constructor

Name	Description
BookmarkStart.BookmarkStart (IWordDocument, string)	Initializes a new instance of the BookmarkStart class.

BookmarkEnd Public Constructor

Name	Description
WTextFormField.WTextFormField(IWordDocument)	Initializes a new instance of the BookmarkEnd class.

Public Properties

Name	Description
EntityType	Gets the type of the entity.
Name	Gets or sets bookmark name.
OwnerParagraph	Gets owner paragraph.

DocIO provides support to navigate between bookmarks. For details, see [BookmarkNavigator](#).

Note: Modification of bookmarks in the Bookmarks Collection causes document corruption.

The following example illustrates how to use bookmarks.

[C#]

```
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();
paragraph.AppendText("Book with one ");
paragraph.AppendBookmarkStart("one_word");
paragraph.AppendText("word");
paragraph.AppendBookmarkEnd("one_word");
paragraph.AppendText(" selected");

section.AddParagraph();
paragraph = section.AddParagraph();
paragraph.AppendBookmarkStart("beginning_paragraph");
paragraph.AppendText("Beginning of the paragraph selected");

section.AddParagraph();
paragraph = section.AddParagraph();
paragraph.AppendBookmarkStart("bigger_bookmark");
paragraph.AppendText("Smaller bookmark ");
paragraph.AppendBookmarkStart("smaller_bookmark");
```

```
paragraph.AppendText("is inside ");
paragraph.AppendBookmarkEnd("smaller_bookmark");
paragraph.AppendText("of the bigger bookmark");
paragraph.AppendBookmarkEnd("bigger_bookmark");

paragraph = section.AddParagraph();
paragraph.AppendBookmarkStart("multi_paragraph");
paragraph.AppendText("Bookmark starts here and ends in the next paragraph");

paragraph = section.AddParagraph();
paragraph.AppendText("This ");
paragraph.AppendBookmarkStart("overlapped bookmark");
paragraph.AppendText("bookmark over");
paragraph.AppendBookmarkEnd("multi_paragraph");
paragraph.AppendText("laps ");
paragraph.AppendBookmarkEnd("overlapped bookmark");
paragraph.AppendText("with previous one");

doc.Save("Bookmarks.doc");
```

[VB.NET]

```
Dim doc As IWordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph = section.AddParagraph()
paragraph.AppendText("Book with one ")
paragraph.AppendBookmarkStart("one_word")
paragraph.AppendText("word")
paragraph.AppendBookmarkEnd("one_word")
paragraph.AppendText(" selected")

section.AddParagraph()
paragraph = section.AddParagraph()
paragraph.AppendBookmarkStart("beginning_paragraph")
paragraph.AppendText("Beginning of the paragraph selected")

section.AddParagraph()
paragraph = section.AddParagraph()
paragraph.AppendBookmarkStart("bigger_bookmark")
paragraph.AppendText("Smaller bookmark ")
paragraph.AppendBookmarkStart("smaller_bookmark")
paragraph.AppendText("is inside ")
paragraph.AppendBookmarkEnd("smaller_bookmark")
paragraph.AppendText("of the bigger bookmark")
paragraph.AppendBookmarkEnd("bigger_bookmark")
```

```

paragraph = section.AddParagraph()
paragraph.AppendBookmarkStart("multi_paragraph")
paragraph.AppendText("Bookmark starts here and ends in the next paragraph")

paragraph = section.AddParagraph()
paragraph.AppendText("This ")
paragraph.AppendBookmarkStart("overlapped bookmark")
paragraph.AppendText("bookmark over")
paragraph.AppendBookmarkEnd("multi_paragraph")
paragraph.AppendText("laps ")
paragraph.AppendBookmarkEnd("overlapped bookmark")
paragraph.AppendText("with previous one")

doc.Save("Bookmarks.doc")

```

4.4.1.3.1 Bookmark Navigator

BookmarkNavigator is used for navigation between bookmarks in the Word document.

- You can navigate to a bookmark by using the **MoveToBookmark** method. There are two overloads for this method.
 - **MoveToBookmark(string bookmarkName, bool isStart, bool isAfter)**, which moves to the bookmark with the specified name
 - The **isStart** parameter defines whether to move to the bookmark start or bookmark end and the **isAfter** parameter defines whether to set virtual "cursor" after or before the bookmark start or end.
 - **MoveToBookmark(string bookmarkName)**, which moves to the bookmark start with the specified name, and sets the "cursor" before the bookmark start.
- You can insert text between bookmark start and bookmark end by using the **InsertText** method.
- You can insert a table between the bookmark start and bookmark end by using the **InsertTable** method.
- You can insert a paragraph between the bookmark start and bookmark end by using the **InsertParagraph** method.
- You can insert the text body part between the bookmark start and bookmark end by using the **InsertTextBodyPart** method.
- You can insert paragraph items between the bookmark start and bookmark end by using the **InsertParagraphItem** method.
- You can delete content between the bookmark start and bookmark end by using the **DeleteBookmarkContent** method.
- You can replace content between the bookmark start and bookmark end by using the **ReplaceBookmarkContent** method.

- You can get the content between the bookmark start and bookmark end by using the **GetBookmarkContent** method.

The following are the restrictions on the GetBookmarkContent method.

Case 1: The bookmark start is positioned in the text and bookmark end is positioned inside the table or vice versa.

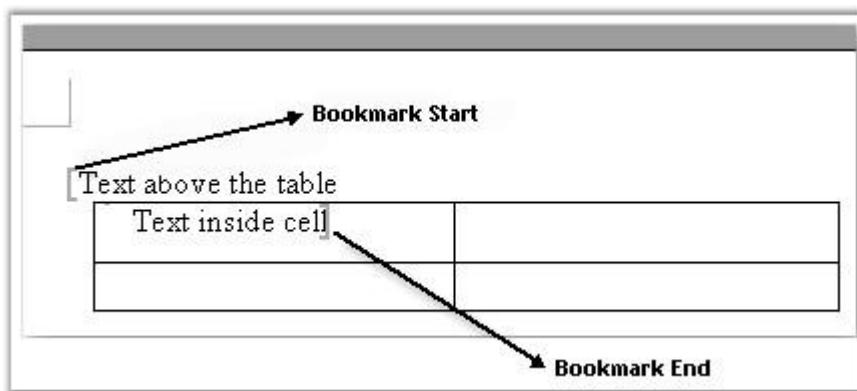


Figure 58: *GetBookmarkContent* method - Restriction 1

Case 2: The bookmark start and bookmark end are positioned inside different tables.

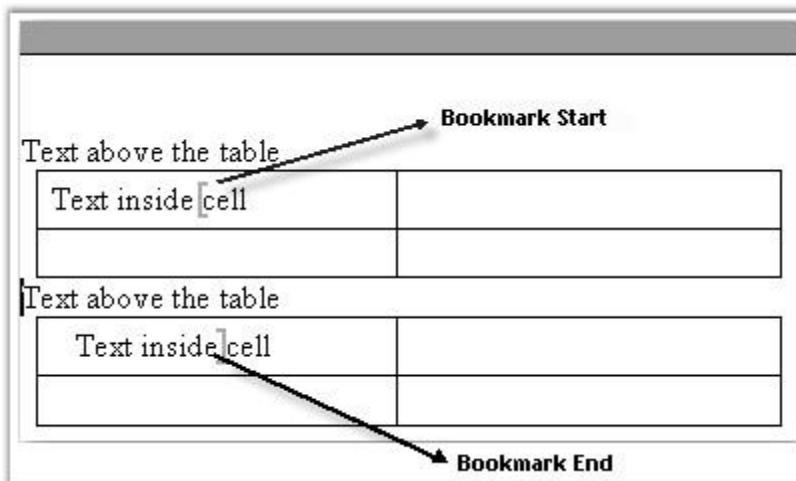


Figure 59: *GetBookmarkContent* method - Restriction 2

Case 3: The bookmark start and bookmark end are positioned inside different cells.

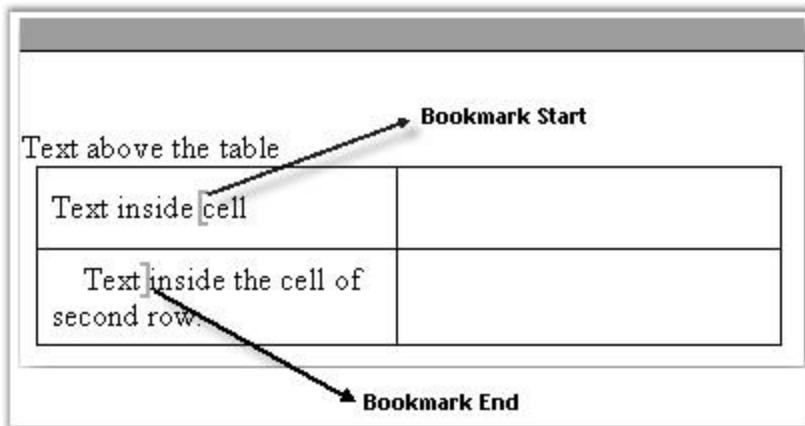


Figure 60: GetBookmarkContent method - Restriction 3

The following exception is raised by the **GetBookmarkContent** method for all the preceding cases: "Bookmark Start and Bookmark End located inside different contents".

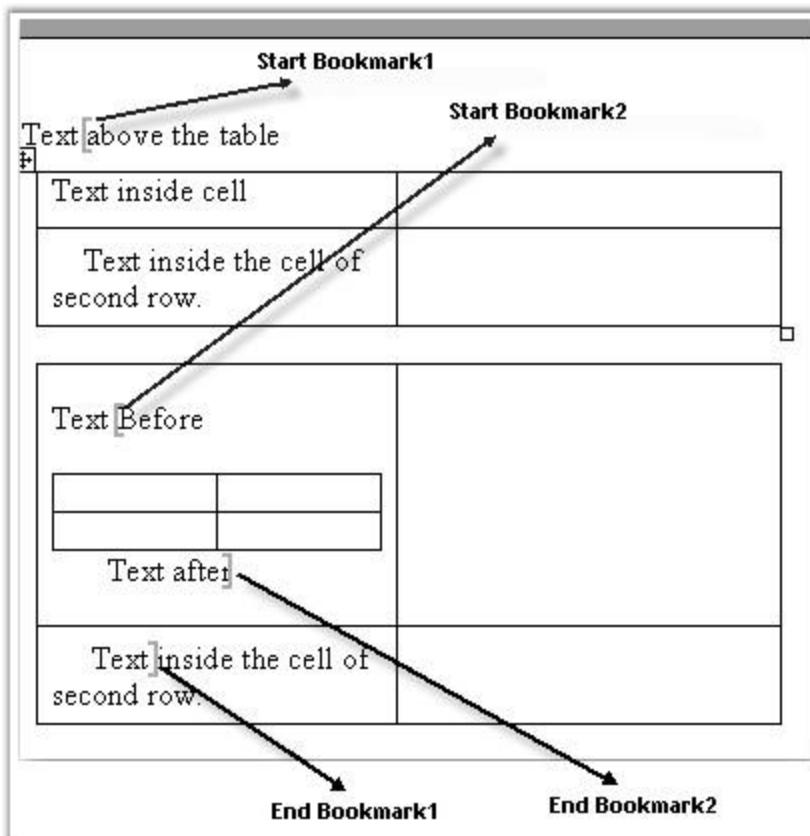


Figure 61: GetBookmarkContent method - Restriction 4



Note: *GetBookmarkContent method works fine for tables, if they are located between the bookmark start and bookmark end.*

Public Constructor

Name	Description
BookmarkNavigator.BookmarkNavigator(IWordDocument)	Initializes a new instance of the BookmarkNavigator class.

Public Properties

Name	Description
CurrentBookmark	Gets the current bookmark.
Document	Gets or sets Document that this object is attached to.

Public Methods

Name	Description
DeleteBookmarkContent	Deletes the bookmark content.
GetBookmarkContent	Gets the bookmark content2.
InsertParagraphItem	Inserts the paragraph item to current position.
InsertTable	Inserts the table.
InsertText	Inserts the text.
MoveToBookmark	Moves to bookmark.
ReplaceBookmarkContent	Replaces bookmark content.
InsertParagraph	Inserts the paragraph.
InsertTextBodyPart	Inserts the body part of the text.

The following example illustrates how to use the BookmarkNavigator class.

[C#]

```
IWordDocument doc = new WordDocument( Path + "BookmarkNavigator.doc" );

BookmarksNavigator bn = new BookmarksNavigator( doc );
bn.MoveToBookmark( "bm_bodypart" );
TextBodyPart part = bn.GetBookmarkContent();
bn.MoveToBookmark( "bm_empty" );
bn.ReplaceBookmarkContent( part );
TextSelection sel = (doc as WordDocument).Find( "11" , false, false );
```

[VB.NET]

```
Dim doc As IWordDocument = New WordDocument(Path & "BookmarkNavigator.doc")

Dim bn As BookmarksNavigator = New BookmarksNavigator(doc)
bn.MoveToBookmark("bm_bodypart")
Dim part As TextBodyPart = bn.GetBookmarkContent()
bn.MoveToBookmark("bm_empty")
bn.ReplaceBookmarkContent(part)
Dim sel As TextSelection = (CType(IIf(TypeOf doc Is WordDocument, doc,
Nothing), WordDocument)).Find("11", False, False)
```

You can also preserve the formatting in the template (target) document while inserting or replacing the bookmark with a string, by deleting the content of the bookmark without deleting its format. The following code illustrates this.

[C#]

```
// Move to the Essential_DocIO bookmark.
bk.MoveToBookmark("Essential_DocIO");

// Delete bookmark content without deleting the format in the target
// document.
bk.DeleteBookmarkContent(false);

// Insert Text
bk.InsertText("Essential XlsIO is a Non UI component that can be used in
both ASP.NET and windows forms applications. The usage is common for both
environments except for the part where the created spreadsheet is saved to
disk or stream in the case of a windows forms application and streamed to
the client browser in the case of asp.net applications.");
```

[VB .NET]

```
' Move to the Essential_DocIO bookmark.
bk.MoveToBookmark("Essential_DocIO")

' Delete bookmark content without deleting the format in the target
document.
bk.DeleteBookmarkContent(false)

' Insert text.
bk.InsertText("Essential XlsIO is a Non UI component that can be used in
both ASP.NET and windows forms applications. The usage is common for both
environments except for the part where the created spreadsheet is saved to
disk or stream in the case of a windows forms application and streamed to
the client browser in the case of asp.net applications.")
```

4.4.1.4 Shapes

Shape is a subdocument of the Word document. Shape is a general name for a group of elements. Shapes can be added to the document with the help of Microsoft Word. The AutoShapes combo box lists the group of shapes. Picture and TextBox are shapes too.

Every shape is represented in Word Document with a special shape marker (among text block) and a block of information about the shape, which is situated in the other part of the document.

DocIO has four classes which represent shapes.

- ShapeObject
- InlineShapeObject
- WPicture
- WTextBox

You have full access to the WPicture and WTextBox classes, i.e., you can read, modify, and create pictures and text boxes. Other shapes are only preserved by DocIO (user cannot create or modify them).

ShapeObject and InlineShapeObject

ShapeObject class represents all types of Word document shapes except pictures and text boxes.

Note: *ShapeObject* and *InlineShapeObject* classes are "read only" classes. You cannot modify shapes. An exception to this is the picture and textbox shape. All shapes except pictures and textbox shapes are only preserved.

ShapeObject class has only one public property – **EntityType**. This property defines the type of the entity, and returns the *EntityType.Shape* for this class.

InlineShapeObject represents all the inline shape objects (except inline textboxes and pictures). Inline shape objects are shape objects which have *Inline* with text layout.

Class Hierarchy

ParagraphItem

|

ShapeObject

|

InlineShapeObject

ShapeObject Public Property

Name	Description
EntityType	Gets the type of the entity.

4.4.1.4.1 Picture

WPicture class represents a picture in the Word document.

A picture is a shape. Positioning properties of *WPicture* class are almost the same as the other shapes.

A picture is positioned by using the *VerticalPosition* and *HorizontalPosition* properties. Measure unit is Point. Relative positioning is defined by using the *HorizontalAlignment* and *VerticalAlignment* properties.

HorizontalAlignment property returns the object of the *ShapeHorizontalAlignment* type. The following are the variants for the *HorizontalAlignment* property.

- None
- Left

- Center
- Right
- Inside
- Outside

VerticalAlignment property returns the object of the ShapeVerticalAlignment type. The following are the variants for the VerticalAlignment property.

- Bottom
- Center
- Inline
- Inside
- None
- Outside
- Top

HorizontalOrigin and **VerticalOrigin** properties define the reference origin, which is used for relative positioning of a picture.

HorizontalOrigin property returns the value of the HorizontalOrigin type. The following are the variants for the HorizontalOrigin property.

- Margin
- Page
- Column
- Character

VerticalOrigin property returns value of VerticalOrigin type. The following are the variants for the VerticalOrigin property.

- Margin
- Page
- Paragraph
- Line

You can set the width and height of the picture by using the **Width** and **Height** properties, and the **HeightScale** and **WidthScale** properties to get or set picture scaling.

The **LoadImage** function is used to set an image by loading the System.Drawing.Image object, or image bytes array. Also, you can use the **AppendPicture** function of the WParagraph class to append a picture to a paragraph.

Class Hierarchy

ParagraphItem

|

WPicture

Public Constructor

Name	Description
WPicture.WPicture (IWordDocument)	Gets the type of the entity.

Public Properties

Name	Description
EntityType	Gets the type of the entity.
Height	Gets or sets picture height.
HeightScale	Gets or sets picture height scale factor in percent.
HorizontalAlignment	Gets or sets picture horizontal alignment.
HorizontalOrigin	Gets sets horizontal origin of the picture.
HorizontalPosition	Gets sets absolute vertical position of the picture.
Image	Gets internal System.Drawing.Image object.
ImageBytes	Gets image byte array.
IsBelowText	Gets or sets whether picture is below image.
Size	Gets or sets size of the picture object.
TextWrappingStyle	Gets or sets text wrapping style of the picture.
TextWrappingType	Gets or sets text wrapping type of the picture.
VerticalAlignment	Gets or sets picture vertical alignment.
VerticalOrigin	Gets or sets absolute horizontal position of the

	picture.
VerticalPosition	Gets or sets text wrapping style of the picture.
Width	Gets or sets picture width (in points). .
WidthScale	Gets or sets picture width scale factor in percent.

Public Methods

Name	Description
AddCaption	Add Caption for current Picture.
LoadImage	Loads image.

The following screen shot illustrates the various layout formats available in MS Word.

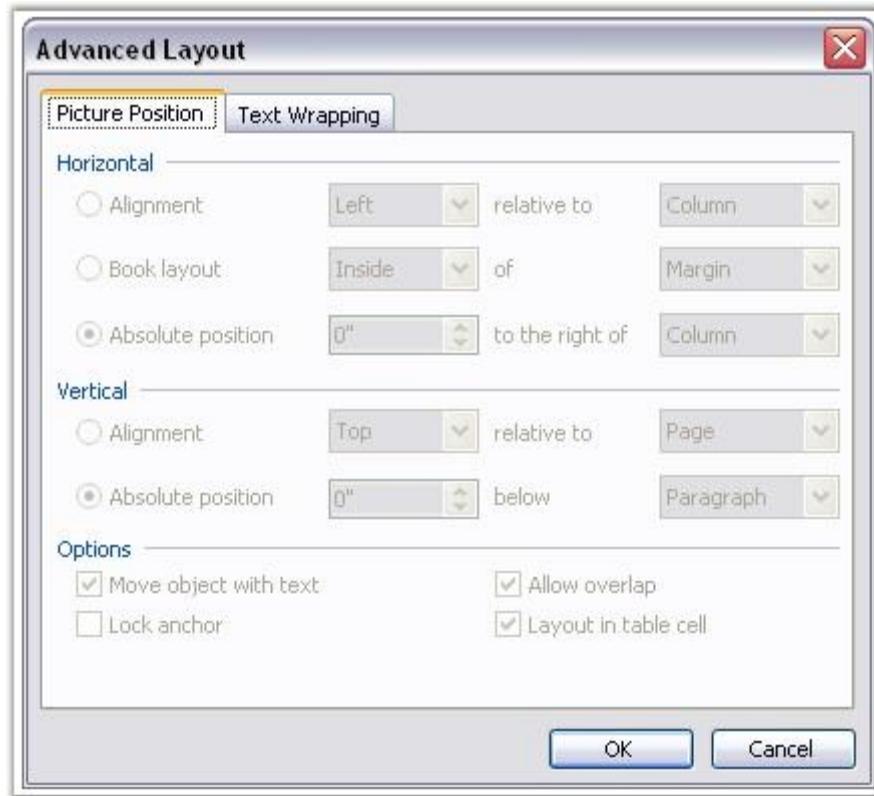


Figure 62: Layout Formats in MS Word

The following code illustrates how to use the **WPicture** class.

[C#]

```
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();
paragraph.AppendText("First image");
IWPicture picture = paragraph.AppendPicture(new Bitmap(ImagesPath +
DEF_IMAGE1_NAME));
picture.HeightScale = 50f;
picture.WidthScale = 50f;

paragraph = section.AddParagraph();
paragraph.AppendText("Second image");
picture = paragraph.AppendPicture(new Bitmap(ImagesPath + DEF_IMAGE2_NAME));
picture.HeightScale = 50f;
picture.WidthScale = 50f;

section.HeadersFooters.OddHeader.Paragraphs.Add(paragraph);
```

[VB.NET]

```
Dim doc As IWordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph = section.AddParagraph()
paragraph.AppendText("First image")
Dim picture As IWPicture = paragraph.AppendPicture(New Bitmap(ImagesPath +
DEF_IMAGE1_NAME))
picture.HeightScale = 50f
picture.WidthScale = 50f

paragraph = section.AddParagraph()
paragraph.AppendText("Second image")
picture = paragraph.AppendPicture(New Bitmap(ImagesPath + DEF_IMAGE2_NAME))
picture.HeightScale = 50f
picture.WidthScale = 50f

section.HeadersFooters.OddHeader.Paragraphs.Add(paragraph)
```

4.4.1.4.2 TextBox

WTextBox class represents a text box in the Word document.

TextBox is a shape. DocIO text box (WTextBox) has two properties which are as follows.

- **TextBoxFormat** – defines formatting of the text box (position, alignment, border colors, and so on)
- **TextBoxBody** – defines the text for the text box

TextBoxFormat property returns the object of the WTextBoxFormat type. For more details, see WTextBoxFormat. **TextBoxBody** property returns the object of the WTextBody type.

You can use the **AppendTextBox** function of the WParagraph class to append a text box to a paragraph.

Class Hierarchy

ParagraphItem

|

WTextBox

Public Constructor

Name	Description
WTextBox.WTextBox (IWordDocument)	Initializes a new instance of the WTextBox class.

Public Properties

Name	Description
ChildEntities	Gets the child entities.
EntityType	Gets the type of the entity.
OwnerParagraph	Gets owner paragraph.
TextBoxBody	Get or sets TextBody value.
TextBoxFormat	Gets or sets TextBoxFormat value.

The following example illustrates how to use the WTextBox and TextBoxFormat classes.

[C#]

```
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();
section.PageSetup.DifferentFirstPage = true;
section.PageSetup.DifferentOddAndEvenPages = true;

// Main doc text boxes
paragraph.AppendText("Testing textboxes");

// 1st text box
IWTextBox mainTextbox = paragraph.AppendTextBox(150, 110);
mainTextbox.TextBoxBody.AddParagraph().AppendText("Textbox text 1");
mainTextbox.TextBoxFormat.FillColor = System.Drawing.Color.Blue;
mainTextbox.TextBoxFormat.LineDashing = LineDashing.LongDashDotDotGEL;
mainTextbox.TextBoxFormat.LineWidth = 4.0f;

// 2nd text box
IWTextBox mainTextbox1 = paragraph.AppendTextBox(150, 100);
mainTextbox1.TextBoxFormat.VerticalPosition = 500;
mainTextbox1.TextBoxBody.AddParagraph().AppendText("Another textbox");
mainTextbox1.TextBoxFormat.FillColor = System.Drawing.Color.Yellow;
mainTextbox1.TextBoxFormat.LineDashing = LineDashing.DashGEL;
mainTextbox1.TextBoxFormat.LineWidth = 3.75f;
mainTextbox1.TextBoxFormat.TextWrappingStyle = TextWrappingStyle.Through;
mainTextbox1.TextBoxFormat.TextWrappingType = TextWrappingType.Both;
mainTextbox1.TextBoxFormat.HorizontalAlignment =
ShapeHorizontalAlignment.Center;
mainTextbox1.TextBoxFormat.VerticalAlignment = ShapeVerticalAlignment.Bottom;

// Header/footer text boxes
paragraph = new WParagraph(doc);
paragraph.AppendText("Hello textboxes");
IWTextBox textbox = paragraph.AppendTextBox(20, 50);
textbox.TextBoxBody.AddParagraph().AppendText("Header textbox");
textbox.TextBoxFormat.FillColor = System.Drawing.Color.Blue;
textbox.TextBoxFormat.LineDashing = LineDashing.LongDashDotDotGEL;
textbox.TextBoxFormat.LineWidth = 4.0f;

IWTextBox textbox1 = paragraph.AppendTextBox(250, 50);
textbox1.TextBoxBody.AddParagraph().AppendText("Header textbox 2");
textbox1.TextBoxFormat.FillColor = System.Drawing.Color.Tomato;
textbox1.TextBoxFormat.VerticalPosition = 250;
textbox1.TextBoxFormat.LineStyle = TextBoxLineStyle.Triple;
textbox1.TextBoxFormat.LineDashing = LineDashing.LongDashGEL;
```

```

textbox1.TextBoxFormat.LineWidth = 6.0f;
textbox1.TextBoxFormat.NoLine = true;

section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph);

paragraph = new WParagraph(doc);
paragraph.AppendText("Hello footer textbox");
IWTextBox textbox2 = paragraph.AppendTextBox(120, 100);
textbox2.TextBoxFormat.VerticalPosition = 5;
textbox2.TextBoxBody.AddParagraph().AppendText("Footer textbox");
textbox2.TextBoxFormat.FillColor = System.Drawing.Color.Yellow;
textbox2.TextBoxFormat.LineDashing = LineDashing.DashGEL;
textbox2.TextBoxFormat.LineWidth = 3.75f;
textbox2.TextBoxFormat.TextWrappingStyle = TextWrappingStyle.Square;
textbox2.TextBoxFormat.HorizontalAlignment = ShapeHorizontalAlignment.Left;
textbox2.TextBoxFormat.VerticalAlignment = ShapeVerticalAlignment.Bottom;
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph);

doc.Save("TextBoxes.doc");

```

[VB.NET]

```

Dim doc As IWordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph = section.AddParagraph()
section.PageSetup.DifferentFirstPage = True
section.PageSetup.DifferentOddAndEvenPages = True

' Main doc text boxes
paragraph.AppendText("Testing textboxes")

' 1st text box
Dim mainTextbox As IWTextBox = paragraph.AppendTextBox(150, 110)
mainTextbox.TextBoxBody.AddParagraph().AppendText("Textbox text 1")
mainTextbox.TextBoxFormat.FillColor = System.Drawing.Color.Blue
mainTextbox.TextBoxFormat.LineDashing = LineDashing.LongDashDotDotGEL
mainTextbox.TextBoxFormat.LineWidth = 4.0f

' 2nd text box
Dim mainTextbox1 As IWTextBox = paragraph.AppendTextBox(150, 100)
mainTextbox1.TextBoxFormat.VerticalPosition = 500
mainTextbox1.TextBoxBody.AddParagraph().AppendText("Another textbox")
mainTextbox1.TextBoxFormat.FillColor = System.Drawing.Color.Yellow
mainTextbox1.TextBoxFormat.LineDashing = LineDashing.DashGEL
mainTextbox1.TextBoxFormat.LineWidth = 3.75f

```

```

mainTextbox1.TextBoxFormat.TextWrappingStyle = TextWrappingStyle.Through
mainTextbox1.TextBoxFormat.TextWrappingType = TextWrappingType.Both
mainTextbox1.TextBoxFormat.HorizontalAlignment =
ShapeHorizontalAlignment.Center
mainTextbox1.TextBoxFormat.VerticalAlignment = ShapeVerticalAlignment.Bottom

' Header/footer text boxes
paragraph = New WParagraph(doc)
paragraph.AppendText("Hello textboxes")
Dim textbox As IWTextBox = paragraph.AppendTextBox(20, 50)
textbox.TextBoxBody.AddParagraph().AppendText("Header textbox")
textbox.TextBoxFormat.FillColor = System.Drawing.Color.Blue
textbox.TextBoxFormat.LineDashing = LineDashing.LongDashDotDotGEL
textbox.TextBoxFormat.LineWidth = 4.0f

Dim textbox1 As IWTextBox = paragraph.AppendTextBox(250, 50)
textbox1.TextBoxBody.AddParagraph().AppendText("Header textbox 2")
textbox1.TextBoxFormat.FillColor = System.Drawing.Color.Tomato
textbox1.TextBoxFormat.VerticalPosition = 250
textbox1.TextBoxFormat.LineStyle = TextBoxLineStyle.Triple
textbox1.TextBoxFormat.LineDashing = LineDashing.LongDashGEL
textbox1.TextBoxFormat.LineWidth = 6.0f
textbox1.TextBoxFormat.NoLine = True

section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph)

paragraph = New WParagraph(doc)
paragraph.AppendText("Hello footer textbox")
Dim textbox2 As IWTextBox = paragraph.AppendTextBox(120, 100)
textbox2.TextBoxFormat.VerticalPosition = 5
textbox2.TextBoxBody.AddParagraph().AppendText("Footer textbox")
textbox2.TextBoxFormat.FillColor = System.Drawing.Color.Yellow
textbox2.TextBoxFormat.LineDashing = LineDashing.DashGEL
textbox2.TextBoxFormat.LineWidth = 3.75f
textbox2.TextBoxFormat.TextWrappingStyle = TextWrappingStyle.Square
textbox2.TextBoxFormat.HorizontalAlignment = ShapeHorizontalAlignment.Left
textbox2.TextBoxFormat.VerticalAlignment = ShapeVerticalAlignment.Bottom
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph)

doc.Save("TextBoxes.doc")

```

4.4.1.4.3 Comment

You can add comments to a Word document. To add a comment to a document, select the text to which you want to apply the comment, open the **Insert** menu and click **Comment**.

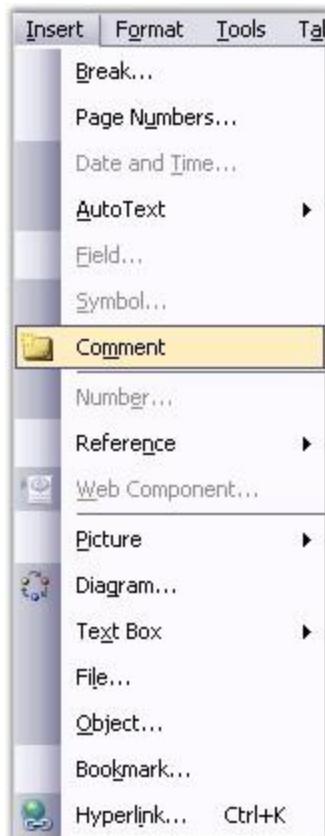


Figure 63: Comment option in Insert Menu

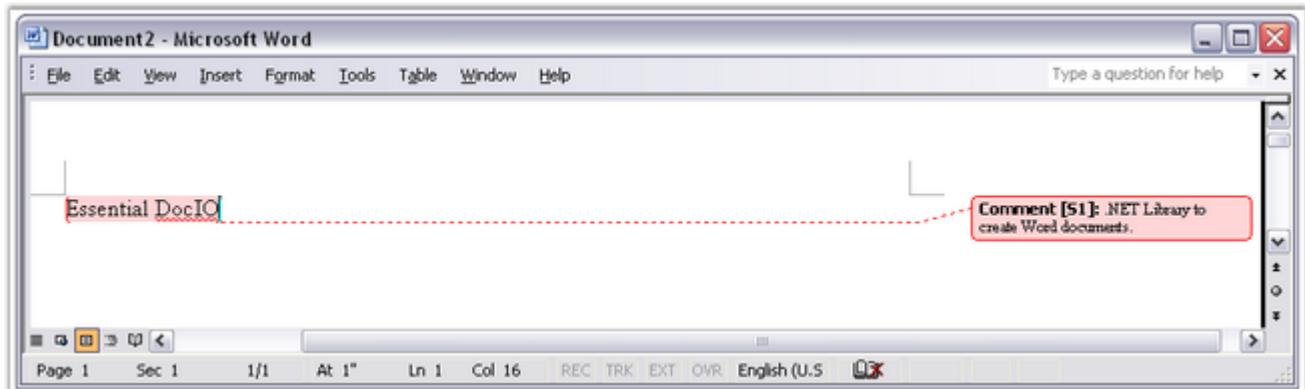


Figure 64: Comment inserted into a Word Document

DocIO has the ability to preserve Word comments, but the creation of comments from APIs or its modification is limited.

Comment is one of the subdocuments of Word. Presentation of this subdocument in the Word document consists of two parts – **special marker**, which defines the comment location in the document, and **special data**, which defines the text and formatting of the comment.

WComment class models the structure and properties of the comments. The following are the main properties of the WComment class.

- **TextBody**: contains text of the comment
- **Format**: specifies the format for the comment

TextBody property returns the object of the **WTextBody** type. Format returns the object of the **WCommentFormat** type.

Class Hierarchy

ParagraphItem

|

WComment

Public Constructor

Name	Description
WComment	Initializes a new instance of the WComment class.

Public Properties

Name	Description
EntityType	Gets the type of the entity.
Format	Gets the format.
TextBody	Gets comment body.

WCommentFormat has the following public properties and methods.

Public Methods

Name	Description
Clone	Creates a new object that is a copy of the current instance.
WCommentFormat	Initializes a new instance of the WCommentFormat class.
RemoveCommentedItems	To remove all the items from the comments.
ReplaceCommentedItems	Replace the commented document content.

Public Properties

Name	Description
User	Gets or sets the user.
UserInitials	Gets or sets the user initials.

The following example illustrates how to search for a comment in the last section of a document. When it is found, the text and format of the comment is changed.

[C#]

```
WSection section = sourceDoc.LastSection;
foreach (IWParagraph para in section.Paragraphs)
{
    foreach (ParagraphItem item in para.Items)
    {
        if (item is WComment)
        {
            WComment comment = item as WComment;
            comment.TextBody.LastParagraph.Text = "NewText";
            comment.Format.User = "TestUser";
        }
    }
}
```

[VB .NET]

```

Dim section As WSection = sourceDoc.LastSection
For Each para As IWParagraph In section.Paragraphs
    For Each item As ParagraphItem In para.Items
        If TypeOf item Is WComment Then
            Dim comment As WComment = CType(IIf(TypeOf item Is WComment, item, Nothing),
WComment)
                comment.TextBody.LastParagraph.Text = "NewText"
                comment.Format.User = "TestUser"
            End If
        Next item
    Next para

```

Comments Collection

You can access comments while browsing through the collection of paragraph items or through the collection of comments by using the **WordDocument.GetComments** method.

Public Methods

Name	Description
Clear	Remove all comments from the document.
RemoveAt	Remove second comments from the document.

The following example illustrates how to get all the comments from the document and remove them.

[C#]

```

WordDocument doc = new WordDocument("sample.doc");
commentsCollection comments = doc.GetComments();

// Remove second comments from the document.
comments.RemoveAt(1);

// Remove all comments from the document.
comments.Clear();

```

[VB]

```

Dim doc As New WordDocument("sample.doc")
Dim comments As CommentsCollection = doc.GetComments()

```

```
' Remove second comments from the document.
comments.RemoveAt(1)

' Remove all comments from the document.
comments.Clear()
```

4.4.1.5 Footnote and Endnote

A **Footnote** is a note of text placed at the bottom of a page in a book or document. It is normally flagged by a superscript number followed by the text where it is referenced to.

An **Endnote** is a note or reference given at the end of a text or a major text section. Endnotes are similar to footnotes, and the only difference is that they are collected together at the end of a chapter or at the end of work.

To add a footnote or endnote to a document:

1. Select the text to which you want to apply the footnote or endnote.
2. Open **Insert** menu.
3. Point to **Reference**, and then click **Footnote** in the **Microsoft Word** menu.



Figure 65: Footnote and Endnote Option in MS Word

DocIO has an ability to preserve Word footnotes and endnotes, but their creation and modification with DocIO API is limited.

Footnotes and Endnotes are the subdocuments of Word. Presentation of these subdocuments in the document consists of two parts namely **special marker**, which defines the footnote or endnote location in the document, and **special data**, which defines the text and formatting of the subdocument.

WFootnote class represents the structure and properties of footnotes and endnotes. As footnotes and endnotes share the same structure in the document, a single class is used to represent them. This class has the **FootnoteType** property, which enables you to add a footnote or endnote. It takes two values namely:

- Footnote
- Endnote

Class Hierarchy

ParagraphItem

|

WFootnote

Public Constructor

Name	Description
WFootnote.WFootnote (IWordDocument)	Initializes a new instance of the WFootnote class

Public Properties

Name	Description
EntityType	Gets the type of the entity
FootnoteType	Gets or sets footnote type: footnote or endnote
IsAutoNumbered	Gets the value indicating if the footnote is auto numbered

MarkerCharacterFormat	Gets the marker character format
TextBody	Gets the text body
SymbolCode	Gets or sets the marker Symbol code
CustomMarker	Defines custom (string) marker for footnote. If footnote is autonumbered, this property won't have any influence (footnote will be autonumbered)

The following code illustrates how to create a Footnote and an Endnote by using Essential DocIO.

[C#]

```
// Creating a new document
WordDocument document = new WordDocument();

// Creating a section
IWSection section1 = document.AddSection();

// Adding a paragraph to a section
IWParagraph paragraph = section1.AddParagraph();

// Creating a footnote
WFootnote footnote = new WFootnote(document);

// Appending endnote
footnote = paragraph.AppendFootnote(Syncfusion.DocIO.FootnoteType.Endnote);

// Setting the footnote character format
footnote.MarkerCharacterFormat.SubSuperScript = SubSuperScript.SuperScript;

// Inserting Text into the paragraph
paragraph.AppendText("Essential DocIO").CharacterFormat.Bold = true;

// Adding footnote text
paragraph = footnote.TextBody.AddParagraph();

paragraph.AppendText("Essential DocIO is a .NET library that has a simple yet and powerful object model which provides the ability to customize the document to a great extent. ");

// Saving the document to disk
document.Save("Sample.doc", Syncfusion.DocIO.FormatType.Doc);
```

[VB .NET]

```

' Creating a new document
Dim document As WordDocument = New WordDocument

' Creating a section
Dim section1 As IWSection = document.AddSection

' Adding a paragraph to a section
Dim paragraph As IWParagraph = section1.AddParagraph

' Creating a footnote
Dim footnote As WFootnote = New WFootnote(document)

' Appending endnote
footnote= paragraph.AppendFootnote(Syncfusion.DocIO.FootnoteType.Endnote)

' Setting the footnote character format
footnote.MarkerCharacterFormat.SubSuperScript = SubSuperScript.SuperScript

' Inserting Text into the paragraph
paragraph.AppendText("Essential DocIO").CharacterFormat.Bold = True

' Adding footnote text
paragraph = footnote.TextBody.AddParagraph
paragraph.AppendText("Essential DocIO is a .NET library that has a simple  
yet and powerful object model which provides the ability to customize the  
document to a great extent. ")

' Saving the document to disk
document.Save("Sample.doc", Syncfusion.DocIO.FormatType.Doc)

```

4.4.1.5.1 Footnote and Endnote Separators

A footnote or endnote separator is a line preserved between the body and the endnotes or footnotes in a Microsoft Word document. A footnote or endnote continuation separator is a line running across the top section indicating that the footnotes or endnotes are carried over from the preceding page if they run over to the second page. A footnote or endnote continuation notice is a special character or word preserved at the bottom of the footer area to indicate that the footnotes or endnotes continue to the next page.

4.4.1.5.1.1 Properties**Public Constructors**

Name	Description
Footnote.Footnote(WordDocument)	Initializes a new instance of the Footnote class.

Endnote.Endnote(WordDocument)	Initializes a new instance of the Endnote class.
-------------------------------	--

Public Properties

Property Name	Description
Separator	Gets or sets the footnote/endnote separator.
ContinuationSeparator	Gets or sets the footnote/endnote continuation separator.
ContinuationNotice	Gets or sets the footnote/endnote continuation notice.

4.4.1.6 Symbol

WSymbol class represents a symbol in the Word document. To insert a symbol, open the **Insert** menu and click **Symbol** in Microsoft Word.

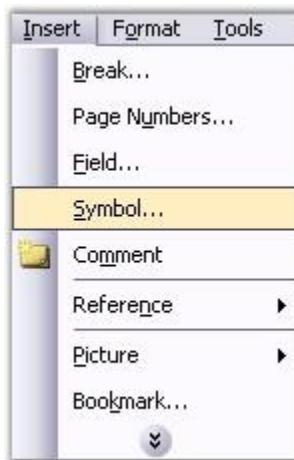


Figure 66: Symbol Option in Insert Menu

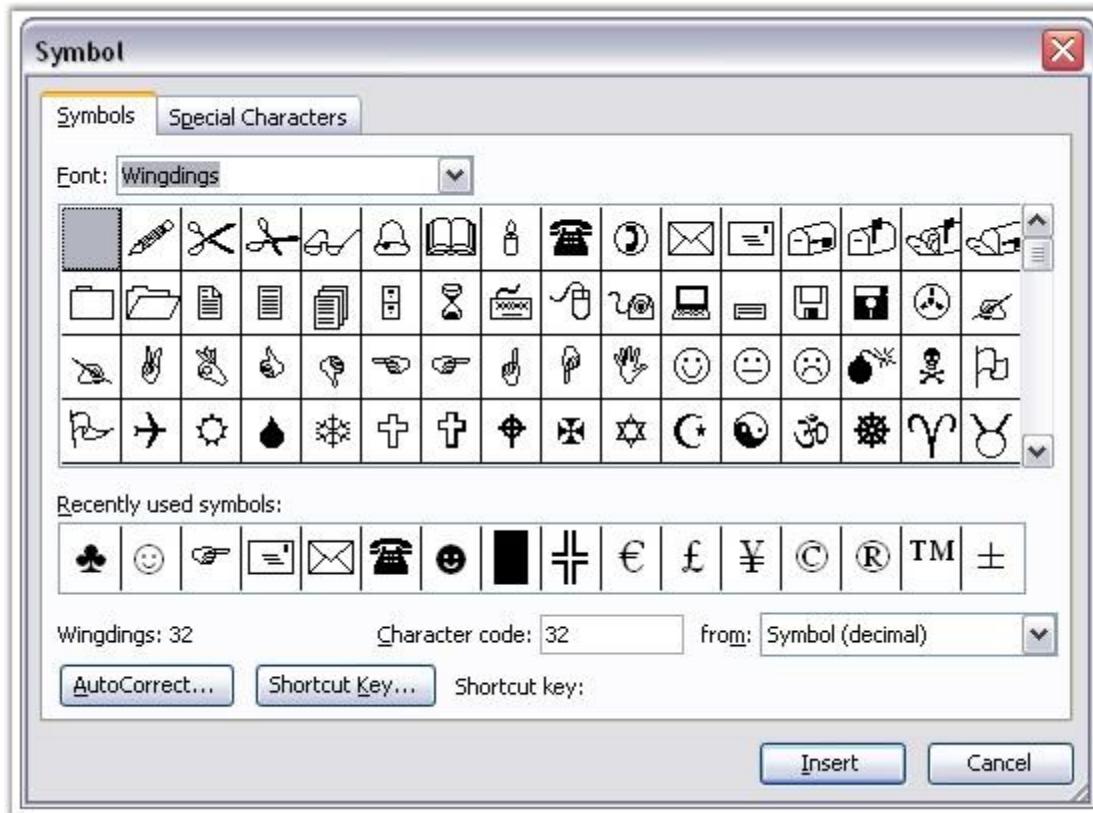


Figure 67: Symbol Dialog Box

You can use the **AppendSymbol** function of WParagraph to insert a symbol by using DocIO.

Also, you can use the **CharacterCode** property to set or get the symbol from the WSymbol class, and the **CharacterFormat** property to set or get the character formatting of the symbol.

Class Hierarchy

ParagraphItem

|

WSymbol

Public Constructor

Name	Description
------	-------------

WSymbol.WSymbol (IWordDocument)	Initializes a new instance of the WSymbol class.
---------------------------------	--

Public Properties

Name	Description
CharacterCode	Gets or sets symbol's character code.
CharacterFormat	Gets character format for the symbol.
EntityType	Gets the type of the entity.
FontName	Gets or sets symbol font name.

The following example illustrates how to use the WSymbol class.

[C#]

```
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();

IWParagraph paragraph = section.AddParagraph();
paragraph.AppendText("Testing symbols");
WSymbol symbol = paragraph.AppendSymbol(140);
symbol.FontName = "Wingdings";

doc.Save("Symbol.doc");
```

[VB .NET]

```
Dim doc As IWordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()

Dim paragraph As IWParagraph = section.AddParagraph()
paragraph.AppendText("Testing symbols")
Dim symbol As WSymbol = paragraph.AppendSymbol(140)
symbol.FontName = "Wingdings"

doc.Save("Symbol.doc")
```

4.4.1.7 Break

Break class represents a break in the Word document. To insert a break, open the **Insert** menu and click **Break** in Microsoft Word.

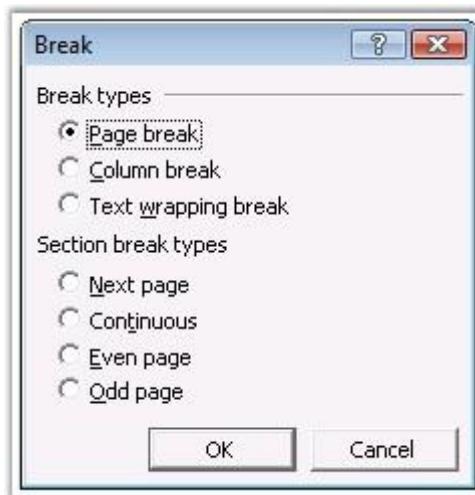


Figure 68: Break Dialog Box

You can use the **AppendBreak** function of **WParagraph** to insert a break by using DocIO. **BreakType** property specifies the type of the break. The following are the types of breaks supported by the Break class.

- PageBreak
- ColumnBreak
- LineBreak



Note: Now, direct support is provided to insert section breaks. Section breaks are inserted by calling the method **InsertSectionBreak**.

Class Hierarchy

ParagraphItem

|

WSymbol

Public Constructors

Name	Description
Break.Break (IWordDocument)	Initializes a new instance of the Break class.

Break.Break (IWordDocument, BreakType)	Initializes a new instance of the Break class.
--	--

Public Properties

Name	Description
BreakType	Gets the type of the break.
EntityType	Gets the type of the entity.

The following example illustrates how to use the Break class.

[C#]

```
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();
IWParagraph para = section.AddParagraph();

para.AppendText("Before line break");
para.AppendBreak(BreakType.LineBreak);
para.AppendText("After line break");

IWParagraph pageBreakPara = section.AddParagraph();
pageBreakPara.AppendText("Before page break");
pageBreakPara.AppendBreak(BreakType.PageBreak);
pageBreakPara.AppendText("After page break");

doc.Save("Breaks.doc");
```

[VB .NET]

```
Dim doc As IWordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()
Dim para As IWParagraph = section.AddParagraph()

para.AppendText("Before line break")
para.AppendBreak(BreakType.LineBreak)
para.AppendText("After line break")

Dim pageBreakPara As IWParagraph = section.AddParagraph()
pageBreakPara.AppendText("Before page break")
pageBreakPara.AppendBreak(BreakType.PageBreak)
pageBreakPara.AppendText("After page break")

doc.Save("Breaks.doc")
```

4.4.1.8 Table Of Contents

The **TableOfContent** class represents the Table Of Contents (TOC) in the Word document.

To add the Table of Contents to the contents in the Word document, follow the steps listed below:

1. Open the Insert menu and click Field.
2. Then select the TOC field type.

You can use the **AppendTOC** method of the WParagraph class, to add a TOC to the DocIO document.

UpperHeadingLevel and **LowerHeadingLevel** define the number of heading levels to be displayed for the TOC. For example, if UpperHeadingLevel is set to 4 and LowerHeadingLevel is set to 3, then TOC will display heading levels from third to fourth.

SetTOCLevelStyle method sets the style for each TOC level. For example, **SetTOCLevelStyle(1, "Normal")** will set the **Normal** style for the first level of TOC.

UpdatingTableOfContents method of WordDocument class updates table of contents field in the word document. Internally Essential DocIO updates page number in table of contents using the Doc to PDF layout engine. Hence the limitations are similar to the limitation in Doc to PDF lay outing.

Note: Updating Table of Contents is not supported in Silverlight platform.

Known Limitations:

The following are the known limitations:

- Currently Auto shapes, foot note and end note, drawing canvas are not preserved in Doc to PDF lay outing, which may leads to updating of incorrect page number.
- Text wrapping support is partially handled in Doc to PDF lay outing, which may leads to updating of incorrect page number.

Class Hierarchy

ParagraphItem

|

TableOfContent

Public Constructors

Name	Description
TableOfContent.TableOfContent (IWordDocument)	Initializes a new instance of the TableOfContent class.
TableOfContent.TableOfContent (IWordDocument, string)	Initializes a new instance of the TableOfContent class.

Public Properties

Name	Description
EntityType	Gets the type of the entity.
IncludePageNumbers	Gets or sets a value indicating whether to include page numbers in TOC. Default value is true.
LowerHeadingLevel	Gets or sets lower heading level (in integer).
RightAlignPageNumbers	Gets or sets a value indicating whether to align page numbers on the right side. Default value is <i>true</i> .
TableID	Gets or sets the table ID (for TC fields).
UpperHeadingLevel	Gets or sets upper heading level (in integer).
UseHeadingStyles	Gets or sets a value indicating whether to use base heading style (Heading 1...Heading 9).
UseHyperlinks	Gets or sets a value indicating whether to insert TOC entries as hyperlinks.
UseOutlineLevels	Gets or sets a value indicating whether to use outline levels.
UseTableEntryFields	Gets or sets a value indicating whether the table will be built from TC fields. If the TableID property is defined, the table is built only from TC fields with the same identifier.

Note: DocIO can't create Outline levels. However, enabling `UseOutlineLevels` property allow TOC creation from existing outline levels.

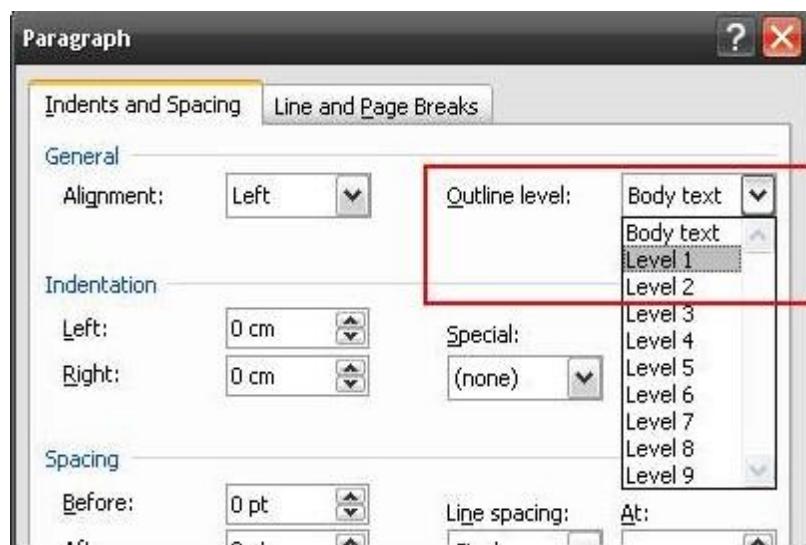


Figure 69: Outline Levels

Public Methods

Name	Description
<code>GetTOCLevelStyleName</code>	Gets the style name for TOC level.
<code>SetTOCLevelStyle</code>	Sets the style for TOC level.

The following code illustrates how to insert TOC, based on custom styles.

[C#]

```
WordDocument doc = new WordDocument();
doc.EnsureMinimal();

WParagraph para = doc.LastParagraph;
TableOfContent toc = para.AppendTOC(1, 1);

toc.UseHeadingStyles = false;

// Set the TOC level style based on which the TOC should be created.
toc.SetTOCLevelStyle(1, "MyStyle1");
```

```

WSection section = doc.LastSection;
WParagraph newPara = section.AddParagraph() as WParagraph;
WTextRange text = newPara.AppendText("My Style1") as WTextRange;
newPara.ApplyStyle("MyStyle1");

// Updates the table of contents.
doc.UpdateTableOfContents();

```

[VB.NET]

```

Dim doc As New WordDocument()
doc.EnsureMinimal()

Dim para As WParagraph = doc.LastParagraph
Dim toc As TableOfContent = para.AppendTOC(1, 1)

toc.UseHeadingStyles = false

' Set the TOC level style based on which the TOC should be created.
toc.SetTOCLevelStyle(1, "MyStyle1")

Dim section As WSection = doc.LastSection
Dim newPara As WParagraph = TryCast(section.AddParagraph(), WParagraph)
Dim text As WTextRange = TryCast(newPara.AppendText("My Style1"), WTextRange)
newPara.ApplyStyle("MyStyle1")

' Updates the table of contents.
doc.UpdateTableOfContents()

```

4.4.1.9 OLE Object

OLE object is used to make the content that is created in one program available in another program. To know what types of content you can insert, click **Insert** tab and select **Object** in the **Text** group.



Note: Only installed programs that support OLE objects appear in the Object dialog box.

Essential DocIO supports insertion and extraction of these OLE objects with small piece of code in both .doc and doc formats. **WOleObject** class is responsible for manipulating OLE objects.

Class Hierarchy

ParagraphItem

|

WOleObject

Object Types

Objects can be either linked to the program or embedded in the program. **Linked Objects** remain as separate files and any changes made to them will be reflected immediately. On the other hand, **Embedded Objects** will be stored in the document that they are inserted and hence changes will not be reflected in them.

If you copy information as an embedded object, the destination file requires more disk space than if you link the information. When the file is opened on another computer, the embedded object can be viewed without having access to the original data. **OleLinkType** property of **WOleObject** is used to set the object type as Embed or Link.

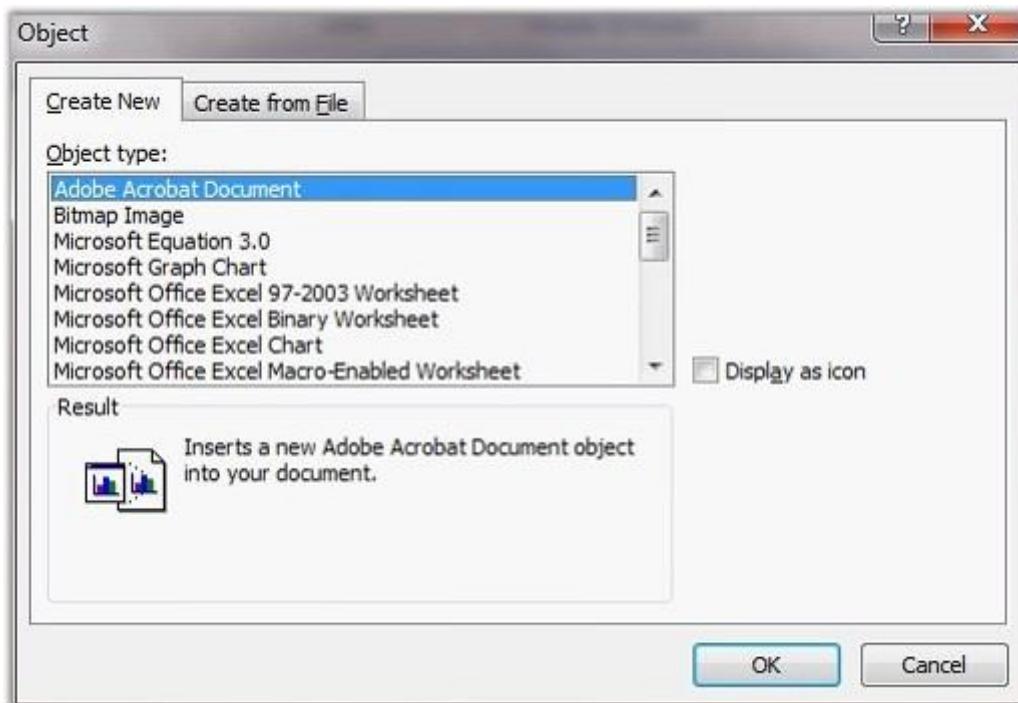


Figure 70: Setting the OLE Object Type in the Object Dialog Box

Inserting Objects

DocIO provides various overloads for the **AppendOleObject** method to enable insertion of objects as bytes or streams by using a single line of code. The following overloads of the AppendOleObject can be used to insert an OLE object.

- AppendOleObject(byte[] oleBytes, WPicture olePicture, OleObjectType type)
- AppendOleObject(byte[] oleBytes, WPicture olePicture, string fileExtension)
- AppendOleObject(Stream oleStream, WPicture olePicture, OleObjectType type)
- AppendOleObject(Stream oleStream, WPicture olePicture, string fileExtension)

The following code examples illustrate insertion of an OLE object by using this method.

[C#]

```
paragraph.AppendOleObject(buffer, pic, OleObjectType.AdobeAcrobatDocument);
paragraph.AppendOleObject(buffer, pic, "pdf");
paragraph.AppendOleObject(stream, pic,
OleObjectType.Excel_97_2003_Worksheet);
paragraph.AppendOleObject(stream, pic, "pdf");
```

[VB]

```
paragraph.AppendOleObject(Buffer, pic, OleObjectType.AdobeAcrobatDocument)
paragraph.AppendOleObject(Buffer, pic, "pdf")
paragraph.AppendOleObject(stream, pic, OleObjectType.Excel_97_2003_Worksheet)
paragraph.AppendOleObject(stream, pic, "pdf")
```

DocIO not only allows inserting objects through container, but also allows inserting objects from disk through file path by using the following overload:

- AppendOleObject(string pathToFile, WPicture olePicture, OleObjectType oleObjectFileType)

DisplayAsIcon

Essential DocIO provides support for embedding OLE objects in a Word document to display them as icons or content using the **DisplayAsIcon** property.

- If the **DisplayAsIcon** property is set to true, the OLE object in the Word document is displayed as an icon.
- If the **DisplayAsIcon** property is set to false, the OLE object in the Word document is displayed as content. This enables the Word document to dynamically update images based on the content present within the OLE object.

 **Note:** Initially DocIO generated documents display the icon (given image) in place of the embedded OLE object. By setting the **DisplayAsIcon** property to true, the icon will not be updated after opening or editing the OLE object using MS Word. However, setting the **DisplayAsIcon** property to false will enable the Word document to update the icons dynamically with the content after opening or editing the OLE object.

Following are the API details of this feature:

Public Constructor

Name	Description
WOleObject.WOLEObject (IWordDocument)	Gets the type of the entity.

Public Properties

Name	Description
OleLinkType	Gets the OLE link type (Embed or Link).
OlePicture	Gets the OLE picture.
EntityType	Gets the type of the entity.
Container	Gets the OLE container.
OleStorageName	Gets or sets the name of the OLE Object storage.
LinkPath	Gets or sets the link path.
LinkType	Gets the type of the OLE object.
NativeData	Gets the native data of embedded OLE object.
PackageFileName	Gets the name of file embedded in the package (only if OleType is "Package").
ObjectType	Gets or sets the type of the OLE object.
DisplayAsIcon	Gets or sets the value indicating whether the OLE object is displayed as an icon or content.

The following code example illustrates how to insert an OLE object in disk to a Word document.

[C#]

```
WordDocument document = new WordDocument();
document.EnsureMinimal();

// Loads the OlePicture from the file.
WPicture pic = new WPicture(document);
pic.LoadImage(Image.FromFile(@"logo.jpg"));
pic.Width = 100f;
pic.Height = 100f;

// Adding new OLE Object.
document.LastParagraph.AppendOleObject(@"Startup.wav", pic,
OleObjectType.Package);
```

[VB.NET]

```
Dim document As WordDocument = New WordDocument()
document.EnsureMinimal()

' Loads the OlePicture from the file.
Dim pic As WPicture = New WPicture(document)
pic.LoadImage(Image.FromFile(@"logo.jpg"))
pic.Width = 100.0F
pic.Height = 100.0F

' Adding new OLE Object.
document.LastParagraph.AppendOleObject(@"startup.wav", pic,
OleObjectType.Package)
```

The following code example illustrates how to extract an OLE object from an existing document and insert it into a new document.

[C#]

```
WordDocument oleSource = new WordDocument("OleTemplate.doc");
WordDocument dest = new WordDocument();
dest.EnsureMinimal();

// Gets OLE object from source document.
WOleObject oleObject = oleSource.LastParagraph.Items[0] as WOLEObject;
WPicture pic = oleObject.OlePicture.Clone() as WPicture;

// Inserts the OLE object into the destination document.
```

```
dest.LastParagraph.AppendOleObject(oleObject.Container, pic,
OleLinkType.Embed);
```

[VB.NET]

```
Dim oleSource As WordDocument = New WordDocument("OleTemplate.doc")
Dim dest As WordDocument = New WordDocument()
dest.EnsureMinimal()

' Gets OLE object from source document.
Dim oleObject As WOLEObject = TryCast(oleSource.LastParagraph.Items(0),
WOLEObject)
Dim pic As WPicture = TryCast(oleObject.OlePicture.Clone(), WPicture)

' Inserts the OLE object into the destination document.
dest.LastParagraph.AppendOleObject(oleObject.Container, pic,
OleLinkType.Embed)
```



Notes: Currently OLE Object support is not available in Silverlight application.

4.4.2 Styles and Formatting

Every word document contains a number of styles. They are as follows.

- Character Style
- Paragraph Style
- Table Style

In Word document style hierarchy, there is also a base style, **Normal**.

DocIO has three classes which represent Word styles.

- **CharacterStyle**: represents Word character style
- **WParagraphStyle**: represents Word paragraph style
- **ListStyle**: represents list properties in the Word paragraph style

Note: DocIO does not support table styles.

DocIO gives user an opportunity to add user-defined paragraph and list styles to the document. For details, see [WParagraphStyle](#).

Collection of DocIO character and paragraph styles is accessible through the **WordDocument.Styles** property. Collection of list styles is accessible through the **WordDocument.ListStyles** property.

DocIO **Style** class is a base class for the **CharacterStyle** and **WParagraphStyle** classes. Style is an abstract class. **CharacterFormat** property of Style class defines character formatting. This property returns the object of **WCharacterFormat** type. **Name** property specifies the name of the style. **BaseStyle** property defines the base style (style inherits formatting of base style). User can apply one of the built-in Word styles by using the **WParagraph.ApplyStyle** method. The built-in styles are accessible through the **BuiltinStyle** enumeration.

Public Properties

Name	Description
CharacterFormat	Gets the character format.
Name	Gets or sets style name.
StyleType	Gets the type of the style.

Public Methods

Name	Description
ApplyBaseStyle	Apply base style for current style.
Clone	Clones itself.

4.4.2.1 Accessing Styles

You can access the collection of styles defined in the document by using the **Styles** property. This collection holds both the built-in and user-defined styles in a document. A particular style can be obtained by its name or index.

The following example illustrates how to access the collection of styles defined in the document.

[C#]

```
// Get a collection of styles defined in the document.
IStyleCollection coll = document.Styles;

// Access particular style.
IStyle style= coll.FindByName(string Stylename) ;
IStyle style= coll.FindByName(string Stylename,StyleType styleType) ;
IStyle style1=coll[0];
```

[VB.NET]

```
' Get a collection of styles defined in the document.
Dim coll As IStyleCollection = document.Styles

' Access particular style.
Dim style As IStyle= coll.FindByName(String Stylename)
Dim style As IStyle= coll.FindByName(String Stylename,StyleType styleType)
Dim style1 As IStyle = coll(0)
```

4.4.2.2 Character Styles and Formats

Character Styles

CharacterStyle class represents character style in the Word document. CharacterStyle has two properties.

StyleType: returns the StyleType.CharacterStyle

BaseStyle: defines the base character style

Collection of DocIO character styles is accessible through the **WordDocument.Styles** property.



Note: DocIO does not provide support to add user-defined character styles to the document.

Class Hierarchy

Style

|

 CharacterStyle

Public Properties

Name	Description
BaseStyle	Gets the base style.
StyleType	Gets the type of the style.
UseContextualAlternates	Gets or sets a value indicating whether to use contextual alternates (Microsoft Word 2010 specific property)
Ligatures	Gets or sets the ligatures type (Microsoft Word 2010 specific property)
NumberForm	Gets or sets the number form type (Microsoft Word 2010 specific property)
NumberSpacing	Gets or sets the number spacing type (Microsoft Word 2010 specific property)
StylisticSet	Gets or sets the stylistic set type (Microsoft Word 2010 specific property)

Public Methods

Name	Description
Clone	Clones itself.
NameToBuiltIn	Converts string style names to BuiltInStyle.

Character Formats

WCharacterFormat class represents character formatting in the Word document.

WCharacterFormat class is used to get or set the formatting for text chunks, special symbol or marker (for example marker of picture, text box, footnote, etc). WCharacterFormat customizes the appearance of the element (for example text chunk or symbol) in the document, starting with font name to the style of texture and spacing between characters.

Class Hierarchy

FormatBase

|
WCharacterFormat

Public Constructor

Name	Description
WCharacterFormat(WCharacterFormat(IWordDocument))	Initializes a new instance of the WCharacterFormat class.

Public Properties

Name	Description
AllCaps	Gets or sets AllCaps property of text.
Bidi	Gets or sets right-to-left property of text.
Bold	Gets or sets bold style.
BoldBidi	Gets or sets bold property for right-to-left text.
Border	Gets border.
CharacterSpacing	Gets or sets space width between characters.
DoubleStrike	Gets or sets doublestrikeout style.
Emboss	Gets or sets emboss property of text.
Engrave	Gets or sets Engrave property of text.
Font	Gets or sets font as System.Drawing.Font.
FontName	Gets or sets font name.
FontNameBidi	Gets or sets font name for right-to-left text.
FontSize	Gets or sets font size (in points).
FontSizeBidi	Gets or sets font size of the right-to-left text (in points).
Hidden	Gets or sets Hidden property of text.
HighlightColor	Gets or sets highlight color of text.
Italic	Gets or sets italic style.

ItalicBidi	Gets or sets italic property for right-to-left text.
LineBreak	Gets or sets line break after.
OutLine	Gets or sets outline character property.
Position	Gets or sets text vertical position.
Shadow	Gets or sets shadow property of text.
SmallCaps	Gets or sets SmallCaps property of text.
Strikeout	Gets or sets strikeout style.
SubSuperScript	Gets or sets subscript / superscript mode.
TextBackgroundColor	Gets or sets text background color.
TextColor	Gets or sets text color.
UnderlineStyle	Gets or sets underline style.
LocaleIdASCII	Gets or sets the locale identifier (language) of the formatted characters.
LocaleIdFarEast	Gets or sets the locale identifier (language) of the formatted Asian characters.

The following example illustrates how to use the WCharacterFormat class.

[C#]

```
// Write different font name / font size
string[] fontNames = new string[] { "Times New Roman", "Verdana", "Symbol",
};

IWSection section = doc.AddSection();
IWPParagraph paragraph;
IWTextRange textRange;

for (int j = 0, len = fontNames.Length; j < len; j++)
{
    paragraph = section.AddParagraph();
    string fontName = fontNames[j];
```

```

        for (int i = 9; i < 40; i++)
    {
        textRange = paragraph.AppendText(fontName + " " + i.ToString() + "
");
        textRange.CharacterFormat.FontSize = i;
        textRange.CharacterFormat.FontName = fontName;
        if (i > 15)
        {
            i += 5;
        }
    }
    IWTTextRange txtRange2 = paragraph.AppendText(fontName + "34,5 ");
    txtRange2.CharacterFormat.FontSize = (float)34.5;
    txtRange2.CharacterFormat.FontName = fontName;
}

// Write bold / italic / underline / strike text.
paragraph = section.AddParagraph();
textRange = paragraph.AppendText("Bold Text_Bold Text      ");
textRange.CharacterFormat.Bold = true;
textRange = paragraph.AppendText("Italic Text_Italic Text      ");
textRange.CharacterFormat.Italic = true;
textRange = paragraph.AppendText("Underline Text_Underline Text      ");
textRange.CharacterFormat.UnderlineStyle = UnderlineStyle.Dash;
textRange = paragraph.AppendText("Strike Text_Strike Text      ");
textRange.CharacterFormat.Strikeout = true;

textRange = paragraph.AppendText("Shadow Text_Shadow Text      ");
textRange.CharacterFormat.Shadow = true;

paragraph = section.AddParagraph();
paragraph = section.AddParagraph();

textRange = paragraph.AppendText("Merged Font Style Text_Merged Font Style
Text");
textRange.CharacterFormat.Bold = true;
textRange.CharacterFormat.Italic = true;
textRange.CharacterFormat.Strikeout = true;
textRange.CharacterFormat.UnderlineStyle = UnderlineStyle.Dash;

// Specify the locale so Microsoft Word recognizes.
// For the list of locale identifiers see
http://www.microsoft.com/globaldev/reference/lcid-all.mspx.

// Sets the locale identifier (language) of the formatted characters.
textRange.CharacterFormat.LocaleIdASCII = (short)LocaleIDs.uk_UA;

```

```
// or
textRange.CharacterFormat.LocaleIdASCII = 1093;

// Sets the locale identifier (language) of the formatted Asian characters.
textRange.CharacterFormat.LocaleIdFarEast = 2052
```

[VB.NET]

```
' Write different font name / font size.
Dim fontNames As String() = New String() { "Times New Roman", "Verdana",
"Symbol", }

Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph
Dim textRange As IWTextRange

Dim j As Integer = 0
len = fontNames.Length
Do While j < len
    paragraph = section.AddParagraph()
    Dim fontName As String = fontNames(j)

    For i As Integer = 9 To 39
        textRange = paragraph.AppendText(fontName & " " & i.ToString() &
" ")
        textRange.CharacterFormat.FontSize = i
        textRange.CharacterFormat.FontName = fontName
        If i > 15 Then
            i += 5
        End If
    Next i
    Dim txtRange2 As IWTextRange = paragraph.AppendText(fontName & "34,5 ")
    txtRange2.CharacterFormat.FontSize = CSng(34.5)
    txtRange2.CharacterFormat.FontName = fontName
    j += 1
Loop

' Write bold / italic / underline / strike text.
paragraph = section.AddParagraph()
textRange = paragraph.AppendText("Bold Text_Bold Text    ")
textRange.CharacterFormat.Bold = True
textRange = paragraph.AppendText("Italic Text_Italic Text    ")
textRange.CharacterFormat.Italic = True
textRange = paragraph.AppendText("Underline Text_Underline Text    ")
textRange.CharacterFormat.UnderlineStyle = UnderlineStyle.Dash
textRange = paragraph.AppendText("Strike Text_Strike Text    ")
```

```

textRange.CharacterFormat.Strikeout = True

textRange = paragraph.AppendText("Shadow Text_Shadow Text    ")
textRange.CharacterFormat.Shadow = True

paragraph = section.AddParagraph()
paragraph = section.AddParagraph()

textRange = paragraph.AppendText("Merged Font Style Text_Merged Font Style
Text")
textRange.CharacterFormat.Bold = True
textRange.CharacterFormat.Italic = True
textRange.CharacterFormat.Strikeout = True
textRange.CharacterFormat.UnderlineStyle = UnderlineStyle.Dash

' Specify the locale so Microsoft Word recognizes.
' For the list of locale identifiers, see
http://www.microsoft.com/globaldev/reference/lcid-all.mspx

```

4.4.2.3 Paragraph Styles

WParagraphStyle class represents paragraph style in the Word document. Paragraph style is a pattern of paragraph formatting. You can also apply custom paragraph styles in MS Word.

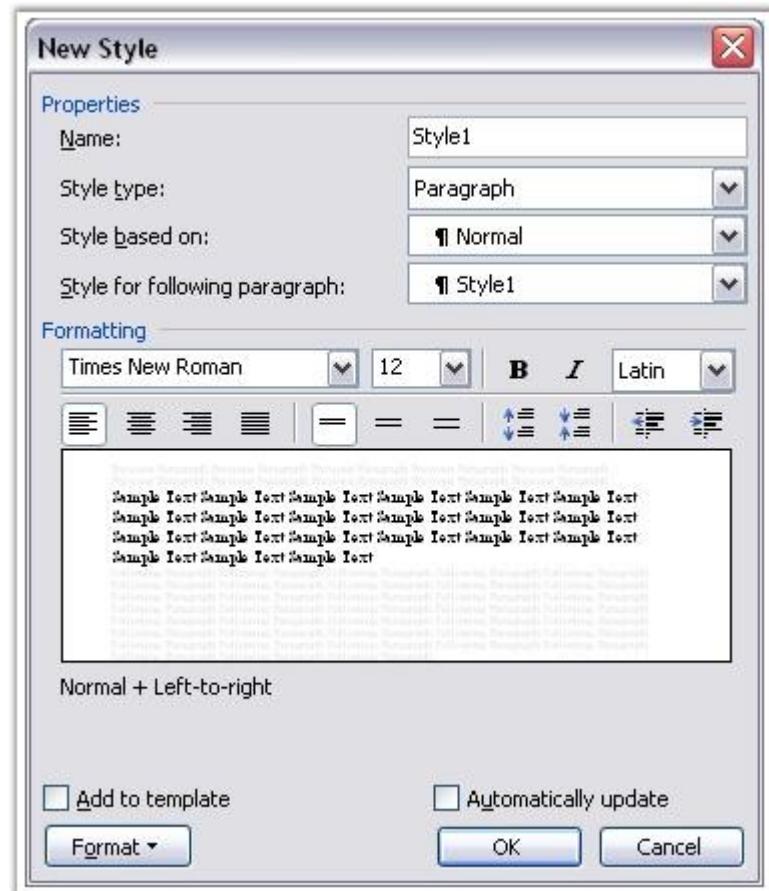


Figure 71: Paragraph Styles

WParagraphStyle class has three properties.

- **ParagraphFormat:** defines formatting for the paragraph to which the style is applied to
- **BaseStyle:** defines base paragraph style
- **StyleType:** returns type of style (StyleType.ParagraphStyle)

You can add your own paragraph styles to the document. Collection of DocIO paragraph styles is accessible through **WordDocument.Styles** property. You can apply one of the built-in Word paragraph styles by using the **WParagraph.ApplyStyle** method.

Also, you can use the **ApplyBaseStyle** method to apply the base style for the current paragraph style.

Class Hierarchy

Style

|

WParagraphStyle

Public Constructor

Name	Description
WParagraphStyle.WParagraphStyle(IWordDocument)	Initializes a new instance of the WParagraphStyle class.

Public Properties

Name	Description
BaseStyle	Gets a base style of paragraph.
ParagraphFormat	Gets formatting of paragraph.
StyleType	Gets the type of the style.

Public Methods

Name	Description
ApplyBaseStyle	Applies base style for current style.
Clone	Clones itself.

The following example illustrates how to create user-defined paragraph styles by using DocIO.

[C#]

```
IWordDocument doc = new WordDocument();

IWParagraphStyle style = (IWParagraphStyle)doc.AddParagraphStyle("Normal");

style = (IWParagraphStyle)doc.AddParagraphStyle("UserStyle_Heading1");
style.CharacterFormat.Bold = true;
style.CharacterFormat.FontName = "Verdana";
style.CharacterFormat.FontSize = 25;

style = (IWParagraphStyle)doc.AddParagraphStyle("UserStyle_Heading2");
```

```
style.CharacterFormat.Italic = true;
style.CharacterFormat.FontName = "Verdana";
style.CharacterFormat.FontSize = 20;

style = (IWParagraphStyle)doc.AddParagraphStyle("UserStyle_Heading3");
style.CharacterFormat.Bold = true;
style.CharacterFormat.FontName = "Times New Roman";
style.CharacterFormat.FontSize = 20;
style.CharacterFormat.UnderlineStyle = UnderlineStyle.Single;

IWSection section = doc.AddSection();

for (int i = 0; i < doc.Styles.Count; i++)
{
    style = (IWParagraphStyle)doc.Styles[i];
    IWParagraph paragraph = section.AddParagraph();
    paragraph.ApplyStyle(style.Name);
    paragraph.AppendText("[ Style Applied ]: ");
    paragraph.AppendText(style.Name);
}
section.AddParagraph();
doc.Save("UserStyle.doc");
```

[VB.NET]

```
Dim doc As IWordDocument = New WordDocument()

Dim style As IWParagraphStyle = CType(doc.AddParagraphStyle("Normal"),
IWParagraphStyle)
Dim style = CType(doc.AddParagraphStyle("UserStyle Heading1"),
IWParagraphStyle)
style.CharacterFormat.Bold = True
style.CharacterFormat.FontName = "Verdana"
style.CharacterFormat.FontSize = 25

style = CType(doc.AddParagraphStyle("UserStyle_Heading2"), IWParagraphStyle)
style.CharacterFormat.Italic = True
style.CharacterFormat.FontName = "Verdana"
style.CharacterFormat.FontSize = 20

style = CType(doc.AddParagraphStyle("UserStyle_Heading3"), IWParagraphStyle)
style.CharacterFormat.Bold = True
style.CharacterFormat.FontName = "Times New Roman"
style.CharacterFormat.FontSize = 20
style.CharacterFormat.UnderlineStyle = UnderlineStyle.Single
```

```
Dim section As IWSection = doc.AddSection()

Dim i As Integer = 0

Do While i < doc.Styles.Count
    style = CType(doc.Styles(i), IWParagraphStyle)
Dim paragraph As IWParagraph = section.AddParagraph()
    paragraph.ApplyStyle(style.Name)
    paragraph.AppendText("[ Style Applied ]: ")
    paragraph.AppendText(style.Name)
    i += 1
Loop
section.AddParagraph()
doc.Save("UserStyle.doc")
```

The following code illustrates how to apply built-in paragraph styles to the Word document.

[C#]

```
IWordDocument doc = new WordDocument();
doc.EnsureMinimal();

doc.LastParagraph.AppendText("Heading 1");
doc.LastParagraph.ApplyStyle(BuiltinStyle.Heading1);

doc.Save("BuiltinStyle.doc");
```

[VB.NET]

```
Dim doc As IWordDocument = New WordDocument()

doc.EnsureMinimal()

doc.LastParagraph.AppendText("Heading 1")
doc.LastParagraph.ApplyStyle(BuiltinStyle.Heading1)

doc.Save("BuiltinStyle.doc")
```

4.4.2.4 Paragraph Formats

WParagraphFormat class represents paragraph formatting in the Word document. To apply paragraph formatting in MS Word, open **Format** menu and click **Paragraph**.

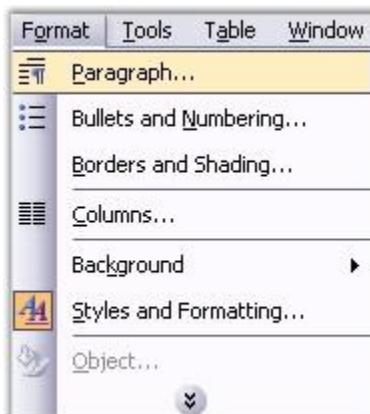


Figure 72: Paragraph Option in Format Menu

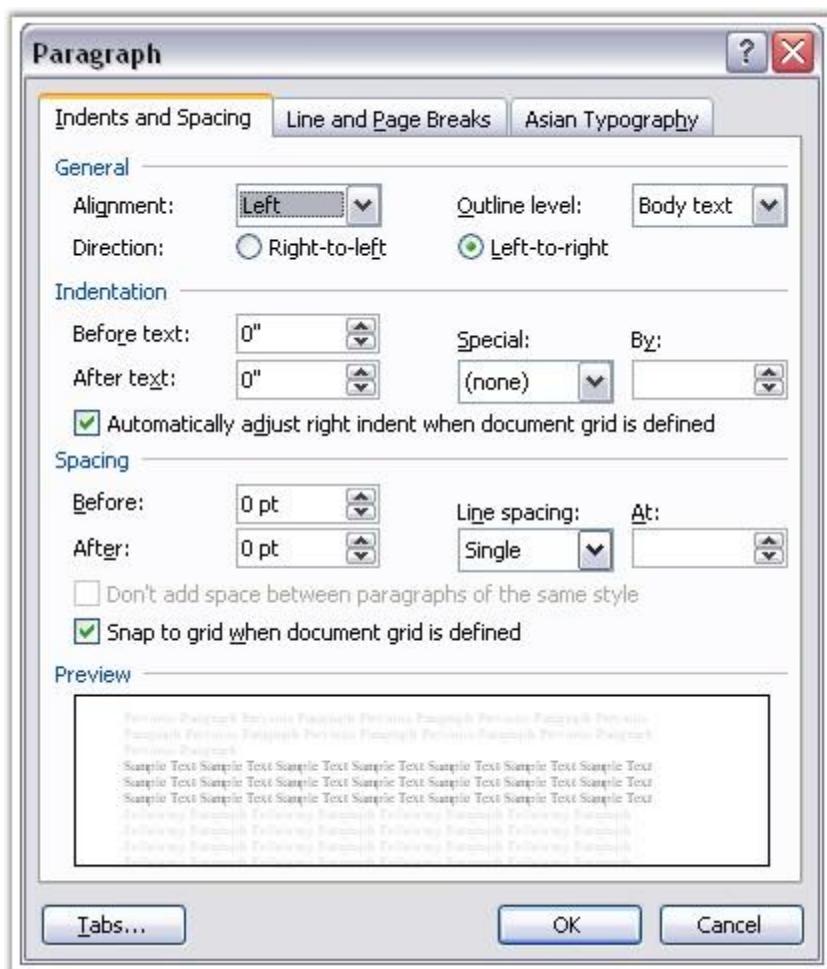


Figure 73: Paragraph Dialog Box

WParagraphFormat class gets or sets the formatting for the paragraph. **Tabs** property returns the collection of tabs.

Class Hierarchy

```
FormatBase
|
WParagraphFormat
```

Public Constructors

Name	Description
WParagraphFormat.WParagraphFormat ()	Initializes a new instance of WParagraphFormat class.
WParagraphFormat.WParagraphFormat (IWordDocument)	Initializes a new instance of WParagraphFormat class.

Public Properties

Name	Description
AfterSpacing	Gets or sets the spacing (in points) after the paragraph.
BackColor	Gets or sets background color of the paragraph.
BeforeSpacing	Gets or sets the spacing (in points) before the paragraph.
Bidi	Gets or sets right-to-left property of the paragraph.
Borders	Gets collection of borders in the paragraph.
ColumnBreakAfter	True if a column break is forced after the paragraph.
FirstLineIndent	Gets or sets first paragraph line indent.
HorizontalAlignment	Gets or sets horizontal alignment for the

	paragraph.
Keep	True if all lines in the paragraph are to remain on the same page.
KeepFollow	True if the paragraph is to remains on the same page as the paragraph that follows it.
FirstLineIndent	Gets or sets first paragraph line indent (in points).
HorizontalAlignment	Gets or sets horizontal alignment for the paragraph.
Keep	True if all lines in the paragraph are to remain on the same page.
KeepFollow	True if the paragraph is to remains on the same page as the paragraph that follows it.
LeftIndent	Gets or sets the value that represents the left indent for paragraph (in points)
LineSpacing	Gets or sets the line spacing property of the paragraph (in points) that specifies the space between two paragraphs.
LineSpacingRule	Gets or sets the line spacing rule property of the paragraph.
PageBreakAfter	True if a page break is forced after the paragraph.
PageBreakBefore	True if a page break is forced before the paragraph.
RightIndent	Gets or sets the right indent for paragraph (in points).
MirrorIndents	Gets or sets a value indicating whether the indentation type is mirror indents (Microsoft Word 2007 and 2010 specific property)

[C#]

```
para.ParagraphFormat.OutlineLevel = OutlineLevel.Level8;
```

[VB]

```
para.ParagraphFormat.OutlineLevel = OutlineLevel.Level8
```

4.4.2.4.1 Tabs

Tabs class represents a tab collection within a paragraph. **Tab** class represents a single tab within the tab collection.

Class Hierarchy

```
WParagraphFormat
|
Tabs
```

Public Properties

Name	Description
Justification	Gets or sets the tab justification.
TabLeader	Gets or sets the tab leader.
Position	Gets or sets the tab position.
DeletePosition	Gets or sets the Clear tab position.

Public Methods

Name	Description
AddTab()	Adds default tab to the paragraph.
AddTab(float position)	Adds tab at the specified position.
AddTab(float position, TabJustification justification, TabLeader leader)	Adds tab at specified location with the specified tab justification and leader.
RemoveAt(int index)	Removes tab at specified index from the tab collection.

RemoveByTabPosition(float position)	Removes tab at specified tab position from the tab collection.
-------------------------------------	--

The following code examples illustrate how to add the tab to the paragraph and delete the tab from the paragraph.

[C#]

```
//Create a new instance for the word document
WordDocument document = new WordDocument();
//Add one section to the document
IWSection section=document.AddSection();
//Add one paragraph to the section
IWParagraph paragraph=section.AddParagraph();

//Add tab stop at the postion 36 with tab justification[Left] and tab
leader[dotted]
paragraph.ParagraphFormat.Tabs.AddTab(36,TabJustification.Left
,Syncfusion.DocIO.DLS.TabLeader.Dotted );
//Add tab stop at the postion 80 with tab justification[Right] and tab
leader[Hyphenated]
paragraph.ParagraphFormat.Tabs.AddTab(80,TabJustification.Right
,Syncfusion.DocIO.DLS.TabLeader.Hyphenated );
//Add tab stop at the postion 144 with tab justification[Center] and with no
tab leader
paragraph.ParagraphFormat.Tabs.AddTab(144,TabJustification.Centered
,Syncfusion.DocIO.DLS.TabLeader.NoLeader );

//Remove tab at index 1 from the tab collection
paragraph.ParagraphFormat.Tabs.RemoveAt(1);
//Remove tab at the position 144
paragraph.ParagraphFormat.Tabs.RemoveByTabPosition(144);
//Append tab character
paragraph.AppendText("\t");
//Append Text to the paragraph
paragraph.AppendText("Tabs are added and removed");
//Save the word document
document.Save("Sample.doc",FormatType.Doc );
```

[VB]

```
'Create a new instance for the word document
Dim document As New WordDocument()
'Add one section to the document
Dim section As IWSection = document.AddSection()
'Add one paragraph to the section
Dim paragraph As IWParagraph = section.AddParagraph()
```

```

'Add tab stop at the position 36 with tab justification[Left] and tab
leader[dotted]
paragraph.ParagraphFormat.Tabs.AddTab(36, TabJustification.Left,
Syncfusion.DocIO.DLS.TabLeader.Dotted)
'Add tab stop at the position 80 with tab justification[Right] and tab
leader[Hyphenated]
paragraph.ParagraphFormat.Tabs.AddTab(80, TabJustification.Right,
Syncfusion.DocIO.DLS.TabLeader.Hyphenated)
'Add tab stop at the position 144 with tab justification[Center] and with no
tab leader
paragraph.ParagraphFormat.Tabs.AddTab(144, TabJustification.Centered,
Syncfusion.DocIO.DLS.TabLeader.NoLeader)

'Remove tab at index 1 from the tab collection
paragraph.ParagraphFormat.Tabs.RemoveAt(1)
'Remove tab at the position 144
paragraph.ParagraphFormat.Tabs.RemoveByTabPosition(144)
'Append tab character
paragraph.AppendText("\t")
'Append Text to the paragraph
paragraph.AppendText("Tabs are added and removed")
'Save the word document
document.Save("Sample.doc", FormatType.Doc)

```

4.4.2.5 Lists

ListStyle class represents list properties in the Word paragraph style. Collection of list styles is accessible through the **WordDocument.ListStyles** property.

You can create your own list style. To add a list style by using DocIO, use the **WordDocument.AddListStyle** method. Every list style has its own name. You can use the **Name** property to access the style name. There are list styles of two types.

- Numbered
- Bulleted

You can specify the type of the list style by using the **ListType** property. Every ListStyle object contains the collection of list levels. This collection can contain from one to nine levels (maximum number of list levels). Collection of list levels is accessible through the **Levels** property. This property returns the ListLevelCollection object. List Level Collection (ListLevelCollection class) contains objects of the **WListLevel** class.

List Styles in MS Word

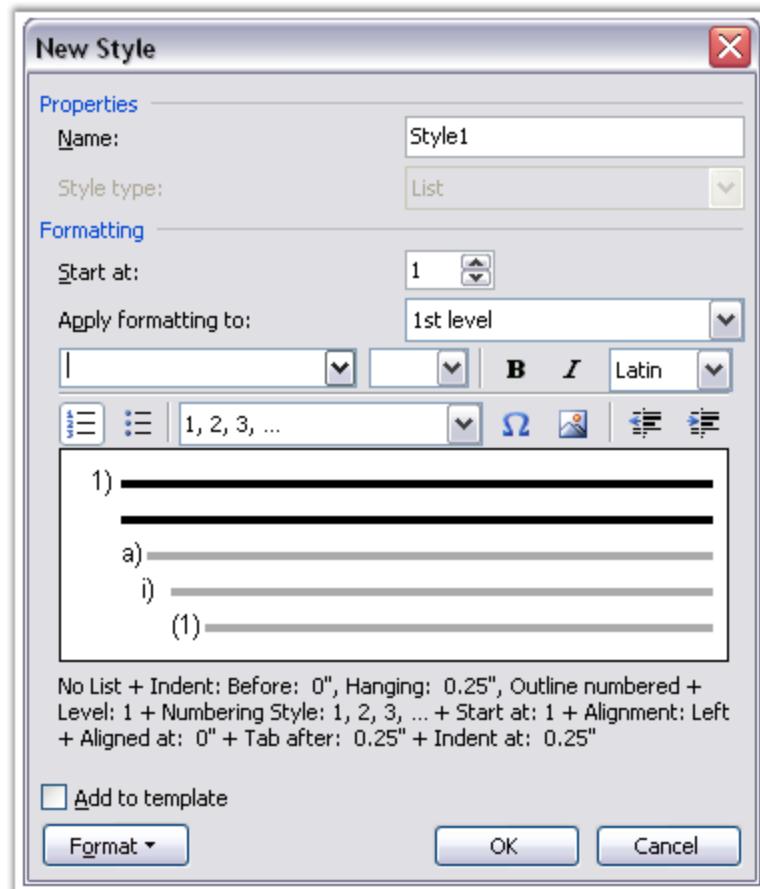


Figure 74: Create New Custom List Style

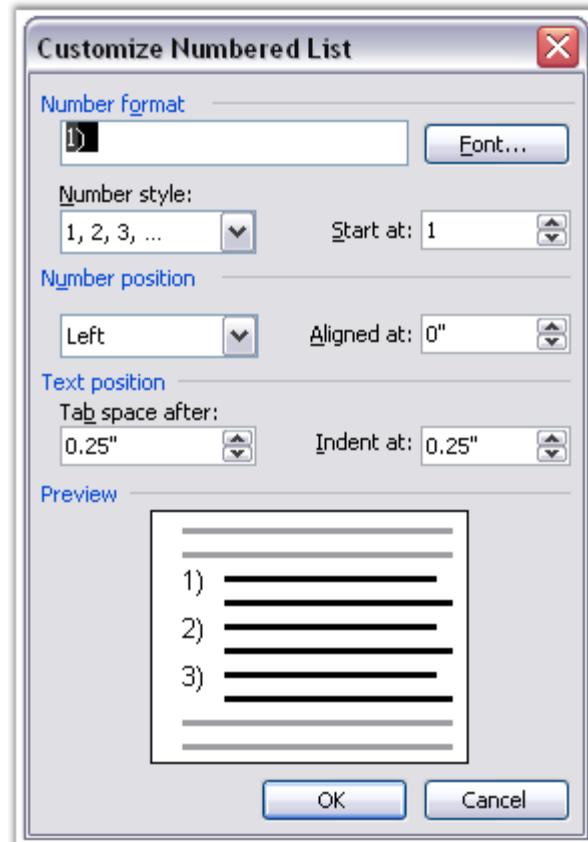


Figure 75: Numbered List Style

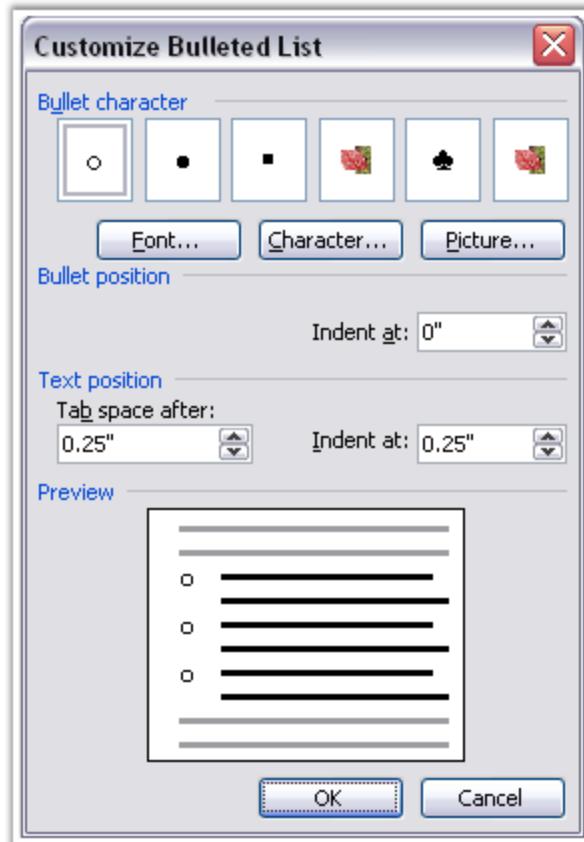


Figure 76: Bulleted List Style

Public Properties

Name	Description
Levels	Gets list levels collection.
ListType	Gets or sets list type.
Name	Gets style name.
StyleType	Gets the type of the style.
ListStyle.ListStyle (IWordDocument, ListType)	Initializes a new instance of the ListStyle class.

Public Methods

Name	Description

Clone	Clones current style object.
CreateEmptyListStyle	Static method. Creates empty list style.

WListLevel class represents list level in the Word document. By using the WListLevel class, you can customize the list level options.

Public Constructor

Name	Description
WListLevel.WListLevel (ListStyle)	Initializes new instance of WListLevel class.

Public Methods

Name	Description
Clone	Clones this instance.
GetListItemText	Gets list symbol for specified item index.

Public Properties

Name	Description
BulletCharacter	Gets or sets bullet pattern.
CharacterFormat	Gets or sets character formats of list symbol.
FollowCharacter	Gets or sets the type of character following the number text for the paragraph.
IsLegalStyleNumbering	Gets or sets ArabicNumberFormat property (true if the level turns all inherited numbers to arabic, false if it preserves their number format code).
NoRestartByHigher	True if the level's number sequence is not restarted by higher (more significant) levels in the list.
NumberAlignment	Gets or sets alignment (left, right, or centered) of the paragraph number.

NumberPosition	Gets or sets the number / bullet position for current listlevel (in points).
NumberPrefix	Gets or sets prefix pattern for numbered level.
NumberSuffix	Gets or sets suffix pattern for numbered level.
ParagraphFormat	Gets or sets paragraph format of list level.
PatternType	Gets or sets list numbering type.
StartAt	Gets or sets start at value.
TabSpaceAfter	Gets or sets spacing after list level's number or bullet (tab position if follow character is tab).
TextPosition	Gets or sets the left listlevel indent (in points).
UsePrevLevelPattern	When true, number generated will include previous levels (used for legal numbering).

The following example illustrates how to create user-defined list styles and apply it to the paragraph.

[C#]

```
//User bullet list style
ListStyle bulStyle = doc.AddListStyle(ListType.Bulleted, "BulletStyle");
WListLevel bulLevel1 = bulStyle.Levels[0];
bulLevel1.FollowCharacter = FollowCharacterType.Space;
bulLevel1.TextPosition = 40f;
bulLevel1.NumberAlignment = ListNumberAlignment.Right;

WListLevel bulLevel2 = bulStyle.Levels[1];
bulLevel2.FollowCharacter = FollowCharacterType.Space;
bulLevel2.TextPosition = 60f;
bulLevel2.NumberAlignment = ListNumberAlignment.Right;
bulLevel2.TabSpaceAfter = 40f;

paragraph = section.AddParagraph();
paragraph.AppendText("First bulleted ( level 0 )");
paragraph.ListFormat.ApplyStyle("BulletStyle");

paragraph = section.AddParagraph();
```

```
paragraph.AppendText("Level 1");
paragraph.ListFormat.ContinueListNumbering();
paragraph.ListFormat.IncreaseIndentLevel();

paragraph = section.AddParagraph();
paragraph.AppendText("Level 0");
paragraph.ListFormat.ContinueListNumbering();
paragraph.ListFormat.DecreaseIndentLevel();

//User numbered list style
ListStyle newStyle = doc.AddListStyle(ListType.Numbered, "NewStyle");
WListLevel listLevelNew = newStyle.Levels[0];
listLevelNew.FollowCharacter = FollowCharacterType.Tab;
listLevelNew.TextPosition = 80f;
listLevelNew.NumberAlignment = ListNumberAlignment.Right;
listLevelNew.TabSpaceAfter = 40f;
listLevelNew.StartAt = 2;
listLevelNew.NumberPrefix = ">>";
listLevelNew.NumberSufix = "<<";
listLevelNew.CharacterFormat.FontSize = 15;
listLevelNew.CharacterFormat.TextColor = Color.Blue;

WListLevel listLevelNew1 = newStyle.Levels[1];
listLevelNew1.IsLegalStyleNumbering = true;

WListLevel listLevelNew2 = newStyle.Levels[2];
listLevelNew1.NoRestartByHigher = true;

paragraph = section.AddParagraph();
paragraph.AppendText("First Numbered ( level 0 )");
paragraph.ListFormat.ApplyStyle("NewStyle");

paragraph = section.AddParagraph();
paragraph.AppendText("Level 1");
paragraph.ListFormat.ContinueListNumbering();
paragraph.ListFormat.IncreaseIndentLevel();

paragraph = section.AddParagraph();
paragraph.AppendText("Level 0");
paragraph.ListFormat.ContinueListNumbering();
paragraph.ListFormat.ListLevelNumber = 0;
```

[VB.NET]

```
'User bullet list style
Dim bulStyle As ListStyle = doc.AddListStyle(ListType.Bulleted,
"BulletStyle")
Dim bulLevel1 As WListLevel = bulStyle.Levels(0)
bulLevel1.FollowCharacter = FollowCharacterType.Space
bulLevel1.TextPosition = 40f
bulLevel1.NumberAlignment = ListNumberAlignment.Right

Dim bulLevel2 As WListLevel = bulStyle.Levels(1)
bulLevel2.FollowCharacter = FollowCharacterType.Space
bulLevel2.TextPosition = 60f
bulLevel2.NumberAlignment = ListNumberAlignment.Right
bulLevel2.TabSpaceAfter = 40f

paragraph = section.AddParagraph()
paragraph.AppendText("First bulleted ( level 0 )")
paragraph.ListFormat.ApplyStyle("BulletStyle")

paragraph = section.AddParagraph()
paragraph.AppendText("Level 1")
paragraph.ListFormat.ContinueListNumbering()
paragraph.ListFormat.IncreaseIndentLevel()

paragraph = section.AddParagraph()
paragraph.AppendText("Level 0")
paragraph.ListFormat.ContinueListNumbering()
paragraph.ListFormat.DecreaseIndentLevel()

'User numbered list style
Dim newStyle As ListStyle = doc.AddListStyle(ListType.Numbered, "NewStyle")
Dim listLevelNew As WListLevel = newStyle.Levels(0)
listLevelNew.FollowCharacter = FollowCharacterType.Tab
listLevelNew.TextPosition = 80f
listLevelNew.NumberAlignment = ListNumberAlignment.Right
listLevelNew.TabSpaceAfter = 40f
listLevelNew.StartAt = 2
listLevelNew.NumberPrefix = ">>"
listLevelNew.NumberSufix = "<<"
listLevelNew.CharacterFormat.FontSize = 15
listLevelNew.CharacterFormat.TextColor = Color.Blue

listLevelNew1 As WListLevel = newStyle.Levels(1)
listLevelNew1.IsLegalStyleNumbering = True

listLevelNew2 As WListLevel = newStyle.Levels(2)
```

```

listLevelNew1.NoRestartByHigher = True

paragraph = section.AddParagraph()
paragraph.AppendText("First Numbered ( level 0 )")
paragraph.ListFormat.ApplyStyle("NewStyle")

paragraph = section.AddParagraph()
paragraph.AppendText("Level 1")
paragraph.ListFormat.ContinueListNumbering()
paragraph.ListFormat.IncreaseIndentLevel()

paragraph = section.AddParagraph()
paragraph.AppendText("Level 0")
paragraph.ListFormat.ContinueListNumbering()
paragraph.ListFormat.ListLevelNumber = 0

```

4.4.2.5.1 List Format

WListFormat class defines the formatting for DocIO list paragraph. The type of the list is specified by using the ListType property of WListFormat. ListLevelNumber property defines the level number for the list paragraph. CurrentListStyle property defines the list style, applied for the current list paragraph. CurrentListLevel property returns the object of the WListLevel type, which defines the formatting for the list level (paragraph). For example, a value that the list starts at (for numbered lists), list symbols, alignment of list text, and so forth.

- To apply default bullet or numbered style to the paragraph, use **ApplyDefBulletStyle** or **ApplyDefNumberedStyle** function.
- To apply custom style, use **ApplyStyle** function.
- **ContinueListNumbering** function is used to continue previous list numbering.
- Use **IncreaseIndentLevel** or **DecreaseIndentLevel** to increase or decrease indent for the level.
- To remove list from the paragraph use **RemoveList** function.

Class Hierarchy

FormatBase

|

WListFormat

Public Constructor

Name	Description
------	-------------

WListFormat.WListFormat (IWParagraph)	Initializes new instance of WListFormat class.
---------------------------------------	--

Public Properties

Name	Description
CurrentListLevel	Gets or sets paragraph's ListLevel.
CurrentListStyle	Gets paragraph's list style.
CustomStyleName	Gets or sets name of custom style.
ListLevelNumber	Gets or sets list nesting level.
ListType	Gets or sets type of the list.
RestartNumbering	Gets or sets whether numbering of the list must restart from previous list.

Public Methods

Name	Description
ApplyDefBulletStyle	Applies default bullet style for current paragraph.
ApplyDefNumberedStyle	Applies default numbered style for current paragraph.
ApplyStyle	Gets or sets name of custom style.
ContinueListNumbering	Continues last list.
DecreaseIndentLevel	Decreases level indent.
IncreaseIndentLevel	Increases level indent.
RemoveList	Removes the list from current paragraph.

The following example illustrates how to use the WListFormat and list styles in DocIO.

[C#]

```
//Write default numbered list
```

```
IWParagraph paragraph = section.AddParagraph();
paragraph.AppendText( "First Numbered ( level 0 )" );
paragraph.ListFormat.ApplyDefNumberedStyle();

paragraph = section.AddParagraph();
paragraph.AppendText( "Level 1" );
paragraph.ListFormat.ContinueListNumbering();
paragraph.ListFormat.IncreaseIndentLevel();

paragraph = section.AddParagraph();
paragraph.AppendText( "Level 0" );
paragraph.ListFormat.ContinueListNumbering();
paragraph.ListFormat.DecreaseIndentLevel();

section.AddParagraph();
section.AddParagraph();

//Write default bulleted list
paragraph = section.AddParagraph();
paragraph.AppendText( "First bulleted ( level 0 )" );
paragraph.ListFormat.ApplyDefBulletStyle();

paragraph = section.AddParagraph();
paragraph.AppendText( "Level 1" );
paragraph.ListFormat.ContinueListNumbering();
paragraph.ListFormat.IncreaseIndentLevel();

paragraph = section.AddParagraph();
paragraph.AppendText( "Level 0" );
paragraph.ListFormat.ContinueListNumbering();
paragraph.ListFormat.DecreaseIndentLevel();

section.AddParagraph();
section.AddParagraph();

//Write mixed bulleted and numbered list
ListStyle myStyle = doc.AddListStyle( ListType.Numbered, "UserStyle" );
WListLevel listLevel1 = myStyle.Levels[ 0 ];
listLevel1.FollowCharacter = FollowCharacterType.Tab;
listLevel1.TextPosition = 80f;
listLevel1.NumberAlignment = ListNumberAlignment.Right;
listLevel1.TabSpaceAfter = 40f;
listLevel1.StartAt = 3;
listLevel1.NumberPrefix = "(((";
listLevel1.NumberSufix = "***.";
```

```

paragraph = section.AddParagraph();
paragraph.AppendText( "First numbered" );
paragraph.ListFormat.ApplyStyle( "UserStyle" );

paragraph = section.AddParagraph();
ListStyle bulletStyle = doc.AddListStyle( ListType.Bulleted, "UserStyle1" );
WListLevel level = bulletStyle.Levels[ 0 ];
level.NumberPosition = 30f;
level.TabSpaceAfter = 15f;
level.FollowCharacter = FollowCharacterType.Tab;

paragraph.AppendText( "First bullet" );
paragraph.ListFormat.ApplyStyle( "UserStyle1" );

paragraph = section.AddParagraph();
paragraph.AppendText( "Bulleted level 1" );
paragraph.ListFormat.ContinueListNumbering();

paragraph = section.AddParagraph();
paragraph.AppendText( "Numbered level 0 again" );
paragraph.ListFormat.ApplyStyle( "UserStyle" );
section.AddParagraph();
section.AddParagraph();

```

[VB.NET]

```

'Write default numbered list
Dim paragraph As IWParagraph = section.AddParagraph()
paragraph.AppendText("First Numbered ( level 0 )")
paragraph.ListFormat.ApplyDefNumberedStyle()

paragraph = section.AddParagraph()
paragraph.AppendText("Level 1")
paragraph.ListFormat.ContinueListNumbering()
paragraph.ListFormat.IncreaseIndentLevel()

paragraph = section.AddParagraph()
paragraph.AppendText("Level 0")
paragraph.ListFormat.ContinueListNumbering()
paragraph.ListFormat.DecreaseIndentLevel()

section.AddParagraph()
section.AddParagraph()

'Write default bulleted list

```

```
paragraph = section.AddParagraph()
paragraph.AppendText("First bulleted ( level 0 )")
paragraph.ListFormat.ApplyDefBulletStyle()

paragraph = section.AddParagraph()
paragraph.AppendText("Level 1")
paragraph.ListFormat.ContinueListNumbering()
paragraph.ListFormat.IncreaseIndentLevel()

paragraph = section.AddParagraph()
paragraph.AppendText("Level 0")
paragraph.ListFormat.ContinueListNumbering()
paragraph.ListFormat.DecreaseIndentLevel()

section.AddParagraph()
section.AddParagraph()

'Write mixed bulleted and numbered list
Dim myStyle As ListStyle = doc.AddListStyle(ListType.Numbered, "UserStyle")
Dim listLevel1 As WListLevel = myStyle.Levels(0)
listLevel1.FollowCharacter = FollowCharacterType.Tab
listLevel1.TextPosition = 80f
listLevel1.NumberAlignment = ListNumberAlignment.Right
listLevel1.TabSpaceAfter = 40f
listLevel1.StartAt = 3
listLevel1.NumberPrefix = "((("
listLevel1.NumberSufix = "***."

paragraph = section.AddParagraph()
paragraph.AppendText("First numbered")
paragraph.ListFormat.ApplyStyle("UserStyle")

paragraph = section.AddParagraph()
Dim bulletStyle As ListStyle = doc.AddListStyle(ListType.Bulleted,
"UserStyle1")
Dim level As WListLevel = bulletStyle.Levels(0)
level.NumberPosition = 30f
level.TabSpaceAfter = 15f
level.FollowCharacter = FollowCharacterType.Tab

paragraph.AppendText("First bullet")
paragraph.ListFormat.ApplyStyle("UserStyle1")

paragraph = section.AddParagraph()
paragraph.AppendText("Bulleted level 1")
paragraph.ListFormat.ContinueListNumbering()
```

```
paragraph = section.AddParagraph()
paragraph.AppendText("Numbered level 0 again")
paragraph.ListFormat.ApplyStyle("UserStyle")
section.AddParagraph()
section.AddParagraph()
```

4.4.2.6 Text Box Format

WTextBoxFormat class defines formatting for the Text Box.

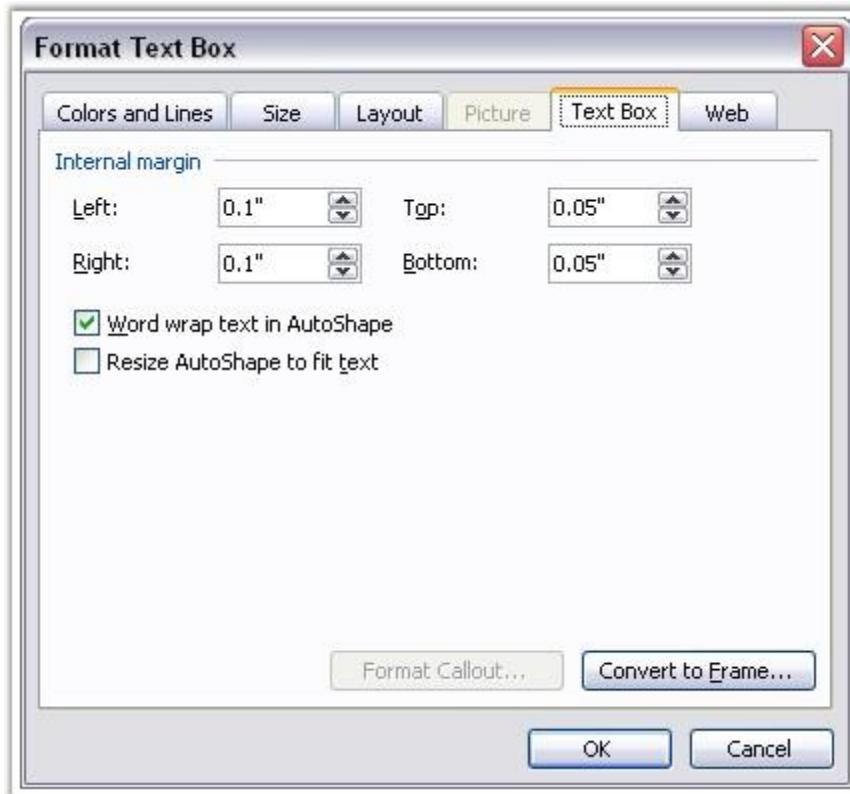


Figure 77: Format Text Box Dialog Box

Position

Absolute positioning of text box is defined by using the `VerticalPosition` and `HorizontalPosition` properties. Measure unit is point. Relative positioning is defined by using the `HorizontalAlignment` and `VerticalAlignment` properties.

HorizontalAlignment returns an object of type, ShapeHorizontalAlignment. The following are the variants for setting the Horizontal alignment of a picture.

- **None**: no horizontal alignment
- **Left**: left horizontal alignment
- **Center**: center horizontal alignment
- **Right**: right horizontal alignment
- **Inside**: inside horizontal alignment
- **Outside**: outside horizontal alignment

VerticalAlignment returns an object of type, ShapeVerticalAlignment. The following are the variants for setting the Vertical alignment of a picture.

- **Bottom**: picture is aligned to the bottom of the reference origin
- **Center**: picture is centered relative to the reference origin
- **Inline**: inline vertical alignment
- **Inside**: inside vertical alignment
- **None**: picture is explicitly positioned by using position properties
- **Outside**: outside vertical alignment
- **Top**: picture is aligned to the top of the reference origin

HorizontalOrigin and **VerticalOrigin** properties define the reference origin which is used for relative positioning of the picture.

HorizontalOrigin property returns a value of type, HorizontalOrigin. The following are the variants for setting the Horizontal origin of a picture.

- Margin
- Page
- Column
- Character

VerticalOrigin property returns a value of type, VerticalOrigin. The following are the variants for setting the Vertical origin of a picture.

- Margin
- Page
- Paragraph
- Line

Border Style

You can specify the style of the border line of the text box by using the **LineStyle** property. It provides the following options.

- Simple
- Double
- ThickThin
- ThinThick
- Triple

Class Hierarchy

FormatBase

|

WTextBoxFormat

Public Constructor

Name	Description
WTextBoxFormat.WTextBoxFormat(IWordDocument)	Initializes a new instance of the WTextBoxFormat class.

Public Properties

Name	Description
FillColor	Gets or sets fill color for textbox.
Height	Gets or sets the textbox height (in points).
HorizontalAlignment	Gets or sets horizontal alignment of textbox.
HorizontalOrigin	Gets or sets horizontal origin.
HorizontalPosition	Gets or sets the horizontal position of textbox (in points).
LineColor	Gets or sets line color.
LineDashing	Gets or sets line dashing for textbox.
LineStyle	Gets or sets linestyle of textbox.

LineWidth	Gets or sets the line width of textbox (in points).
NoLine	Gets or sets value which defines if there is a line around textbox shape.
TextWrappingStyle	Gets or sets text Wrapping style.
TextWrappingType	Gets or sets wrapping type for textbox.
VerticalAlignment	Gets or sets vertical alignment of textbox.
VerticalOrigin	Gets or sets vertical origin.
VerticalPosition	Gets or sets the textbox vertical position (in points).
Width	Gets or sets the textbox width (in points).

Public Methods

Name	Description
Clone	Clones textbox format.

The following example illustrates how to use the WTextBox and TextBoxFormat classes.

[C#]

```
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();
section.PageSetup.DifferentFirstPage = true;
section.PageSetup.DifferentOddAndEvenPages = true;

// Main doc textboxes
paragraph.AppendText("Testing textboxes");
// 1 textbox
IWTextBox mainTextbox = paragraph.AppendTextBox(150, 110);
mainTextbox.TextBoxBody.AddParagraph().AppendText("Textbox text 1");
mainTextbox.TextBoxFormat.FillColor = System.Drawing.Color.Blue;
mainTextbox.TextBoxFormat.LineDashing = LineDashing.LongDashDotDotGEL;
mainTextbox.TextBoxFormat.LineWidth = 4.0f;
// 2 textbox
IWTextBox mainTextbox1 = paragraph.AppendTextBox(150, 100);
```

```

mainTextbox1.TextBoxFormat.VerticalPosition = 500;
mainTextbox1.TextBoxBody.AddParagraph().AppendText("Another textbox");
mainTextbox1.TextBoxFormat.FillColor = System.Drawing.Color.Yellow;
mainTextbox1.TextBoxFormat.LineDashing = LineDashing.DashGEL;
mainTextbox1.TextBoxFormat.LineWidth = 3.75f;
mainTextbox1.TextBoxFormat.TextWrappingStyle = TextWrappingStyle.Through;
mainTextbox1.TextBoxFormat.TextWrappingType = TextWrappingType.Both;
mainTextbox1.TextBoxFormat.HorizontalAlignment =
ShapeHorizontalAlignment.Center;
mainTextbox1.TextBoxFormat.VerticalAlignment = ShapeVerticalAlignment.Bottom;

//Header/footer textboxes
paragraph = new WParagraph(doc);
paragraph.AppendText("Hello textboxes");
IWTextBox textbox = paragraph.AppendTextBox(20, 50);
textbox.TextBoxBody.AddParagraph().AppendText("Header textbox");
textbox.TextBoxFormat.FillColor = System.Drawing.Color.Blue;
textbox.TextBoxFormat.LineDashing = LineDashing.LongDashDotDotGEL;
textbox.TextBoxFormat.LineWidth = 4.0f;

IWTextBox textbox1 = paragraph.AppendTextBox(250, 50);
textbox1.TextBoxBody.AddParagraph().AppendText("Header textbox 2");
textbox1.TextBoxFormat.FillColor = System.Drawing.Color.Tomato;
textbox1.TextBoxFormat.VerticalPosition = 250;
textbox1.TextBoxFormat.LineStyle = TextBoxLineStyle.Triple;
textbox1.TextBoxFormat.LineDashing = LineDashing.LongDashGEL;
textbox1.TextBoxFormat.LineWidth = 6.0f;
textbox1.TextBoxFormat.NoLine = true;

section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph);

paragraph = new WParagraph(doc);
paragraph.AppendText("Hello footer textbox");
IWTextBox textbox2 = paragraph.AppendTextBox(120, 100);
textbox2.TextBoxFormat.VerticalPosition = 5;
textbox2.TextBoxBody.AddParagraph().AppendText("Footer textbox");
textbox2.TextBoxFormat.FillColor = System.Drawing.Color.Yellow;
textbox2.TextBoxFormat.LineDashing = LineDashing.DashGEL;
textbox2.TextBoxFormat.LineWidth = 3.75f;
textbox2.TextBoxFormat.TextWrappingStyle = TextWrappingStyle.Square;
textbox2.TextBoxFormat.HorizontalAlignment = ShapeHorizontalAlignment.Left;
textbox2.TextBoxFormat.VerticalAlignment = ShapeVerticalAlignment.Bottom;
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph);
doc.Save("TextBoxes.doc");

```

[VB]

```

Dim doc As IWordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph = section.AddParagraph()
section.PageSetup.DifferentFirstPage = True
section.PageSetup.DifferentOddAndEvenPages = True

' Main doc textboxes
paragraph.AppendText("Testing textboxes")

' 1 textbox
Dim mainTextbox As IWTextBox = paragraph.AppendTextBox(150, 110)
mainTextbox.TextBoxBody.AddParagraph().AppendText("Textbox text 1")
mainTextbox.TextBoxFormat.FillColor = System.Drawing.Color.Blue
mainTextbox.TextBoxFormat.LineDashing = LineDashing.LongDashDotDotGEL
    mainTextbox.TextBoxFormat.LineWidth = 4.0f

' 2 textbox
Dim mainTextbox1 As IWTextBox = paragraph.AppendTextBox(150, 100)
mainTextbox1.TextBoxFormat.VerticalPosition = 500
mainTextbox1.TextBoxBody.AddParagraph().AppendText("Another textbox")
mainTextbox1.TextBoxFormat.FillColor = System.Drawing.Color.Yellow
mainTextbox1.TextBoxFormat.LineDashing = LineDashing.DashGEL
mainTextbox1.TextBoxFormat.LineWidth = 3.75f
mainTextbox1.TextBoxFormat.TextWrappingStyle = TextWrappingStyle.Through
mainTextbox1.TextBoxFormat.TextWrappingType = TextWrappingType.Both
mainTextbox1.TextBoxFormat.HorizontalAlignment =
ShapeHorizontalAlignment.Center
mainTextbox1.TextBoxFormat.VerticalAlignment = ShapeVerticalAlignment.Bottom

'Header/footer textboxes
paragraph = New WParagraph(doc)
paragraph.AppendText("Hello textboxes")
Dim textbox As IWTextBox = paragraph.AppendTextBox(20, 50)
textbox.TextBoxBody.AddParagraph().AppendText("Header textbox")
textbox.TextBoxFormat.FillColor = System.Drawing.Color.Blue
textbox.TextBoxFormat.LineDashing = LineDashing.LongDashDotDotGEL
textbox.TextBoxFormat.LineWidth = 4.0f

Dim textbox1 As IWTextBox = paragraph.AppendTextBox(250, 50)
textbox1.TextBoxBody.AddParagraph().AppendText("Header textbox 2")
textbox1.TextBoxFormat.FillColor = System.Drawing.Color.Tomato
textbox1.TextBoxFormat.VerticalPosition = 250
textbox1.TextBoxFormat.LineStyle = TextBoxLineStyle.Triple
textbox1.TextBoxFormat.LineDashing = LineDashing.LongDashGEL

```

```

textbox1.TextBoxFormat.LineWidth = 6.0f
textbox1.TextBoxFormat.NoLine = True

section.HeadersFooters.FirstPageHeader.Paragraphs.Add(paragraph)

paragraph = New WParagraph(doc)
paragraph.AppendText("Hello footer textbox")
Dim textbox2 As IWTextBox = paragraph.AppendTextBox(120, 100)
Private textbox2.TextBoxFormat.VerticalPosition = 5
textbox2.TextBoxBody.AddParagraph().AppendText("Footer textbox")
textbox2.TextBoxFormat.FillColor = System.Drawing.Color.Yellow
textbox2.TextBoxFormat.LineDashing = LineDashing.DashGEL
textbox2.TextBoxFormat.LineWidth = 3.75f
textbox2.TextBoxFormat.TextWrappingStyle = TextWrappingStyle.Square
textbox2.TextBoxFormat.HorizontalAlignment = ShapeHorizontalAlignment.Left
textbox2.TextBoxFormat.VerticalAlignment = ShapeVerticalAlignment.Bottom
section.HeadersFooters.FirstPageFooter.Paragraphs.Add(paragraph)
doc.Save("TextBoxes.doc")

```

4.4.2.7 Importing XHTML

Essential DocIO supports inserting XHTML formatted strings in the Word document. There are two different usage scenarios namely:

- Insert HTML Document
- Insert HTML Formatted String

Insert HTML Document

This option enables to insert a whole HTML document with the following limitations:

- XHTML 1.0 Strict is preferred; XHTML 1.0 Transitional is also acceptable.
- There is an option to validate against either XHTML Strict or Transitional schema. By default the given html string is validated against XHTML 1.0 Transitional schema and an exception is thrown, if the html is found to be non-complaint. However, you can set this property on the document object to either, validate against XHTML Transitional schema or, Strict schema.
- If a block element is not supported, then its style would still be parsed and applied to the supported child elements inside.
- All the limitations are documented in Appendix 1 and 2.

The following code illustrates inserting a HTML string into a section of the Word document.

[C#]

```
//Inserting some XHTML
section.Body.InsertXHTML(HtmlString);
```

The **InsertXHTML** method has two overloads that specify the start paragraph item index and the end paragraph item index. The inserted XHTML adds several Word paragraphs, internally (with several possible paragraph items like lists and so on), and tables inside the Textbody of the Word document (Textbody is the container for paragraphs and tables in the Word object model).

It is also possible to check if the given XHTML string is valid or not by using the **WTextBody.IsValidXHTML** method. Refer to Appendix 1 and 2 for the list of supported tags and attributes.

Also, you can open the HTML file directly from the Word document constructor, and convert the HTML file into a Word document.

The following code illustrates opening a **XHTML** document and saving it as a Word document by using DocIO.

[C#]

```
WordDocument document = new WordDocument(String filename, FormatType
formattype, XHTMLValidationType XHTMLvalidationtype);
document.Save("sample.doc");
```

Support for partial path of an image:

Currently EssentialDocIO provides support for **the** partial path of an image only **when directly loading the HTML file into the Word document** using **document.Open()** method.

The following are the two overloaded methods.

- Document.open(string fileName, FormatType formatType, XHTMLValidationType validationType, string baseUrl)
- Document.open(Stream stream, FormatType formatType, XHTMLValidationType validationType, string baseUrl)

The following code illustrates loading the HTML file containing the partial path of an image.

[C#]

```
//Creating a new instance for the Word document
WordDocument document = new WordDocument();
//Setting the base folder path
string basePath=@"C:\InputFolder\";
//Opening the html file along with the base path of the html file
document.Open(Path.Combine(basePath, "Input.html"),
Syncfusion.DocIO.FormatType.Automatic, XHTMLValidationType.None, basePath);
//Saving the document
document.Save("sample.doc");
```

Insert Formatted Text Snippet

This option enables to insert XHTML formatted text inside Word paragraphs with the following limitations:

- The content will be placed inside a `<p>` tag, to validate against the XHTML schemas as explained before.
- This html snippet cannot contain any block elements like div, and so on, and will result in an exception being thrown otherwise. The only exception to this case is a single `<p>` tag.
- Among the supported XHTML tags, only the inline tags can be used for formatting text.

The following code illustrates appending a HTML formatted string into a paragraph.

[C#]

```
//Adding a new paragraph to the section.
IWParagraph paragraph = section.AddParagraph();
paragraph.AppendXHTML(htmlstring);
```

The following references enable to validate the HTML string for XHTML compliance.

- <http://www.w3schools.com/tags/default.asp>
- <http://validator.w3.org/check>

Appendix 1 – Imported HTML Tags

HTML Element	Style Attributes specific to this element (In addition to standard style attributes)	Non-Style Attributes supported (These attributes are applied directly to the element)	Limitations

HTML Element	Style Attributes specific to this element (In addition to standard style attributes)	Non-Style Attributes supported (These attributes are applied directly to the element)	Limitations
b	-	-	-
big	-	-	-
br	-	clear	-
sub	-	-	-
sup	-	-	-
em	-	-	-
font	-	face size color	The attribute value <i>normal</i> and <i>bigger</i> are not supported.
h1,h2,h3,h4,h5,h6	-	-	-
i	-	-	-
li	-	-	-
p	-	align	-
pre	-	-	Font style will be changed as courier. Space cannot be inserted.
s	-	-	-
small	-	-	-
strike	-	-	-
strong	-	-	-
table		border border-color cellpadding	-

HTML Element	Style Attributes specific to this element (In addition to standard style attributes)	Non-Style Attributes supported (These attributes are applied directly to the element)	Limitations
		cellspacing background-color bgcolor align width	
td	width, vertical-align	width rowspan colspan align valign	-
th	width, vertical-align	width rowspan colspan align valign	-
tr	-	height align	-
tt	-	-	-
u	-	-	-
ul	-	-	-.
img	-	height width src align	Currently Partial Image path is not supported.
div	-	-	-

HTML Element	Style Attributes specific to this element (In addition to standard style attributes)	Non-Style Attributes supported (These attributes are applied directly to the element)	Limitations
blockquote	-	-	-
a	-	href target id name	-
tbody	-	-	-
thead	-	-	-
tfoot	-	-	-
style	-	-	-
body	-	Vlink link	-

Appendix 2 - Standard Style Attributes

Standard Style Attributes (These style attributes are supported by all supported XHTML elements)	Limitations
font-family	-
font-size	-
font-style	-
font-weight	Different levels of bold are not supported as Word does n't support bold level.
text-align	-
text-decoration	Only underline and strike through are supported.
color	In the body and font, elements and foreground color can be set using either hexidecimal format (#RRGGBB) or one of 16 predefined case-

Standard Style Attributes (These style attributes are supported by all supported XHTML elements)	Limitations
	sensitive color names. In the color property of the style attribute, foreground color can be set using hexadecimal format (#RRGGBB).
line-height or height	-
background-color	-
margin	-
margin-left, margin-bottom, margin-top, margin-right	-
text-indent	-
border-bottom, border-top, border-right, border-left	The border color has to be specified in hexadecimal format (#RRGGBB).
border-color	The border color has to be specified in hexadecimal format (#RRGGBB).
border-left-color, border-right-color, border-top-color, border-bottom-color	The border color has to be specified in hexadecimal format (#RRGGBB).
border-width	-
border-left-width, border-right-width, border-bottom-width, border-top-width	-
border-style	-
border-left-style, border-right-style, border-top-style, border-bottom-style	-
page-break-before	-
page-break-after	-
padding	-
padding-left	-

Standard Style Attributes (These style attributes are supported by all supported XHTML elements)	Limitations
padding-right	-
padding-bottom	-
padding-top	-
line-height	-
letter-spacing	-
text-transform	-
white-space	-
list-style-image	-
list-style-type	-
outline	-
outline-style	-
outline-width	-
outline-color	-



Notes: Currently inserting XHTML formatted string in the Word document is not supported in Silverlight application.

4.5 Find and Replace

Find and replace text in a Word document is an essential feature in MS Word. This feature may often prove to be extremely helpful in the translator's work. Essential DocIO provides support to find a string, document element, or a bookmark in a Word document, and replace it with other document elements. This enables you to work more easily and efficiently with Word documents and reports.

You need to understand how TextSelection, TextBodySelection and TextBodyPart classes work, to understand this feature in DocIO. In short, this feature enables to find specific text in a document and replace it with new text.

This section describes the rich support of Essential DocIO for finding and replacing strings and document elements.

For More Information Refer:

[Text Selection](#), [TextBody Part](#), [TextBody Selection](#), [Find](#), [Replace](#)

4.5.1 TextSelection

TextSelection represents selected text in the Word document with the following limitations.

- Selected Text must be complete (text selection does not represent split selections).
- Selected Text must be a single paragraph (text selection inside two or more paragraphs is ignored).

TextSelection uses the **Find** and **FindAll** methods to select text. For details, see Find.

Public Constructors

Name	Description
TextSelection.TextSelection(WParagraph, int, int)	Initializes a new instance of the TextSelection class.

Public Properties

Name	Description
Count	Gets the count of text chunks.
SelectedText	Gets the selected text.

Public Methods

Name	Description

GetAsOneRange	Gets as one range.
GetEnumerator	Returns an enumerator that iterates through a collection.
GetRanges	Gets the ranges.
this[int]	Gets the System.String at the specified index.



Note: *TextSelection and GetAsOneRange should not be used for text replacement.*

The following example illustrates how to use the TextSelection class.

[C#]

```
docTemplate.Open( FINDTEMPLATE );
TextSelection rangesHolder1 = docSource1.Find( "The PlaceHolder1", false,
false );
```

[VB .NET]

```
docTemplate.Open(FINDTEMPLATE)
Dim rangesHolder1 As TextSelection = docSource1.Find("The PlaceHolder1",
False, False)
```

4.5.2 TextBodyPart

TextBodyPart class contains the collection of body items (it means that TextBodyPart can contain paragraphs, tables and even sections). TextBodyPart is usually used with the Bookmark Navigator.



Note: *TextBodyPart contains the copy of objects from the documents (paragraph(s), table(s), section(s), etc). So if you modify the content of the TextBodyPart, it does not affect the objects inside the document.*

Public Constructors

Name	Description
TextBodyPart.TextBodyPart ()	Initializes a new instance of the TextBodyPart class.

TextBodyPart.TextBodyPart (TextBodySelection)	Initializes a new instance of the TextBodyPart class.
TextBodyPart.TextBodyPart (TextSelection)	Initializes a new instance of the TextBodyPart class.
TextBodyPart.TextBodyPart (WordDocument)	Initializes a new instance of the TextBodyPart class.

Public Properties

Name	Description
BodyItems	Gets the body items.

Public Methods

Name	Description
Clear	Clears this instance.
Copy	Copies the specified item.
PasteAfter	Pastes ParagraphItem or TextBodyItem after specified item.
PasteAt	Pastes ITextBody at specified position.
PasteAtEnd	Pastes ITextBody at end of textbody.

The following example illustrates how to use the TextBodyPart class.

[C#]

```
WordDocument doc = new WordDocument( "sample.doc" );

BookmarksNavigator bn = new BookmarksNavigator(doc);
bn.MoveToBookmark("bookmark1");
TextBodyPart bodyPart = bn.GetBookmarkContent();
```

[VB.NET]

```
Dim doc As WordDocument = New WordDocument("sample.doc")
```

```
Dim bn As BookmarksNavigator = New BookmarksNavigator(doc)
bn.MoveToBookmark("bookmark1")
Dim bodyPart As TextBodyPart = bn.GetBookmarkContent()
```

4.5.3 TextBodySelection

TextBodySelection gives you an opportunity to select items in the TextBodyPart.

For example, you have the following text:

Text NEED COPY

THIS TEXT Other text

You may want to copy the "NEED COPY" table and "THIS TEXT", and paste in another location.

Objects Tree is as follows.

TextBody

- [0]Paragraph
 - [0]TextRange – "Text"
 - [1]TextRange – "NEED"
 - [2]TextRange – "COPY"
- [1]Table
- [2]Paragraph
 - [0]TextRange – "THIS"
 - [1]TextRange – "TEXT"
 - [2]TextRange – "Other Text"

The following code is used for this purpose.

[C#]

```
TextBodySelection bodySelection = new TextBodySelection( body, 0, 2, 1, 1 );
```

where,

- 0, 2, - paragraph starting index and ending index
- 1, - paragraph item starting index in first paragraph
- 1 - paragraph item ending index in last paragraph

Public Constructors

Name	Description
TextBodySelection.TextBodySelection (ITextBody, int, int, int, int)	Initializes a new instance of the TextBodySelection class.
TextBodySelection.TextBodySelection (ParagraphItem, ParagraphItem)	Initializes a new instance of the TextBodySelection class.

Public Properties

Name	Description
ItemEndIndex	Gets or sets the end index of the text body item.
ItemStartIndex	Gets or sets the start index of the text body item.
ParagraphItemEndIndex	Gets or sets the end index of the paragraph item.
ParagraphItemStartIndex	Gets or sets the start index of the paragraph item.
TextBody	Gets the text body.

Copy and **Paste** methods of **TextBodyPart** of the **TextBodySelection** class are used to copy and paste the text and body element at any position in the document. The following code snippet illustrates this.

[C#]

```
// Create TextBodySelection and select the items of interest.
TextBodySelection textSel = new TextBodySelection(sec.Body, 0,
lastItemIndex, 0, lastPItemIndex);

// Create TextBodyPart and copy the selected items.
TextBodyPart replacePart = new TextBodyPart(doc);
replacePart.Copy(textSel);

// Paste the copied content at the end of the text body.
```

```
replacePart.PasteAt(txtbdy, itemIndex);
```

[VB]

```
' Create TextBodySelection and select the items of interest.
Dim textSel As TextBodySelection = New TextBodySelection(sec.Body, 0,
lastItemIndex, 0, lastPItemIndex)

' Create TextBodyPart and copy the selected items.
Dim replacePart As TextBodyPart = New TextBodyPart(doc)
replacePart.Copy(textSel)

' Paste the copied text at the end of the text body.
replacePart.PasteAt(txtbdy, itemIndex)
```

4.5.4 Find

Essential DocIO provides various methods to improve the flexibility of the Find feature in the Word document.

Find Method

Find method is used to find an entry with a specified text or regular expression in the Word document.

Following are the possible input parameters of the Find method.

- given string to find.
- **caseSensitive**: defines if replace is case sensitive. For example, if case sensitive is set to false, and you want to find "AA" string, then in such case strings like "aA" and "Aa" also will be returned (will fit the search conditions).
- **wholeWord**: if set to true, string given must be the whole word (not part of the word).

The following are the variants of the Find method.

- **TextSelection Find(string given, bool caseSensitive, bool wholeWord)** - finds and returns an entry of specified string along with formatting, taking into consideration case-sensitive and whole word options.

- **TextSelection Find(Regex pattern)** - finds and returns entry of specified regular expression along with formatting.
- **TextSelection[] FindAll(Regex pattern)** - finds and returns all entries of specified regular expression along with formatting.
- **TextSelection[] FindAll(string given, bool caseSensitive, bool wholeWord)** - finds and returns all entries of specified string along with formatting, taking into consideration case-sensitive and whole word options.

The following example illustrates how to use the Find method.

[C#]

```
doc.Open( "sample.doc" );
TextSelection rangesHolder1 = doc.Find( "The PlaceHolder1", false, false );
```

[VB .NET]

```
doc.Open("sample.doc")
Dim rangesHolder1 As TextSelection = doc.Find("The PlaceHolder1", False,
False)
```

FindNext Method

You can find a particular string from a paragraph region or table by using the **FindNext** method. The following code illustrates this.

[C#]

```
//To find and replace particular table
IWTable table = doc.Sections[0].Tables[0];
TextSelection selc = doc.FindNext(table as TextBodyItem, "textAA", false,
false);
WTextRange range = selc.GetAsOneRange();
range.Text = "TextReplaced";

//or To find and replace from particular paragraph
IWTable table1 = doc.Sections[0].Tables[0];
IWParagraph par = table1.Rows[1].Cells[0].Paragraphs[0];
TextSelection sell = doc.FindNext(par as TextBodyItem, "ref AA", false,
false);
WTextRange range1 = sell.GetAsOneRange();
range1.Text = "New Text";
```

[VB.NET]

```
'To find and replace particular table
Dim table As IWTable = doc.Sections(0).Tables(0)
Dim selc As TextSelection = doc.FindNext(TryCast(table, TextBodyItem),
"textAA", False, False)
Dim range As WTextRange = selc.GetAsOneRange()
range.Text = "TextReplaced"

'or To find and replace from particular paragraph
Dim table1 As IWTable = doc.Sections(0).Tables(0)
Dim par As IWParagraph = table1.Rows(1).Cells(0).Paragraphs(0)
Dim sell As TextSelection = doc.FindNext(TryCast(par, TextBodyItem), "ref
AA", False, False)
Dim range1 As WTextRange = sell.GetAsOneRange()
range1.Text = "New Text"
```

Find with SingleLine mode

FindSingleLine method is used to find an entry in a document with a specified text or regular expression, including the newline or carriage return. This works in the same way as the SingleLine mode of .NET Regex. Note that the Find method will find the text only in a single paragraph without any newlines or carriage return considerations.

Name	Description
FindSingleLine(Regex)	Finds the first entry of specified pattern in single-line mode.
FindSingleLine(String, Boolean, Boolean)	Finds the first entry of given text in single-line mode.

It is also possible to find the string with SingleLine mode from a particular region by using the **FindNextSingleLine** method of the WordDocument class.

Name	Description
FindNextSingleLine(TextBodyItem, Regex)	Finds the next text which fit the specified pattern starting from start TextBodyItem using single-line mode.
FindNextSingleLine(TextBodyItem, String, Boolean, Boolean)	Finds the next given text starting from specified. TextBodyItem using single-line mode.

The following example illustrates how to find a string in the SingleLine mode.

[C#]

```
WordDocument doc = new WordDocument("Sample.doc");
string search = "\\[start\\](.*)\\[end\\]";

// The singleline option should cause \r\n to be included in .*
Regex expr = new Regex(search, RegexOptions.Singleline);

WTable table = doc.Sections[0].Tables[0] as WTable;
TextSelection[] sel = doc.FindNextSingleLine(table as TextBodyItem, expr);
```

[VB.NET]

```
Dim doc As New WordDocument("Sample.doc")
Dim search As String = "[start](.*)[end]"

' The singleline option should cause \r\n to be included in .*
Dim expr As New Regex(search, RegexOptions.Singleline)

Dim tab As WTable = TryCast(doc.Sections(0).Tables(0), WTable)
Dim sel As TextSelection() = doc.FindNextSingleLine(TryCast(tab,
TextBodyItem), expr)
```

4.5.5 Replace

Replace method provides support to replace text in the Word document. The following are the possible input parameters of the Replace method.

- **Pattern:** character pattern (object of Regex class)
- **Replace:** replace string
- **Given:** string to be replaced
- **CaseSensitive:** defines if replace is case sensitive.

For example if case sensitive is set to false and you want to replace "AA" string, then in such case strings like "aA" and "Aa" also will be replaced.

- **WholeWord:** if set to true, string given must be the whole word (not the part of the word)
- **SaveFormatting:** if set to true, it will preserve the existing place holder formatting

The following are the variants of the Replace method.

- **Replace(Regex pattern, string replace)**: replaces of all occurrences of a character pattern specified by a regular expression with replace string
- **Replace(string given, string replace, bool caseSensitive, bool wholeWord)**: replaces all entries of given string with replace string, taking into consideration caseSensitive and wholeWord options
- **Replace(Regex pattern, TextSelection textSelection)**: replaces all entries of given regular expression with TextRangesHolder (TextSelection)
- **Replace(string given, TextSelection textSelection, bool caseSensitive, bool wholeWord)**: replaces all entries of given string with TextSelection, taking into consideration caseSensitive and wholeWord options
- **Replace(Regex pattern, TextBodyPart bodyPart)**: replaces all entries of given regular expression with TextBodyPart
- **Replace(string given, TextBodyPart bodyPart, bool caseSensitive, bool wholeWord)**: replaces all entries of given string with TextBodyPart, taking into consideration caseSensitive and wholeWord options
- **Replace(string given, IWordDocument replaceDoc, bool caseSensitive, bool wholeWord, bool saveFormatting)**: replaces all entries of given string with given word document, taking into consideration caseSensitive, wholeWord options and formatting option (to preserve the formatting of the existing place holder or the new document place holder)
- **Replace(string given, TextBodyPart bodyPart, bool caseSensitive, bool wholeWord, bool saveFormatting)**: replaces all entries of given string with given TextBodyPart, taking into consideration caseSensitive, wholeWord options and formatting option
- **Replace(string given, TextSelection textSelection, bool caseSensitive, bool wholeWord, bool saveFormatting)**: replaces all entries of given string with TextSelection, taking into consideration caseSensitive, wholeWord options and formatting option

[C#]

Example 1:

```
//This sample replace specified regular expression with Picture
TextBodyPart textBodyPart = new TextBodyPart( doc );
Image image = Image.FromFile( ImagesPath + "Image.gif" );
WPicture pict = new WPicture( doc );
pict.LoadImage( image );

WParagraph para = new WParagraph( doc );
para.Items.Add( pict );
textBodyPart.BodyItems.Insert( 0, para );
doc.Replace( new Regex( "A" ), textBodyPart );
```

Example 2:

```

WordDocument docSource1 = new WordDocument();
docSource1.Open( DocumentsPath + FINDSOURCE1 );
WordDocument docSource2 = new WordDocument();
docSource2.Open( DocumentsPath + FINDSOURCE2 );
WordDocument docTemplate = new WordDocument();
docTemplate.Open( DocumentsPath + FINDTEMPLATE );

//Finds and returns entry of specified regular expression along with
//formatting
TextSelection rangesHolder1 = docSource1.Find( "The PlaceHolder1 was replaced
by
this sample Text.", false, false );

//Create new TextSelection object, with text and it's formatting specified by
//character range.In current sample character range is a paragraph's range of
//symbols from 1 to 4 position.
TextSelection rangesHolder2 = new TextSelection( docSource2.LastParagraph, 1,
4 );

docTemplate.Replace( new Regex( "PlaceHolder1" ), rangesHolder1 );
docTemplate.Replace( new Regex( "PlaceHolder2" ), rangesHolder2 );

```

[VB.NET]**Example 1:**

```

'This sample replace specified regular expression with Picture
Dim textBodyPart As TextBodyPart = New TextBodyPart(doc)
Dim image As Image = Image.FromFile(ImagesPath & "Image.gif")
Dim pict As WPicture = New WPicture(doc)
pict.LoadImage(image)

Dim para As WParagraph = New WParagraph(doc)
para.Items.Add(pict)
textBodyPart.BodyItems.Insert(0, para)
doc.Replace(New Regex("A"), textBodyPart)

```

Example 2:

```

Dim docSource1 As WordDocument = New WordDocument()
docSource1.Open(DocumentsPath + FINDSOURCE1)
Dim docSource2 As WordDocument = New WordDocument()
docSource2.Open(DocumentsPath + FINDSOURCE2)
Dim docTemplate As WordDocument = New WordDocument()
docTemplate.Open(DocumentsPath + FINDTEMPLATE)

```

```
'Finds and returns entry of specified regular expression along with
Formatting
Dim rangesHolder1 As TextSelection = docSource1.Find("The PlaceHolder1 was
replaced by this sample Text.", False, False)

'Create new TextSelection object, with text and it's formatting specified by
'character range.In current sample character range is a paragraph's range of
'symbols from 1 to 4 position.
Dim rangesHolder2 As TextSelection = New
TextSelection(docSource2.LastParagraph, 1, 4)

docTemplate.Replace(New Regex("PlaceHolder1"), rangesHolder1)
docTemplate.Replace(New Regex("PlaceHolder2"), rangesHolder2)
```

If you want to replace the first occurrence of a particular text alone, which appears more than once, set **doc.ReplaceFirst** property to **True**.

[C#]

```
doc.ReplaceFirst = true;
```

[VB.NET]

```
doc.ReplaceFirst = true;
```

Replace with SingleLine Mode

It is also possible to replace the string with .NET Regex **SingleLine** mode by using the **ReplaceSingleLine** method of DocIO. This enables the user to find the specified text of regular expression including the newline or carriage return, and replaces it with the given text.

The following table lists the overloads of this method.

Name	Description
ReplaceSingleLine(Regex, TextBodyPart)	Replaces the pattern with specified replacement in single-line mode.
ReplaceSingleLine(Regex, TextSelection)	Replaces the given pattern with replacement in single-line mode.

ReplaceSingleLine(Regex, String)	Replaces all entries with specified pattern with replace text in single-line mode.
ReplaceSingleLine(String, TextBodyPart, Boolean, Boolean)	Replaces the given text with specified replacement in single-line mode.
ReplaceSingleLine(String, TextSelection, Boolean, Boolean)	Replaces the given text with replacement in single-line mode.
ReplaceSingleLine(String, String, Boolean, Boolean)	Replaces all entries of given text with replace text in single-line mode.

The following code snippet illustrates the SingleLine mode replacement.

[C#]

```
WordDocument doc = new WordDocument("sample.doc");
string search = "\\|(.|\\r\\n|\\r|\\n)*?\\|";
// The singleline option should cause \\r\\n to be included in .*
Regex expr = new Regex(search, RegexOptions.Singleline);

doc.ReplaceSingleLine(expr, "test");
```

[VB.NET]

```
Dim doc As New WordDocument("sample.doc")
Dim search As String = "\|(.|" & vbCr & vbLf & "|" & vbCr & "|" & vbLf &
")*?\\|"

' The singleline option should cause \\r\\n to be included in .*
Dim expr As New Regex(search, RegexOptions.Singleline)

doc.ReplaceSingleLine(expr, "test")
```

4.6 Mail Merge

The mail merge function allows you to fill a template document with data from the data source. It is represented by the **MailMerge** class. The following data source types are used in mail merge: String Arrays, DataTable, DataReader or DataView class objects. The template of the document is created by using the MergeFields of MS Word.

To perform Mail Merge

1. Open the **Insert** menu, point to **Fields**, and then click **MergeField**.

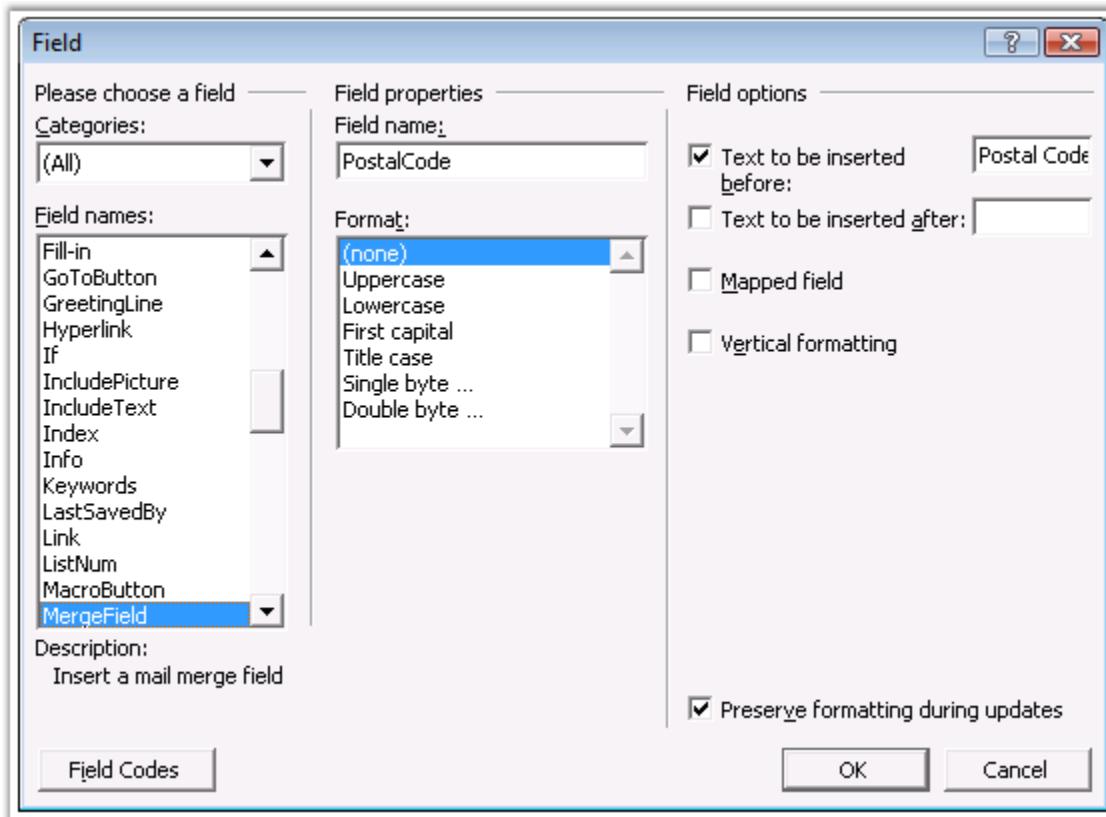


Figure 78: Field Dialog Box

2. Provide the merge field name, and the text to be inserted before and after the field. Note that the merge field name should match the field name of the data source.

```
*FirstName* *LastName*,  
*Company*,  
*Address*,  
*City* - *Country* |  
  
Phone : *Phone*  
Email : *Email*
```

Figure 79: Merge field name Provided

Mail merge operations are performed by the **Execute** or **ExecuteGroup** methods. There are several overloads of these methods for different data sources.

Execute

- **void Execute(string[] fieldNames, string[] fieldValues)**: performs replacements of every merge field in the document, in which field name matches one of the values from fieldNames string array, with the corresponding value from fieldValues string array.
- **void Execute(DataTable table)**: performs replacements of merge fields, in which field names match the table column names, with the corresponding values of table cell. These replacements are performed for every row contained in the table.

For example, consider you have a document containing a single page, with a MergeField on it, whose field name is "Country", with the words: "is a beautiful country". Also, you have a DataTable, "Geography", containing the following data.

Column	Continent	Country
1	Australia	Australia
2	North America	USA
3	Eurasia	France

After performing mail merge operations, you will have a document containing three pages with the following text.

1 page: Australia is a beautiful country.

2 page: USA is a beautiful country.

3 page: France is a beautiful country.

You can also use the **Next** field, if you want to display the values of some rows at the same place.

For example, if you want to enumerate all the countries contained in the table, you have to insert the Next field before every MergeField, with field name "Country" beginning with the second MergeField.

- **void Execute(DataRow row)**: works similarly to that with DataTable parameter for the only row
- **void Execute(DataView dataView)**: works similarly to that with DataTable parameter
- **void Execute(IDataReader dataReader)**: works similarly to that with DataTable parameter

ExecuteGroup

- **void ExecuteGroup(DataTable table)**: performs replacements of merge fields, in which field names match the table column names, with the corresponding values of table cell

These replacements are performed for every row contained in the table in the specified region. The region where the mail merge operations are to be performed must be marked by two MergeFields with the following names:

- **TableStart:TableName** – for the entry point of the region
- **TableEnd:TableName** – for the end point of the region

For example, consider you have a DataTable containing the same data as in previous example, and you want to get the document with following content:

Australia

USA

France

are beautiful countries.

You have to insert three MergeFields in the document with the following field names:

- **TableStart:Geography** – marks the beginning of mail merge region
- **Country** – will be replaced by values from table, "Geography"
- **TableEnd:Geography** – marks the end of mail merge region
- **void ExecuteGroup(DataView dataView)**: works similarly to that with DataTable parameter
- **void ExecuteGroup(IDataReader dataReader)**: works similarly to that with DataTable parameter

Public Methods

Name	Description
Execute	Executes mail merge.
ExecuteGroup	Executes mail merge for a group (region).
GetMergeFieldNames	Returns a collection of mergefield names found in

	the document.
ExecuteNestedGroup	Executes nested mail merge for a group(Region or tables)

The following example illustrates how to use the MailMerge class for different data sources.

[C#]

```
IWordDocument document = new WordDocument( "MailMergeDocument.doc" );

// Mail merge operations with string arrays data sources
string[] fieldNames = new string[]{ "Continent", "Country", "Region" };
string[] fieldValues = new string[]{ "Eurasia", "Germany", "Bavaria" };
document.MailMerge.Execute( fieldNames, fieldValues );

// Mail merge operations with DataTable data source containing data from
// database.
DataTable table = GetDataTable( "select * from Geography" );
document.MailMerge.Execute( table );

// Mail merge operations with DataView data source, created basing on the
// DataTable
DataView dataView = new DataView( GetDataTable( "select * from Geography" ) );
dataView.Sort = "CountryName ASC";
document.MailMerge.Execute( dataView );

// Mail merge operations with DataReader data source
IDataReader dataReader = GetDataReader( "select * from Geography" );
document.MailMerge.Execute( dataReader );
dataReader.Close();

// Mail merge operations for the specified region
DataTable table = GetDataTable( "select * from Employees" );
table.TableName = "Geography";
document.MailMerge.ExecuteGroup( table );

document.Save( "AfterMailMerge.doc" );
//or
document.Save("AfterMailMerge.docx", FormatType.Docx);
```

[VB .NET]

```

Dim document As IWordDocument = New WordDocument("MailMergeDocument.doc")

' Mail merge operations with string arrays data sources
Dim fieldNames As String() = New String() {"Continent", "Country", "Region"}
Dim fieldValues As String() = New String() {"Eurasia", "Germany", "Bavaria"}
document.MailMerge.Execute(fieldNames, fieldValues)

' Mail merge operations with DataTable data source containing data from
database.
Dim table As DataTable = GetDataTable("select * from Geography")
document.MailMerge.Execute(table)

' Mail merge operations with DataView data source, created basing on the
DataTable
Dim dataView As DataView = New DataView(GetDataTable("select * from Geography
"))
dataView.Sort = "CountryName ASC"
document.MailMerge.Execute(dataView)

' Mail merge operations with DataReader data source
Dim dataReader As IDataReader = GetDataReader("select * from Geography ")
document.MailMerge.Execute(dataReader)
dataReader.Close()

' Mail merge operations for the specified region
Dim table As DataTable = GetDataTable("select * from Employees")
table.TableName = "Geography"
document.MailMerge.ExecuteGroup(table)

document.Save("AfterMailMerge.doc")
'or
document.Save("AfterMailMerge.docx", FormatType.Docx)

```

It is also possible to conditionally format the merge fields dynamically by using the **MergeField** events.

See Also

[Nested Mail Merge](#), [Mail merge Events](#), [Additional Mail Merge Features](#)

4.6.1 Nested Mail Merge

Use **WordDocument.MailMerge.ExecuteNestedGroup** method to perform nested mail merge. Each item of "command" ArrayList must contain 0 or more DictionaryEntry objects. Each DictionaryEntry object must contain the Table Name (DataTable name) and String Command with command, which must be applied to the table. Use the following expression for getting the current value of specified column in the table - "%TableName.ColumnName%".

The following code snippet illustrates how a nested mail merge for a region or table is implemented by using DocIO.

[C#]

```
// Get Data from the Database.
OleDbConnection conn = new
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + DataBase);
conn.Open();

// ArrayList contains the list of commands
ArrayList commands = new ArrayList();

// DictionaryEntry contains "Source table" (KEY) and "Command" (VALUE)
DictionaryEntry entry = new DictionaryEntry("Employees", "Select TOP 10 * from Employees");
commands.Add(entry);

// To retrieve customer details
entry = new DictionaryEntry("Customers", "SELECT DISTINCT TOP 10 * FROM ((Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID) INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID) WHERE Employees.EmployeeID = %Employees.EmployeeID%");
commands.Add(entry);

// To retrieve order details
entry = new DictionaryEntry("Orders", "SELECT DISTINCT TOP 10 * FROM Orders WHERE Orders.CustomerID = '%Customers.CustomerID%' AND Orders.EmployeeID = %Employees.EmployeeID%");
commands.Add(entry);

//Execute Mail merge
doc.MailMerge.ExecuteNestedGroup(conn, commands);
```

[VB]

```
' Get Data from the Database.
Dim conn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + DataBase)
```

```

conn.Open()

' ArrayList contains the list of commands
Dim commands As New ArrayList()

' DictionaryEntry contains "Source table" (KEY) and "Command" (VALUE)
Dim entry As New DictionaryEntry("Employees", "Select TOP 10 * from Employees")
commands.Add(entry)

' To retrieve customer details
entry = New DictionaryEntry("Customers", "SELECT DISTINCT TOP 10 * FROM ((Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID) INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID) WHERE Employees.EmployeeID = %Employees.EmployeeID%")
commands.Add(entry)

' To retrieve order details
entry = New DictionaryEntry("Orders", "SELECT DISTINCT TOP 10 * FROM Orders WHERE Orders.CustomerID = '%Customers.CustomerID%' AND Orders.EmployeeID = %Employees.EmployeeID%")
commands.Add(entry)

'Execute Mail merge
doc.MailMerge.ExecuteNestedGroup(conn, commands)

```

**Note:**

- Nested mail merge for a region works when the group start and end is BeginGroup and EndGroup respectively.**
- Nested mail merge for a table works when the group start and end is StartTable and EndTable respectively.**

Public Methods

Name	Description
ExecuteNestedGroup(DataSet dataSet, ArrayList commands)	Executes nested mail merge within a Group, for the specified data from the Data Set.
ExecuteNestedGroup(DbConnection conn, ArrayList commands)	Executes nested mail merge within a Group, for the specified data from the Database.
ExecuteNestedGroup(DbConnection conn,	Executes nested mail merge within a Group, for

ArrayList commands, bool isSqlConnection)	the specified data from the Database. If it is an SQL connection, then set to True.
---	---

The following code snippet illustrates how to implement a nested mail merge for a region or table by using a Data Set.

[C#]

```
public void CreateDocument()
{
WordDocument doc = new WordDocument();
doc.Open( dataPath + "NestedMailMerge1.doc" );

// Get Data from the Database.
DataTable table1 = GetNestedDataTable( "select * from Table1" );
table1.TableName = "Table1";

DataTable table2 = GetNestedDataTable( "select * from Table2" );
table2.TableName = "Table2";

DataTable table3 = GetNestedDataTable( "select * from Table3" );
table3.TableName = "Table3";

DataSet dataSet = new DataSet( "Test dataset" );
dataSet.Tables.Add( table1 );
dataSet.Tables.Add( table2 );
dataSet.Tables.Add( table3 );

// ArrayList contains the list of commands.
ArrayList commands = new ArrayList();
DictionaryEntry entry = new DictionaryEntry( "Table1", string.Empty );
commands.Add( entry );

entry = new DictionaryEntry("table2", "sellerid = %table1.sellerid%");
commands.Add( entry );

entry = new DictionaryEntry("Table3", "CustomerID = %Table2.CustomerID%");
commands.Add( entry );

doc.MailMerge.RemoveEmptyParagraphs = true;

// Execute NestedGroup by passing the DataSet.
doc.MailMerge.ExecuteNestedGroup(dataSet, commands );
}
```

```

doc.Save( "NestedMailMerge.doc" );
System.Diagnostics.Process.Start( "NestedMailMerge.doc" );
}

/// <summary>
/// Get the table
/// </summary>
/// <param name="sqlQuery"></param>
/// <returns></returns>
private DataTable GetNestedDataTable( string sqlQuery )
{
    OleDbConnection conn = null;
    try
    {
        string connString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
System.IO.Path.Combine( dataPath, @"NestedMailMerge1.mdb" );

        // Get data from the database.
        conn = new OleDbConnection( connString );
        conn.Open();
        OleDbCommand cmd = new OleDbCommand( sqlQuery, conn );
        OleDbDataAdapter da = new OleDbDataAdapter( cmd );
        DataTable table = new DataTable();
        da.Fill( table );
        return table;
    }
    finally
    {
        if( conn != null )
            conn.Close();
    }
}
}

```

[VB.NET]

```

Public Sub CreateDocument()
    Dim doc As WordDocument = New WordDocument()
    doc.Open(dataPath + "NestedMailMerge1.doc")

    ' Get Data from the Database.
    Dim table1 As DataTable = GetNestedDataTable("select * from Table1")
    table1.TableName = "Table1"

    Dim table2 As DataTable = GetNestedDataTable("select * from Table2")
    table2.TableName = "Table2"

```

```

Dim table3 As DataTable = GetNestedDataTable("select * from Table3")
table3.TableName = "Table3"

Dim dataSet As DataSet = New DataSet("Test dataset")
dataSet.Tables.Add(table1)
dataSet.Tables.Add(table2)
dataSet.Tables.Add(table3)

' ArrayList contains the list of commands.
Dim commands As ArrayList = New ArrayList()
Dim enTry As DictionaryEntry = New DictionaryEntry("Table1", String.Empty)
commands.Add(enTry)

enTry = New dictionaryenTry("table2", "sellerid = Decimal.Remainder( ,
table1.sellerid)%")
commands.Add(enTry)

enTry = New DictionaryEnTry("Table3", "CustomerID = Decimal.Remainder( ,
Table2.CustomerID)%")
commands.Add(enTry)

doc.MailMerge.RemoveEmptyParagraphs = True

' Execute NestedGroup by passing the DataSet.
doc.MailMerge.ExecuteNestedGroup(dataSet, commands)

doc.Save("NestedMailMerge.doc")
System.Diagnostics.Process.Start("NestedMailMerge.doc")
End Sub

'<summary>
' Get the table
'</summary>
'<param name="sqlQuery"></param>
'<returns></returns>
Private Function GetNestedDataTable(ByVal sqlQuery As String) As DataTable
    Dim conn As OleDbConnection = Nothing
    Try
        Dim connString As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" + System.IO.Path.Combine(dataPath, "NestedMailMerge1.mdb")

        ' Get data from the database.
        conn = New OleDbConnection(connString)
        conn.Open()
        Dim cmd As OleDbCommand = New OleDbCommand(sqlQuery, conn)

```

```

Dim da As OleDbDataAdapter = New OleDbDataAdapter(cmd)
Dim table As DataTable = New DataTable()
da.Fill(table)
Return table
Finally
    If Not conn Is Nothing Then
        conn.Close()
    End If
End Try
End Function
}

```

4.6.2 Mail Merge Events

The MailMerge class provides two events to expand the mail merge capabilities.

MailMerge.MergeField and **MailMerge.MergeImageField** events are used to implement custom control logic over the mail merge process.

The MailMerge.MergeField event occurs during mail merge when a simple mail merge field is encountered in the document. This allows to implement further control over the mail merge and perform any actions when the event occurs. Use the **MergeFieldEventHandler** delegate to reference the method that will handle MergeField. This method should accept a MergeFieldEventArgs object that provides data for the MergeField event.

[C#]

```

public void MailMerge()
{
// Load the template.
WordDocument document = new WordDocument( @"Template.doc" );

// Using Merge events to do conditional formatting during runtime.
document.MailMerge.MergeField += new
MergeFieldEventHandler(AlternateRows_MergeField);

// Execute Mail Merge with groups.
document.MailMerge.Execute (GetDataTable());

// Save the document.
document.Save("sample.doc");
}

/// <summary>

```

```

/// Called for every merge field encountered in the document.
/// We can either return some data to the mail merge engine or do something
/// else with the document. In this case we modify cell formatting.
/// </summary>

private void AlternateRows_MergeField(object sender, MergeFieldEventArgs args)
{
    // Conditionally format data during Merge.
    if( args.RowIndex % 2 == 0 )
    {
        args.CharacterFormat.TextColor = Color.FromArgb( 255 , 102 ,0 );
    }
}

///<summary>
/// Create DataTable and fill it with data.
/// </summary>
private static DataTable GetDataTable()
{
    DataTable dataTable = new DataTable("Employee");
    dataTable.Columns.Add("EmpName");
    dataTable.Columns.Add("EmpNumber");
    for (int i = 0; i < 20; i++)
    {
        DataRow datarow = dataTable.NewRow();
        dataTable.Rows.Add(datarow);
        datarow[0] = "Emp" + i.ToString();
        datarow[1] = "Emp" + i.ToString();
    }
    return dataTable;
}

```

[VB.NET]

```

Public Sub MailMerge()

    ' Load the template.
    Dim document As WordDocument = New WordDocument("Template.doc")

    ' Using Merge events to do conditional formatting during runtime.
    AddHandler document.MailMerge.MergeField, AddressOf
AlternateRows_MergeField
    document.MailMerge.Execute (GetDataTable());

    ' Save the document.

```

```

        document.Save("sample.doc")
End Sub

' <summary>
' Called for every merge field encountered in the document.
' We can either return some data to the mail merge engine or do something
' else with the document. In this case we modify cell formatting.
' </summary>
Private Sub AlternateRows_MergeField(ByVal sender As Object, ByVal args As
MergeFieldEventArgs)

    ' Conditionally format data during Merge.
    If args.RowIndex Mod 2 = 0 Then
        args.CharacterFormat.TextColor = Color.FromArgb(255, 102, 0)
    End If
End Sub

'<summary>
' Create DataTable and fill it with data.
' </summary>
Private Shared Function GetDataTable() As DataTable
    Dim dataTable As DataTable = New DataTable("Employee")
    dataTable.Columns.Add("EmpName")
    dataTable.Columns.Add("EmpNumber")
    For i As Integer = 0 To 19
        Dim datarow As DataRow = dataTable.NewRow()
        dataTable.Rows.Add(datarow)
        datarow(0) = "Emp" & i.ToString()
        datarow(1) = "Emp" & i.ToString()
    Next i
    Return dataTable
End Function

```

MergeImageField Events

The MailMerge.MergeImageField occurs during mail merge when an image mail merge field is encountered in the document. Image mail merge field is a merge field, with format, Image:MyFieldName. You can respond to this event to return a file name, stream, or an Image object to the mail merge engine, so that it is inserted into the document.

Use the MergeImageFieldEventHandler delegate representing the method that will handle the MergeImageField event. The event handler receives an argument of type, MergeImageFieldEventArgs.

Comment2: Looks like repeated code.

[C#]

```
public void MailMerge()
{
    // Load the template.
    WordDocument document = new WordDocument( @"Template.doc" );

    // Using Merge events handler for image fields.
    document.MailMerge.MergeImageField += new
    MergeImageFieldEventHandler(MergeField_ProductImage);

    // Execute Mail Merge with groups.
    document.MailMerge.Execute (GetDataTable());

    //Save the document

    document.Save("sample.doc");

}

/// <summary>

/// This is called when mail merge engine encounters Image:XXX merge field in
the document.

/// You have a chance to return an Image object, file name or a stream that
contains the image.

/// </summary>
private void MergeField_ProductImage(object sender, MergeImageFieldEventArgs
args)
{
    // Get the image from disk during Merge.
    if (args.FieldName == "ProductImage")
    {
        string ProductFileName = args.FieldValue.ToString();
        args.Image = Image.FromFile (ProductFileName) ;
    }
}
```

```
//<summary>

/// Create DataTable and fill it with data.

/// </summary>

private static DataTable GetDataTable()

{

    DataTable dataTable = new DataTable("Employee");

    dataTable.Columns.Add("EmpName");

    dataTable.Columns.Add("EmpNumber");

    for (int i = 0; i < 20; i++)

    {

        DataRow datarow = dataTable.NewRow();

        dataTable.Rows.Add(datarow);

        datarow[0] = "Emp" + i.ToString();

        datarow[1] = "Emp" + i.ToString();

    }

    return dataTable;

}
```

[VB]

```
Public Sub MailMerge()

    ' Load the template.
    Dim document As WordDocument = New WordDocument("Template.doc")

    ' Execute Mail Merge with groups.
```

```

' Using Merge events handler for image fields. AddHandler
document.MailMerge.MergeImageField, AddressOf MergeField_ProductImage
document.MailMerge.Execute (GetDataTable());

'Save the document
document.Save("sample.doc")

End Sub

''' <summary>

''' This is called when mail merge engine encounters Image:XXX merge field in
the document.

''' You have a chance to return an Image object, file name or a stream that
contains the image.

''' </summary>
Private Sub MergeField_ProductImage(ByVal sender As Object, ByVal args As
MergeImageFieldEventArgs)

    ' Get the image from disk during Merge.
    If args.FieldName = "ProductImage" Then
        Dim ProductFileName As String = args.FieldValue.ToString()
        args.Image = Image.FromFile (ProductName)
    End If
End Sub

'''<summary>

''' Create DataTable and fill it with data.

''' </summary>

Private Shared Function GetDataTable() As DataTable

    Dim dataTable As DataTable = New DataTable("Employee")

    dataTable.Columns.Add("EmpName")

    dataTable.Columns.Add("EmpNumber")

    For i As Integer = 0 To 19

```

```

Dim datarow As DataRow = dataTable.NewRow()

dataTable.Rows.Add(datarow)

datarow(0) = "Emp" & i.ToString()

datarow(1) = "Emp" & i.ToString()

Next i

Return dataTable

End Function

```

4.6.3 Additional Mail Merge Features

Using Mapped Fields

The MailMerge class allows to automatically map between names of fields in your data source and names of mail merge fields in the document. To perform this, use the **MailMerge.MappedDataFields** property that returns a MappedDataFields object. MappedDataFields is a collection of string keys into string values. The keys are the names of mail merge fields in the document and the values are the names of fields in your data source. The class provides all properties and methods typical for a regular .NET collection such as Add, Clear, Remove, and so on.

The following example illustrates how to add a mapping when a merge field in a document and a data field in a data source have different names.

[C#]

```
doc.MailMerge.MappedDataFields.Add("FieldName_InDocument",
"FieldName_InDataSource");
```

[VB]

```
doc.MailMerge.MappedDataFields.Add("FieldName_InDocument",
"FieldName_InDataSource")
```

Obtaining Merge Field Names

You can get the collection of the merge field names available in the document by using the **MailMerge.GetFieldNames** method. This returns an array of string that contains the names.

The following example illustrates how to get the names of all the merge fields in a document.

[C#]

```
string[] fieldNames = doc.MailMerge.GetMergeFieldNames();
```

[VB]

```
Dim fieldNames As String() = doc.MailMerge.GetMergeFieldNames()
```

Obtaining Merge Field Group Names

You can get the collection of the Merge Field Group names available in the document by using the **MailMerge.GetMergeGroupNames** method. This returns an array of string that contains the names.

[C#]

```
string[] groupNames = doc.MailMerge.GetMergeGroupNames()
```

[VB]

```
Dim filednames As String() = doc.MailMerge.GetMergeFieldNames(groupName)
```

Obtaining Merge Fields for a Specific Group

You can get the collection of the Merge Fields for a specific group in the document by using the **doc.MailMerge.GetMergeFieldNames(String groupName)** method. This returns an array of string that contains the field names.

[C#]

```
string[] filednames = doc.MailMerge.GetMergeFieldNames(groupName);
```

[VB]

```
Dim filednames As String() = doc.MailMerge.GetMergeFieldNames(groupName)
```

Removing Empty Paragraphs

To remove paragraphs that contain empty mail merge fields from the document, set the **doc.MailMerge.RemoveEmptyParagraphs** to **True**.

The following example illustrates how to remove paragraphs that contain empty mail merge fields.

[C#]

```
doc.MailMerge.RemoveEmptyParagraphs = true;
```

[VB]

```
doc.MailMerge.RemoveEmptyParagraphs = True
```

Removing Empty Groups

To remove empty groups from the document during mail merge, set the **document.MailMerge.RemoveEmptyGroup** to **True**. The default value of RemoveEmptyGroup is false.

The following example illustrates how to remove empty groups from the document.

[C#]

```
document.MailMerge.RemoveEmptyGroup = true;
```

[VB]

```
document.MailMerge.RemoveEmptyGroup = True
```

Clear Fields

To remove empty mail merge fields from the document, set **MailMerge.ClearField** property to **True**.

[C#]

```
WordDocument doc = new WordDocument("Sample.doc");
string[] fieldname = { "FirstName", "LastName" };
string[] fieldvalues = { "John", "David" ,};
doc.MailMerge.ClearFields = false;
doc.MailMerge.Execute(fieldname, fieldvalues);
```

[VB.NET]

```
Dim doc As WordDocument = New WordDocument("Sample.doc")
Dim fieldname As String() = {"FirstName", "LastName"}
Dim fieldvalues As String() = {"John", "David"}
doc.MailMerge.ClearFields = false
doc.MailMerge.Execute(fieldname, fieldvalues)
```

4.7 Security

DocIO provides support to protect a Word document. Here protection restricts the access to the elements present within the document. In MS Word, a document is protected through the **Protect Document** option in the **Tools** menu.

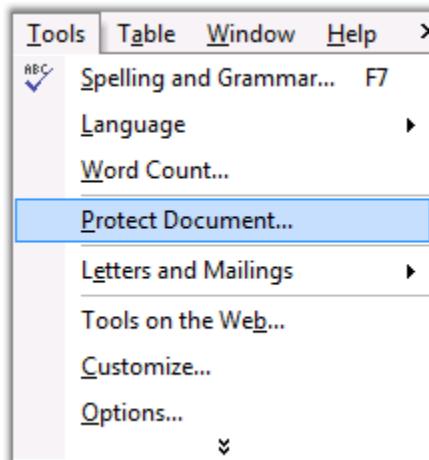


Figure 80: Protect Document option in the Tools Menu

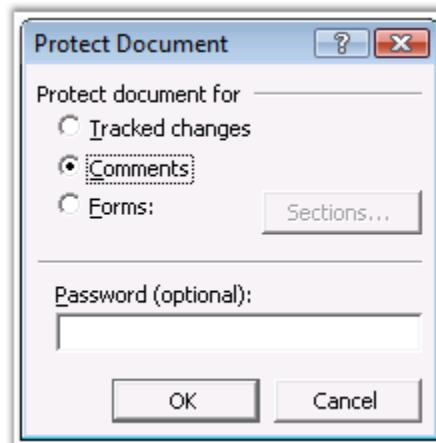


Figure 81: Protect Document Dialog Box

DocIO supports such protection while reading and writing Word documents for both **.doc** and **.docx** formats, and this can be provided through the following APIs.

- **AllowOnlyComments**—only comments are allowed.
- **AllowOnlyFormFields**—modification of form field value is allowed.
- **AllowOnlyRevisions**—only revisions are allowed.
- **AllowOnlyReading**—only reading is allowed.
- **NoProtection**—document has no protection.

You can also provide a password to restrict the user from editing documents. You can enable or disable document protection by using the **WordDocument.ProtectionType** property, when the document is opened with DocIO.

The following example illustrates the use of ProtectionType property.

[C#]

```
IWordDocument doc = new WordDocument( "sample.doc");
doc.Protect(ProtectionType.AllowOnlyComments, "password");

doc.Save( "Protection.doc" );
```

[VB .NET]

```
Dim doc As IWordDocument = New WordDocument("sample.doc")
doc.Protect(ProtectionType.AllowOnlyComments, "password")

doc.Save("Protection.doc")
```

For More Information Refer:

[Encryption and Decryption](#)

4.7.1 Encryption and Decryption

Encryption is a method of protecting the Word document. It is based on a password, which converts it into a form that cannot be understood. It restricts anonymous users from accessing a document.

Decryption is the process of converting encrypted data, back into its original form so that data can be read from the document. A password for encrypting a Word document is set in Microsoft Word through the **Security** tab in the **Options** dialog box.

The following example illustrates how to encrypt and decrypt a Word document.

[C#]

```
//Encrypting the Word document with password.
document.EncryptDocument(password);

// Opening the encrypted Workbook.
WordDocument document = new WordDocument(filename, password);
```

[VB .NET]

```
'Encrypting the Word document with password.
document.EncryptDocument(password)

'Opening the encrypted Workbook.
Dim document As New WordDocument(filename, password)
```



Note: Currently in the Silverlight platform, Essential DocIO does not support encryption and decryption techniques of MS Word 2013 format documents.

Encryption and Decryption Samples Installation Location:

To Locate the Encrypt and Decrypt samples:

\DocIO.WPF\Samples\3.5\WindowsSamples\Prepare\Encrypt and Decrypt

Viewing Encryption and Decryption samples:

1. Click Start-->All Programs-->**Synfusion-->Essential Studio <version number> -->Dashboard**.
2. Open **Reporting** edition samples. Click the drop-down button of **WPF** platform and select the **Explore samples** option.

For more information refer to section [2.2 Samples and Location](#).

4.8 Conversion

DocIO provides support to convert a Word document into an image of type Bitmap or EMF. It enables to easily convert a single specified page, group of pages or a whole document into image format.

The following overloads of the **RenderAsImages** method that can be used to convert a Word document into an image.

- **WordDocument.RenderAsImages(imageType)**-This is used to convert the whole document into an image.
- **WordDocument.RenderAsImages(pageIndex, imageFormat)**-This is used to render/convert a particular page of the document into an image; it returns the resultant image of type Stream.
- **WordDocument.RenderAsImages(pageIndex, imageType)**-This is used to render/convert a particular page of the document into an image; it returns the resultant image of type Image.
- **WordDocument.RenderAsImages(pageIndex, noOfPages, imageType)**-This is used to render/convert multiple number of pages in the document, starting from the specified page index. It returns the resultant image of type Image[] array.

[C#]

```
Image[] images = document.RenderAsImages(ImageType.Metafile);
Stream stream = document.RenderAsImages(0, ImageFormat.Emf);
Image image = document.RenderAsImages(5, ImageType.Metafile);

// This converts pages 2,3,4 in the document into images.
Image[] images = document.RenderAsImages(1, 3, ImageType.Metafile);
```

[VB]

```
Dim images() As Image = document.RenderAsImages(ImageType.Metafile)
Dim stream As Stream = document.RenderAsImages(0, ImageFormat.Emf)
Dim image As Image = document.RenderAsImages(5, ImageType.Metafile)

' This converts pages 2,3,4 in the document into images.
Dim images() As Image = document.RenderAsImages(1, 3, ImageType.Metafile)
```

Note:

- **Parameter "pageIndex" is a zero based index.**
- **Layouting of pages in DocIO is not the same as layouting of pages in Word. The total number of pages and layouting of the elements tend to vary.**
- **Currently Doc to Image conversion is not supported in Silverlight application.**

For More Information Refer:

[Doc To RTF](#), [Doc To HTML](#), [Doc to PDF](#)

4.8.1 Doc to RTF

You can now open or create Word documents and save to the .RTF format, enabling RTF conversion by using DocIO.

The following example illustrates how to save a Word document to RTF format.

[C#]

```
WordDocument doc = new WordDocument("sample.doc");
doc.Save( "samplertf.rtf", FormatType.Rtf );
```

[VB.NET]

```
Dim doc As New WordDocument("sample.doc")
doc.Save( "samplertf.rtf", FormatType.Rtf )
```

Document Elements

DocIO supports the following document elements.

- Element Name
- Main Document and Document Properties
- Paragraph
- Table
- Picture
- Header / Footer
- Field (Simple)
- TOC Field
- Bookmark
- Break (Line, Page)
- Section Property
- Paragraph Format
- Table Format
- Character Format
- Text Box
- Form Fields

- Document Background
- Watermark
- Nested Table
- Footnote / Endnote
- Lists
- Hyperlink
- Symbols [Not all symbols are supported]

DocIO does not support the following document elements.

- OLE Object
- RTL



Note: Currently Doc to RTF conversion is not supported in Silverlight application. Doc to HTML

4.8.2 Doc to HTML

You can now open or create Word documents and save to the HTML format, enabling HTML conversion by using DocIO.

The following example illustrates how to save a Word document to HTML format.

[C#]

```
WordDocument doc = new WordDocument(@"..\..\DocToHTML.doc");
HTMLExport htmlExport = new HTMLExport();
htmlExport.SaveAsXhtml(doc, "doctohtml_res.html");
```

[VB.NET]

```
Dim doc As New WordDocument("..\\..\\DocToHTML.doc")
Dim htmlExport As New HTMLExport()
htmlExport.SaveAsXhtml(doc, "doctohtml_res.html")
```

Document Elements

DocIO supports the following document elements.

Document Element	Attribute	Supported	Notes
Bookmark	Position	Yes	-

Border	Color	Yes	-
Border	Distance from text	Yes	-
Border	Line style	Partial	Some line styles are rendered as solid.
Border	Line width	Yes	-
Document Properties		Yes	-
Drawing objects	Shapes	Partial	Images and horizontal rules are exported.
Drawing objects	Text	Partial	Text from text boxes and other shapes is rendered in the main text.
Drawing objects	Images	-	-
Field		Yes	Current field result is output, but the result is not recalculated.
Footnotes and Endnotes		Yes	-
Form Field	Text input	Yes	-
Header / Footer	Different per section	Partial	Only primary header is output at the beginning of a section.
Hyperlink	External URL	Yes	-
Hyperlink	Local	Yes	-
Image	Cropping	Yes	-
Image	Inline	Yes	-
Image	Scale	Yes	-
List	Custom bullets	Yes	-
List	Multi-level	Yes	-
List	Numbered	Yes	-

List	Restart numbering	Yes	-
List	Standard bullets	Yes	-
Paragraph	Alignment	Yes	-
Paragraph	Borders	Yes	See Borders, for more details.
Paragraph	First line indent or hanging	Yes	-
Paragraph	Keep together	Yes	-
Paragraph	Keep with next	Yes	-
Paragraph	Left and right indent	Yes	-
Paragraph	Line spacing	Partial	All line spacing is output as atleast line spacing.
Paragraph	Line spacing	Yes	-
Paragraph	Page break before	Yes	-
Paragraph	Shading	Yes	See Shading, for more details.
Paragraph	Spacing before and after	Yes	-
Paragraph	Window control	Yes	Output as both windows and orphans.
Shading	Background color	Partial	Solid background colors are supported.
Shading	Foreground color	Partial	Solid foreground color is used when background color is auto.
Styles	Paragraph styles	Yes	-
Styles	Character styles	Yes	-
Styles	List styles	Partial	Not all formatting has effect. Considered in only inline styles.
Table	Alignment	Yes	-

Table	Cell margins	Yes	-
Table	Column widths	Yes	-
Table	Indent from left	Yes	-
Table	Preferred width	Yes	-
Table	Spacing between cells	Yes	-
Table Cell	Borders	Partial	See Borders, for more details.
Table Cell	Cell margins	Partial	Only default table cell margins left and right are supported.
Table Cell	Horizontal merge	Yes	-
Table Cell	Shading	Partial	See Shading, for more details.
Table Cell	Text direction	Yes	-
Table Cell	Vertical alignment	Yes	-
Table Cell	Vertical merge	Yes	-
Table Row	Height	Yes	-
Table Row	Padding	Yes	-
Text	All caps	Yes	-
Text	Bold	Yes	-
Text	Character spacing	Yes	-
Text	Color	Yes	-
Text	Emboss	Partial	Rendered as bold.
Text	Engrave	Partial	Rendered as bold.
Text	Font	Yes	-
Text	Hidden	Yes	-
Text	Highlighting	Yes	-

Text	Imprint	Partial	Rendered as bold.
Text	Italic	Yes	-
Text	Line breaks	Yes	-
Text	Outline	Partial	Rendered as bold.
Text	Page breaks	Yes	-
Text	Shading	Partial	See Shading, for more details.
Text	Small caps	Yes	-
Text	Special symbols	Yes	-
Text	Strike out	Yes	-
Text	Subscript / Superscript	Yes	-
Text	Underline	Partial	Underline types and colors are ignored.



Notes: Currently Doc to Html conversion is not supported in Silverlight application.

4.8.3 Doc to PDF

Essential DocIO allows you to export the word document into a PDF document. Use the **ConvertToPDF** method of **DocToPDFConverter** class, to convert the doc to pdf, and save the PDF document. Using this, the user can easily convert the word document to PDF document.



Note: You need to have Essential PDF and Essential DocIO installed in your system. Since "Syncfusion.DocToPDFConverter.Base.dll" is conditionally shipped when both DocIO.Base and Pdf.Base is installed.

Assembly Dependency for this Conversion

- Syncfusion.DocToPDFConverter.Base.dll
- Syncfusion.DocIO.Base.dll

- Syncfusion.Pdf.Base.dll
- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll

[C#]

```
WordDocument wordDoc = new WordDocument("sample.doc");
DocToPDFConverter converter = new DocToPDFConverter();

// Convert word document into PDF document
PdfDocument pdfDoc = converter.ConvertToPDF(wordDoc);

// Save the pdf file
pdfDoc.Save("DoctoPDF.pdf");
```

[VB.NET]

```
Dim wordDoc As New WordDocument("sample.doc")
Dim converter As New DocToPDFConverter()

' Convert word document into PDF document
Dim pdfDoc As PdfDocument = converter.ConvertToPDF(wordDoc)

' Save the pdf file
pdfDoc.Save("DoctoPDF.pdf")
```

Supported Elements

With the initial version of the feature, this feature provides support for the following elements.

- Paragraph and character formatting
- Multi-Column Texts
- Headers and Footers
- Bulleted, numbered and multi-level lists
- Images
- Tables (both simple and nested)
- Table styles for docx formats (Word 2007 and Word 2010 formats)
- Breaks (page, section, linebreak, etc)
- OLEObject
- Textbox
- Page Settings and background image
- Document Properties

Paragraph and Character formatting

This feature supports almost all the paragraph formatting except Full-Justification. The supported paragraph formatting features are,

- Paragraph and character fonts
- Font styles (Bold, Italic, Underline, and Strike through)
- Subscript and Superscript
- Paragraph and text highlighting
- Indents, tabs and spaces
- Line spacing
- Left, right and center justification

Known Limitations

- Borders around paragraphs.
- Full Justification.

Multi-Column Texts

The word documents containing multi-column text were supported.

Known Limitations - But the output may look different in case full-justification formatting is applied on to the columns.

Headers and Footers

The page headers and footers are supported and can contain images, texts and page number fields.

Bulleted, Numbered and Multi-level lists

The bulleted list, numbered and multi-level list were supported with proper indentation and alignments as represented in the word document.

Known Limitations - In some case, the image bullets which is set on document may be replaced by the following symbol in the generated document. For eg :

 Essential DocIO.  Essential XlsIO.  Essential PDF.	==> Will be replaced as ==>	<ul style="list-style-type: none">• Essential DocIO.• Essential XlsIO.• Essential PDF.
--	-----------------------------	--

Figure 82: Bulleted, Numbered and Multi-level Lists

Images

The images present in the document are supported along with their corresponding positions and sizes.

Known Limitations - However, the images placed inside a shape will not be preserved in the generated PDF document.

Tables

Both simple and nested tables are supported with proper preservation of text formatting and images present inside the table cell. Text directions are also supported.

Known Limitations

- Tables making use of patterns and 3D borders will not be retained in the output document.
- Absolutely positioned tables are not supported.

Doc to PDF Conversion Support for Table Styles for Word 2007 and Word 2010 Documents

Support is now added for table styles in Doc to PDF conversion for Word 2007 and Word 2010 documents. During Doc to PDF conversion, Table-style support provides a unique look and feel to tables in the converted PDF documents, similar to the tables in Word documents.



Figure 83: MS Word Document with Table Style

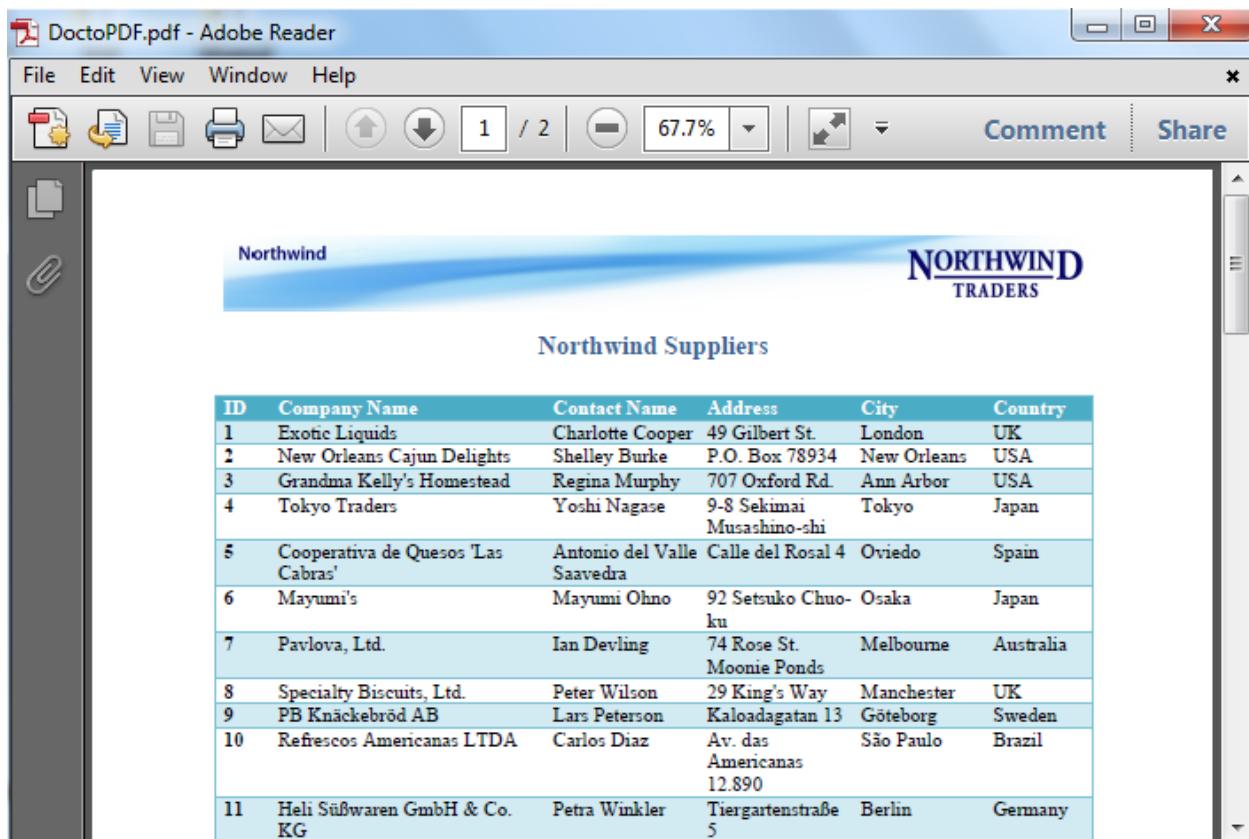


Figure 84: Converted PDF with Table Style – Light Shading

Known Limitations - Table styles for Word 97 – 2003 documents are not supported

Breaks

The columns, section, line and page breaks are fully-supported.

OLEObject

The OLEObjects are partially supported, (i.e) image which represents a particular document will be available in the generated PDF document. But the object associated with the object will not be converted into the generated document.

Text box

The text value present in the text box will be rendered as text at its actual position in the generated PDF document. Text directions are also supported.

PageSettings

The actual page settings will be preserved in the generated PDF documents, which includes page size, orientation, page borders and its background image if available.

Document Properties

The document properties present in the word documents will also be preserved in the generated PdfDocument.

Un-Supported Elements

The following are the list of un-supported elements, which will be supported in the future releases and will not be preserved in the generated PDF document.

- Shapes and auto shapes
- Comments
- Hyperlinks
- Bookmarks
- Foot note and end note
- Dynamic Fields
- Charts
- Table of Contents

Known Limitations - Pagination

Pagination

Essential DocIO makes sensible decision while layouting the text, and its supported elements while generating the PDF documents. But however, we cannot guarantee pagination with all the documents.



Note: Currently Doc to Pdf conversion is not supported in Silverlight application.

4.8.4 RTF to Doc

Essential DocIO allows to import the RTF document directly into the word document.

The following code illustrates how RTF file can be opened and saved as a word document.

[C#]

```
//Opening the RTF file
WordDocument doc = new WordDocument("Sample.rtf",
FormatType.Rtf);
//Saving the RTF file as a word document
doc.Save ("RtfToDoc_Res.doc", FormatType.Doc);
```

[VB.NET]

```
'Opening the RTF file
Dim doc As New WordDocument("Sample.rtf", FormatType.Rtf)
'Saving the RTF file as a word document
doc.Save("RtfToDoc_Res.doc", FormatType.Doc);
```

Supported Elements

This feature provides support for the following elements:

- Paragraph and Character Formatting
- Tables
- Bookmarks
- Headers and Footers
- Images
- List
- Page setting
- Multi column text
- Breaks
- Document properties
- Fields

Fields

This feature supports the preservation of fields in Rtf to Doc conversion.

Paragraph and Character Formatting

This feature supports almost all the paragraph and character formatting. The supported formatting features are:

- Paragraph borders
- Indentation and Pagination
- Spacing and Tabs

- Left, Right and Center justification
- Font styles(Bold, Italic, Underline, Strike through)
- Font sizeand font name for the character
- Text highlighting and Text color

Known Limitation:

- Subscript and Superscript
- Revision tracking

Tables

This feature supports both simple and the nested tables. This feature also provides support for Text formatting, Paragraph formatting and images inside the tables

Known Limitation

- Tables styles and 3D border for the tables are not supported.

Bookmarks

This feature fully supports the bookmark present in the document.

Headers and Footers

This feature supports page headers and footers. The Page header and footer can contain paragraphs, tables and images.

Images

This feature support images present in the RTF document along with their position and size.

Known Limitation:

- Image Present inside the shapes are not supported.

List

This feature support bullets, numbered and multi level bulleted list along with their alignment and indentation.

Known limitaion:

- Image bullets are not supported

Page Settings

This feature support page settings such as Margin, Orientation, Paper size, Header distance and Footer distance.

Known Limitation:

- Background image and background color for the page is not supported.

Multi Column text

This feature fully supports multi column text within the RTF document.

Breaks

This feature supports breaks such as column break, section break, line break and page break.

Document properties

This feature supports document properties present in the RTF document.

Unsupported elements

The following are the unsupported elements which will be supported in our future release:

- Shapes and Auto Shapes
- Footnotes and End notes
- Comments
- Table of contents
- Hyperlinks
- Table styles
- OLE objects
- Document protection
- RTL support
- Water marks
- Symbols



Notes: Currently RTF to Doc conversion is not supported in Silverlight application.

4.8.5 Doc to EPub

EPub is the short form of Electronic Publication; the popular e-book standard by [International Digital Publishing Forum](#) (IDPF) with the extension .epub. EPub files have reflowable content, which allows the document text display to be optimized based on the reader and device in use.

Note: A reflowable document is a type of electronic document that can adapt its presentation to the output device. Typical desktop publishing (DTP) output formats like Postscript or PDF are page-oriented, so are not generally reflowable, whereas the world wide web standard, HTML is a reflowable format.

Use Case Scenario

This feature helps users to convert Word documents to reflowable content (EPUB Formatted Book) that can be used for distribution and sales.

EPub Conversion Using DocIO

Essential DocIO supports conversion of MS Word documents to EPub v2.0.1. DocIO supports conversion of elements such as Text and Paragraph formatting, Lists, Images, Hyperlinks, Tables and Footnotes to EPub format.

By default, Table of Contents (TOC) is enabled in the EPub document. It is generated based on the built-in heading styles or custom styles mentioned in the TOC field.

Note: You need to have an EPub reader installed in the machine to view the output EPub document.

Support for conversion to EPub is available in the following platforms:

- Windows Forms
- ASP.NET
- WPF
- ASP.NET MVC

The following code illustrates how to convert a Word document to EPub file format.

[C#]

```
// Load any .doc or .docx file
WordDocument document = new WordDocument(filename);

// Save the EPub file
document.Save("Sample.epub", FormatType.EPub);
```

[VB]

```
' Load any .doc or .docx file
Dim document As WordDocument = New WordDocument(filename)

' Save the EPub file
document.Save("Sample.epub", FormatType.EPub)
```

The following is the sample image of the output EPub document when converted, using the above code.

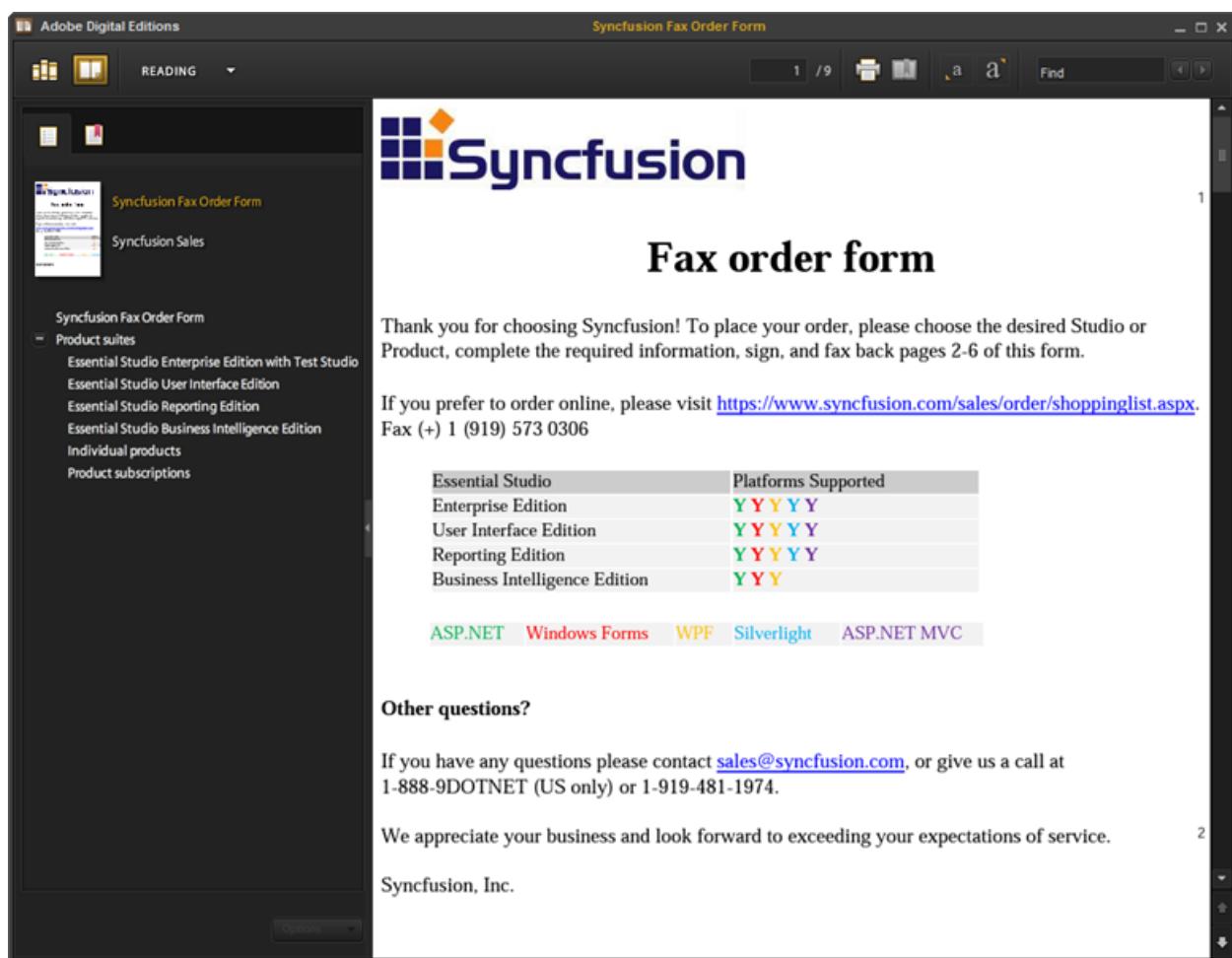


Figure 85: Output EPub document

Embedding Font

Conversion of EPub using default options does not embed font files. Hence, the reading device uses its own default font for the texts in the document, which may vary depending on the reader being used. To read the texts in the same font as used in the input word document, the user should embed the font files into the generated EPub. This can be done by turning on EPubExportFont property. By default, this property is set to false since this actually embeds the exact font file from the machine, which may increase the size of the EPub document.

The following code illustrates how to embed font file.

[C#]

```
// Load any .doc or .docx file
WordDocument document = new WordDocument(filename);

// Turn on embedding font files
document.SaveOptions.EPubExportFont = true;

// Save the EPub file
document.Save("Sample.epub", FormatType.EPub);
```

[VB]

```
' Load any .doc or .docx file
Dim document As WordDocument = New WordDocument(filename)

' Turn on embedding font files
document.SaveOptions.EPubExportFont = True

' Save the EPub file
document.Save("Sample.epub", FormatType.EPub)
```

The following is the sample image of output EPub document when converted using the above code.

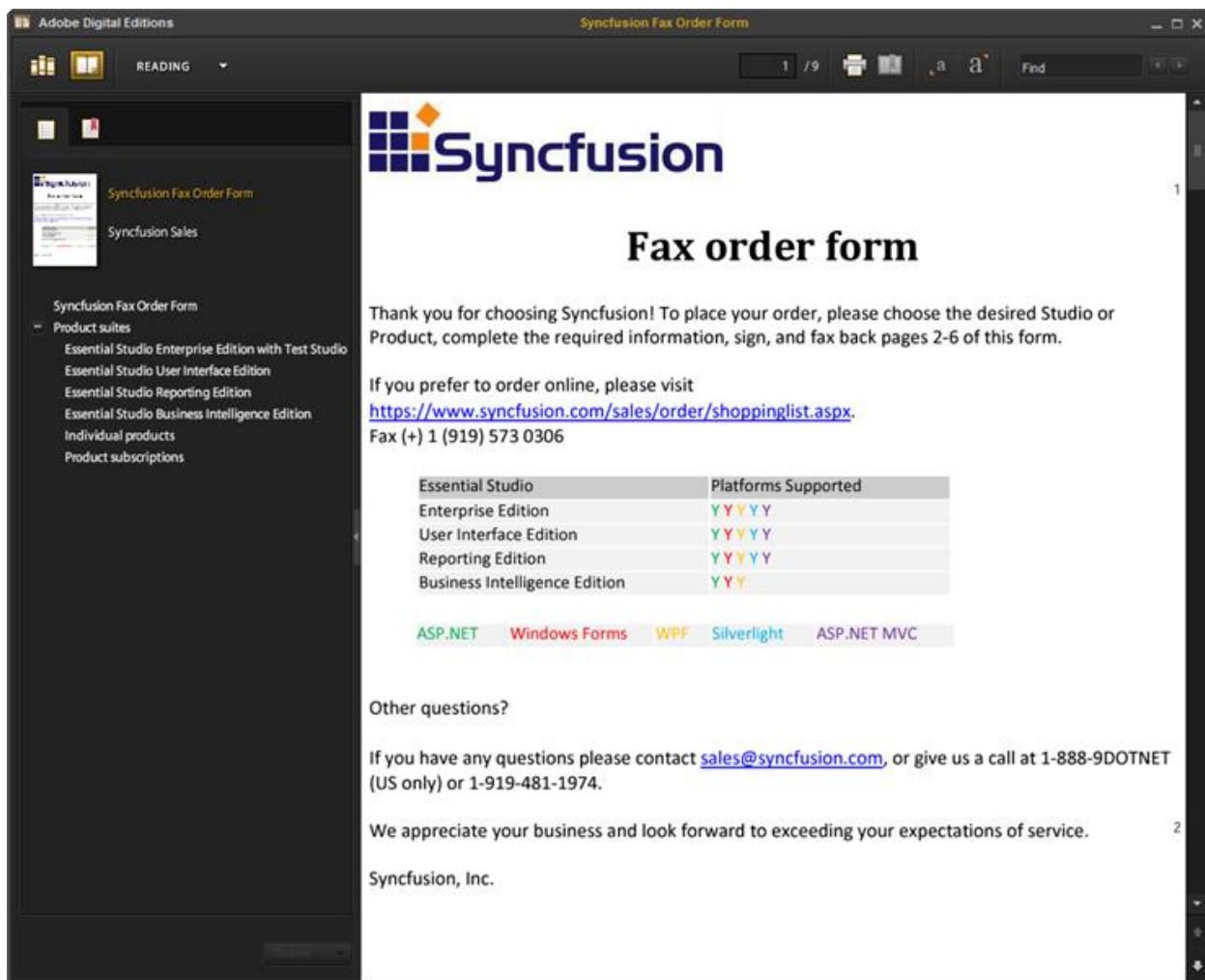


Figure 86: EPub with embedded font files

Exporting Header and Footer

Header and Footer in the Word document are helpful in placing specific information that has to be displayed on every page. These headers and footers can be exported to the EPub document in such a way that only the first section header would appear at the top of the document and the first section footer would appear at the end of the document. This can be done by turning on `HtmlExportHeadersFooters` property. By default, this property is set to true and hence it always exports header and footer.

The following code illustrates how to export header and footer.

```
[C#]

// Load any .doc or .docx file
WordDocument document = new WordDocument(filename);
```

```
// Turn on exporting headers and footers
document.SaveOptions.HtmlExportHeadersFooters = true;

// Save the EPub file
document.Save("Sample.epub", FormatType.EPub);
```

[VB]

```
' Load any .doc or .docx file
Dim document As WordDocument = New WordDocument(filename)

' Turn on exporting headers and footers
document.SaveOptions.HtmlExportHeadersFooters = True

' Save the EPub file
document.Save("Sample.epub", FormatType.EPub)
```

The following is the sample image of the output EPub document with header and footer disabled.

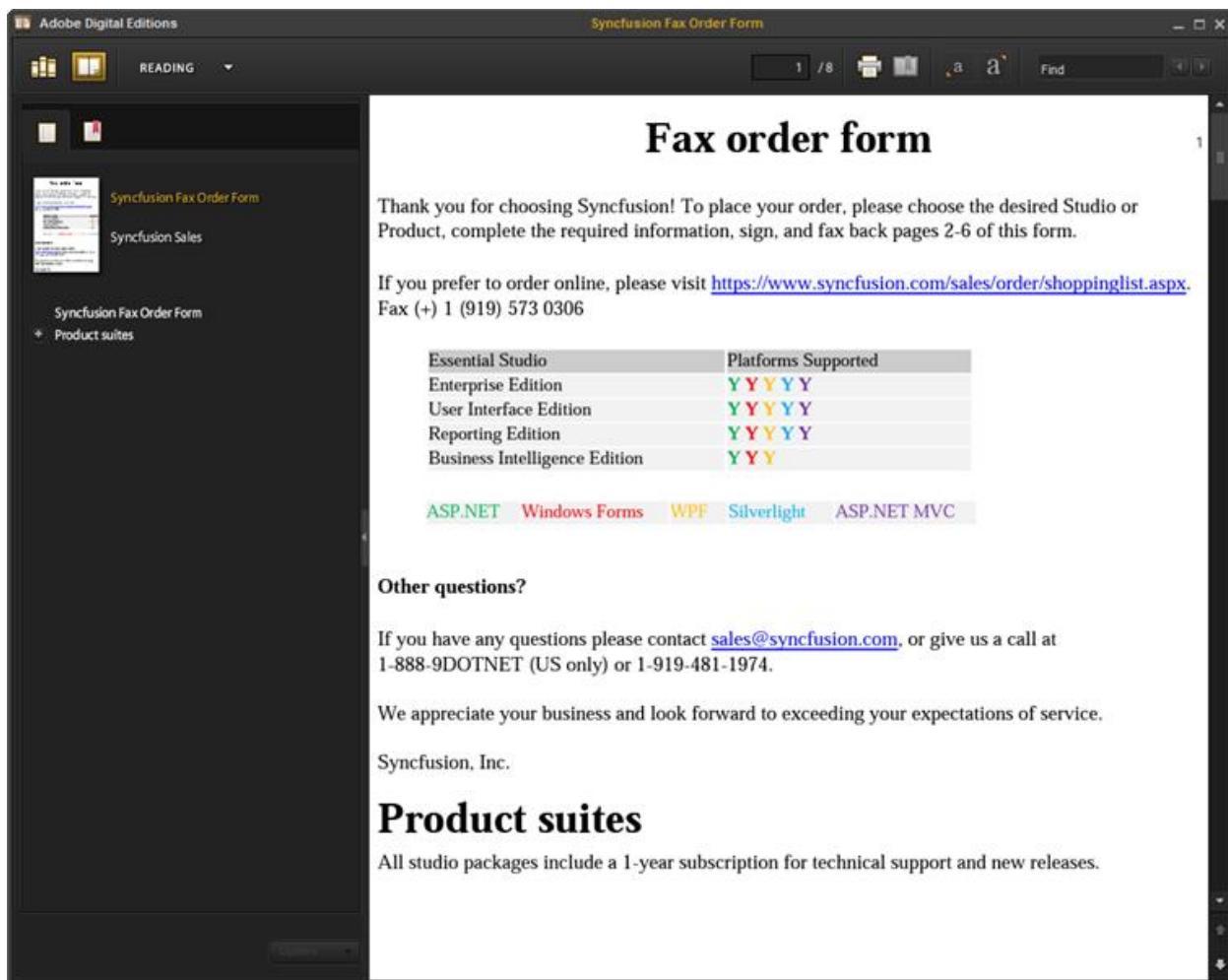


Figure 87: EPub document without header and footer

Sample Link

The paths to access the samples are as given below:

Windows Forms:

Start->All Programs->Syncfusion->Essential Studio x.x.x.xx->Dashboard->Windows Forms->DocIO.Windows->Samples->2.0->Import And Export->Doc to EPub

ASP.NET:

Start->All Programs->Syncfusion->Essential Studio x.x.x.xx->Dashboard->ASP.NET->DocIO.Web->Samples->3.5->Import and Export->DocToEPub

WPF:

Start->All Programs->Syncfusion->Essential Studio x.x.x.xx->Dashboard->WPF->DocIO.WPF->Samples->3.5-> WindowsSamples->Import and Export-> Doc to EPub

ASP.NET MVC:

Start->All Programs->Syncfusion->Essential Studio x.x.x.xx->Dashboard->ASP.NET MVC->DocIO.MVC->Samples->3.5->Views->ImportandExport->DOCToEPub.aspx

Supported Elements

The following are the Supported Elements:

- Text and Paragraph Formatting
- Lists
- Tables
- Images
- Footnote
- Hyperlink
- Styles
- Table of Contents
- Document Properties

Known Limitations

The following are the known limitations:

- Embedding font files may increase the size of the EPub document
- Embedding font files is not supported in medium trust



Notes:Currently Doc to EPub conversion is not supported in Silverlight application.

4.9 Docx Support

Microsoft introduced the .docx file format in its new Office and Word applications to replace the commonly used doc format. Essential DocIO now provides support for.docx files. Docx files are created using the same APIs as for .doc files using Essential DocIO. Essential DocIO provides support for:

- Creating a .docx file from scratch
- Read/Modify a .docx file

Creating a .docx file

To create a .docx file:

1. Create a word document using Essential DocIO APIs.

[C#]

```
WordDocument document = new WordDocument();
//Add a new section to the document.
IWSection section = document.AddSection();
//Adding a new paragraph to the section.
IWParagraph paragraph = section.AddParagraph();
//Insert Text into the paragraph
paragraph.AppendText( "Hello World!" );
```

2. Create an instance of SaveFile dialog.

The following lines of code create an instance of SaveFile Dialog and set the properties and display the Save dialog on the screen.

[C#]

```
SaveFileDialog sfd = new SaveFileDialog()
{
    Filter = "Docx files (*.docx)|*.docx|All files (*.*)|*.*",
    DefaultExt = ".docx",
    FilterIndex = 1
};
if (sfd.ShowDialog() == true)
```

3. Save the document as .docx format.

The following code snippet will save the Word document.

[C#]

```
{
    using (Stream stream = sfd.OpenFile())
    {
        document.Save(stream, FormatType.Docx);
    }
}
```

4. Run the application. The .doc file is converted to .docx file.

The same can be achieved using the VB.Net code as well.

[VB.NET]

```

Dim document As New WordDocument()
'Add a new section to the document.
Dim section As IWSection = document.AddSection()
Adding a new paragraph to the section.
Dim paragraph As IWPParagraph = section.AddParagraph()
'Insert Text into the paragraph
paragraph.AppendText("Hello World!")
Dim sfd As New SaveFileDialog()
If sfd.ShowDialog() = True Then
    Using stream As Stream = sfd.OpenFile()
        document.Save(stream, FormatType.Docx)
    End Using
End If

```

Saving a .doc File as .docx

DocIO also has the ability to save .doc files into .docx format (i.e, Microsoft Word 2007/2010/2013 files format). All the elements supported by .Doc are supported in Docx. Some of the supported elements are listed below.

1. Creating an instance of Word document.

The following code snippet creates an instance of the word document and opens the word document named SourceDocument.doc. The FormatType.Doc specifies that the document is of type Word 97-2003.

[C#]

```

WordDocument document = new WordDocument();
document.Open("SourceDocument.doc", FormatType.Doc);
SaveFileDialog sfd = new SaveFileDialog()
{
    Filter = "Docx files (*.docx)|*.docx|All files (*.*)|*.*",
    DefaultExt = ".docx",
    FilterIndex = 1
};

```

2. Save the document as .docx format.

The following code snippet will display the Save dialog on the screen and save the Word document.

[C#]

```

if (sfd.ShowDialog() == true)
{
    using (Stream stream = sfd.OpenFile())
    {
        document.Save(stream, FormatType.Docx);
    }
}

```

3. Run the application. The .doc file is converted to .docx file.
The same can be achieved using the VB.Net code as well.

[VB.NET]

```

Dim document As New WordDocument()
document.Open("SourceDocument.doc", FormatType.Doc)
Dim sfd As New SaveFileDialog()
If sfd.ShowDialog() = True Then
    Using stream As Stream = sfd.OpenFile()
        document.Save(stream, FormatType.Docx)
    End Using
End If

```

Reading/Modifying .docx File

Once a file is saved into .docx format, Essential DocIO provides support for reading and modifying .docx files. Docx files can be read using the same API as that of .doc files; except for the format in which it is opened.

The control uses the following API to read the .docx format.

[C#]

```
WordDocument doc = new WordDocument("sample.docx", FormatType.Docx);
```

[VB.NET]

```
Dim doc As New WordDocument("sample.docx", FormatType.Docx)
```

Run the file. You will be able to open and edit a .docx file.

4.10 Supported Elements

Supported Elements in Silverlight

The list of various supported and non-supported elements of Essential DocIO in Silverlight platform is given in the following table.

Table 5: Supported element in Silverlight platform

Element	Doc		Word 2007/Word 2010/ Word 2013	
	Read	Write	Read	Write
Break – Page break, section break, column break and Line break	Yes	Yes	Yes	Yes
Symbols	Yes	Yes	Yes	Yes
Comments	Yes	Yes	Yes	Yes
Footnote and endnote	Yes	Yes	Yes	Yes
Table of Contents	Yes	Yes	Yes	Yes
Pictures (*.bmp, *.jpg, *.png, *.emf, *.tif and *.gif)	Yes	Yes	Yes	Yes
Paragraph border	Yes	Yes	Yes	Yes
Bullets and lists	Yes	Yes	Yes	Yes
Background Color	Yes	Yes	Yes	Yes
Fill effect and Watermark	Yes	Yes	Yes	Yes
Table with formatting	Yes	Yes	Yes	Yes
Table styles	No	No	No	No
Form field objects	Yes	Yes	Yes	Yes
Header/Footer with images	Yes	Yes		
Drawing Objects	Yes	Yes Limited[Can't create new container]	No	No
Page Setup	Yes	Yes	Yes	Yes
Hyperlink	Yes	Yes	Yes	Yes

Element	Doc		Word 2007/Word 2010/ Word 2013	
	Read	Write	Read	Write
Font Settings	Yes	Yes	Yes	Yes
Paragraph Settings	Yes	Yes	Yes	Yes
Border and Shading	Yes	Yes	Yes	Yes
Textbox	Yes	Yes	Yes	Yes
Text direction	Yes	Yes	Yes	Yes
Theme settings	Yes	No	Yes	No
Track changes	Yes	Yes – limited [can only accept/reject]	Yes	Yes – limited [can only accept/reject]
Macros	No	No	No	No
Auto shapes	Yes	No	Yes	No
Document variables	Yes	Yes	Yes	Yes
Encryption and Decryption	Yes	Yes	Yes—limited [except Word 2013]	Yes—limited [except Word 2013]
Chart	No	No	No	No
Mail merge	Yes	Yes	Yes	Yes
OLE object	Yes	Yes	Yes	Yes
Hyperlinks	Yes	Yes	Yes	Yes
View Setup	Yes	Yes	Yes	Yes
Line numberings	Yes	Yes	Yes	Yes
Built-in Styles (Paragraph and Character styles)	Yes	Yes	Yes	Yes
Custom Styles (Paragraph	Yes	Yes	Yes	Yes

Element	Doc		Word 2007/Word 2010/ Word 2013	
	Read	Write	Read	Write
and Character styles)				
Document Properties	No	No	Yes	Yes

Supported Elements in Windows, ASP.Net, WPF and ASP.Net MVC

The list of various supported and non-supported elements of Essential DocIO in Windows, ASP.Net, WPF, and ASP.NET MVC platforms is given in the following table.

Table 6: Supported elements in Windows, ASP.NET, WPF, and ASP.NET MVC

Element	Doc		Word 2007 / Word 2010 / Word 2013	
	Read	Write	Read	Write
Break – Page break, section break, column break and Line break	Yes	Yes	Yes	Yes
Symbols	Yes	Yes	Yes	Yes
Comments	Yes	Yes	Yes	Yes
Footnote and endnote	Yes	Yes	Yes	Yes
Table of Contents	Yes	Yes	Yes	Yes
Pictures (*.bmp, *.jpg, *.png, *.emf, *.tif and *.gif)	Yes	Yes	Yes	Yes
Paragraph border	Yes	Yes	Yes	Yes
Bullets and lists	Yes	Yes	Yes	Yes
Background Color	Yes	Yes	Yes	Yes
Fill effect and Watermark	Yes	Yes	Yes	Yes
Table with formatting	Yes	Yes	Yes	Yes
Table styles	No	No	No	No

Element	Doc		Word 2007 / Word 2010 / Word 2013	
	Read	Write	Read	Write
Form field objects	Yes	Yes	Yes	Yes
Header/Footer with images	Yes	Yes		
Drawing Objects	Yes	Yes Limited[Can't create new container]	No	No
Page Setup	Yes	Yes	Yes	Yes
Hyperlink	Yes	Yes	Yes	Yes
Font Settings	Yes	Yes	Yes	Yes
Paragraph Settings	Yes	Yes	Yes	Yes
Border and Shading	Yes	Yes	Yes	Yes
Textbox	Yes	Yes	Yes	Yes
Text direction	Yes	Yes	Yes	Yes
Theme settings	Yes	No	Yes	No
Track changes	Yes	Yes – limited [can only accept/reject]	Yes	Yes – limited [can only accept/reject]
Macros	No	No	No	No
Auto shapes	Yes	No	Yes	No
Document variables	Yes	Yes	Yes	Yes
Encryption and Decryption	Yes	Yes	Yes	Yes
Chart	No	No	No	No
Mail merge	Yes	Yes	Yes	Yes
OLE object	Yes	Yes	Yes	Yes

Element	Doc		Word 2007 / Word 2010 / Word 2013	
	Read	Write	Read	Write
Hyperlinks	Yes	Yes	Yes	Yes
View Setup	Yes	Yes	Yes	Yes
Line numberings	Yes	Yes	Yes	Yes
Built-in Styles (Paragraph and Character styles)	Yes	Yes	Yes	Yes
Custom Styles (Paragraph and Character styles)	Yes	Yes	Yes	Yes
Document Properties	Yes	Yes	Yes	Yes

Supported Elements in Windows Store Platform

The list of various supported and non-supported elements of Essential DocIO in Windows store platform is given in the following table.

Table 7: Supported element in Silverlight platform

Element	Doc		Word 2007 / Word 2010 / Word 2013	
	Read	Write	Read	Write
Break – Page break, section break, column break and Line break	Yes	Yes	Yes	Yes
Symbols	Yes	Yes	Yes	Yes
Comments	Yes	Yes	Yes	Yes
Footnote and endnote	Yes	Yes	Yes	Yes
Table of Contents	Yes	Yes	Yes	Yes
Pictures (*.bmp, *.jpg, *.png, *.emf, *.tif and *.gif)	Yes	Yes	Yes	Yes

Element	Doc		Word 2007 / Word 2010 / Word 2013	
	Read	Write	Read	Write
Paragraph border	Yes	Yes	Yes	Yes
Bullets and lists	Yes	Yes	Yes	Yes
Background Color	Yes	Yes	Yes	Yes
Fill effect and Watermark	Yes	Yes	Yes	Yes
Table with formatting	Yes	Yes	Yes	Yes
Table styles	No	No	No	No
Form field objects	Yes	Yes	Yes	Yes
Header/Footer with images	Yes	Yes		
Drawing Objects	Yes	Yes Limited[Can't create new container]	No	No
Page Setup	Yes	Yes	Yes	Yes
Hyperlink	Yes	Yes	Yes	Yes
Font Settings	Yes	Yes	Yes	Yes
Paragraph Settings	Yes	Yes	Yes	Yes
Border and Shading	Yes	Yes	Yes	Yes
Textbox	Yes	Yes	Yes	Yes
Text direction	Yes	Yes	Yes	Yes
Theme settings	Yes	No	Yes	No
Track changes	Yes	Yes – limited [can only accept/reject]	Yes	Yes – limited [can only accept/reject]
Macros	No	No	No	No

Element	Doc		Word 2007 / Word 2010 / Word 2013	
	Read	Write	Read	Write
Auto shapes	Yes	No	Yes	No
Document variables	Yes	Yes	Yes	Yes
Encryption and Decryption	Yes	Yes	Yes	Yes
Chart	No	No	No	No
Mail merge	Yes	Yes	Yes	Yes
OLE object	Yes	Yes	Yes	Yes
Hyperlinks	Yes	Yes	Yes	Yes
View Setup	Yes	Yes	Yes	Yes
Line numberings	Yes	Yes	Yes	Yes
Built-in Styles (Paragraph and Character styles)	Yes	Yes	Yes	Yes
Custom Styles (Paragraph and Character styles)	Yes	Yes	Yes	Yes
Document Properties	No	No	Yes	Yes

5 Frequently Asked Questions

Here are some common FAQs:

5.1 Does DocIO require MS Word to be installed on the report generation machine / server?

No. Essential DocIO is a 100% Native .NET library that does not depend on MS Word.

5.2 Would it be possible to view the generated report [DOC] using Essential DocIO?

No. Essential DocIO does not have a Graphical User Interface to view the generated DOC files. MS Word is required to view the DOC files generated by using DocIO. However there is a free Word Viewer from Microsoft which can also be used to view DOC files, and hence does not require the end-users to purchase MS Word.

5.3 Can Essential DocIO be used to read MS Word files? What are the MS Word versions that Essential DocIO can read?

Yes. Essential DocIO provides support for reading MS Word files. It can read the following MS Word versions.

- Word 97
- Word 2000
- Word 2002
- Word 2003
- Word 2007
- Word 2010
- Word 2013

Further topics under this section will help you find answers to common questions regarding Essential DocIO. It comprises the following FAQs:

5.4 How to format a Hyperlink by using DocIO?

Some details about the fields. Each field consists of the following.

- Field, which defines field properties (not formatting)
- FieldMark (FieldSeparator mark)
- Text of the Field (WTextTange)
- FieldMark (FieldEnd mark)

If you want to set formatting for the field text, you have to find **WTextRange(s)** between the field separator and field text to set the CharacterFormat for them.

The following code illustrates how to format the hyperlink text.

[C#]

```
doc.LastParagraph.AppendHyperlink("www.google.com", "google",
HyperlinkType.WebLink);
for (int i = 0, cnt = doc.LastParagraph.Items.Count; i < cnt; i++)
{
    if (doc.LastParagraph.Items[i] is WTextRange)
    {
        WTextRange text = doc.LastParagraph.Items[i] as WTextRange;
        text.CharacterFormat.FontSize = 33f;
    }
}
```

[VB]

```
doc.LastParagraph.AppendHyperlink("www.google.com", "google",
HyperlinkType.WebLink)
Dim i As Integer = 0, cnt As Integer = doc.LastParagraph.Items.Count
While i < cnt
    If TypeOf doc.LastParagraph.Items(i) Is WTextRange Then
        Dim text As WTextRange = TryCast(doc.LastParagraph.Items(i),
WTextRange)
        text.CharacterFormat.FontSize = 33.0F
    End If
    i += 1
End While
```

 **Note:** Following is the order in the paragraph.

- 1) **WField object**
- 2) **Field separator**
- 3) **Text to display**
- 4) **Field end**

5.5 How to format a Table?

You can format a table in two ways.

Method 1

You can use the **TableFormat** property of the table object.

[C#]

```
WTable table= doc.LastSection.AddTable() as WTable;
table.ResetCells(4, 4);
table.TableFormat.Borders.BorderType =
Syncfusion.DocIO.DLS.BorderStyle.Double;
table.TableFormat.LeftIndent=20;
```

[VB]

```
Dim table As IWTable = sec.body.AddTable()
table.ResetCells(4, 4)
table.TableFormat.Borders.BorderType =
Syncfusion.DocIO.DLS.BorderStyle.Double
table.TableFormat.LeftIndent = 20
```

Method 2

You can use the **ResetCell** method of the table object to format the cell. The following code illustrates this.

[C#]

```
WTable table= doc.LastSection.AddTable() as WTable;
RowFormat rowFormat=new RowFormat();
rowFormat.Borders.BorderType = Syncfusion.DocIO.DLS.BorderStyle.Double;
rowFormat.LeftIndent=20;
table.ResetCells(3,3,rowFormat,100);
```

[VB]

```
Dim table As IWTable = sec.body.AddTable()
Dim rowFormat As New RowFormat()
rowFormat.BackColor = Color.Purple
rowFormat.Borders.BorderType = Syncfusion.DocIO.DLS.BorderStyle.Double
rowFormat.LeftIndent = 20
table.ResetCells(3, 3, rowFormat, 100)
```

5.6 How to modify the Built-in styles?

You can use the **CreateBuiltInStyle** method of the Style class, to override the built-in styles.

The following code illustrates how to modify the Heading1 style.

[C#]

```
Style style = Style.CreateBuiltInStyle(BuiltinStyle.Heading1, doc) as Style;
style.CharacterFormat.Italic = true;
style.CharacterFormat.UnderlineStyle = UnderlineStyle.DotDash;
doc.Styles.Add(style);
para.ApplyStyle(style.Name);
```

[VB]

```
Dim style As Style = TryCast(Style.CreateBuiltInStyle(BuiltinStyle.Heading1,
doc), Style)
style.CharacterFormat.Italic = True
style.CharacterFormat.UnderlineStyle = UnderlineStyle.DotDash
doc.Styles.Add(style)
para.ApplyStyle(style.Name)
```

5.7 How to open a Document from Stream by using DocIO?

DocIO provides support for opening documents from stream. You can retrieve the document by using .NET classes, and then pass the stream to DocIO as follows.

[C#]

```
HttpWebRequest request =
(HttpWebRequest)WebRequest.Create("http://www.nfpa.org/assets/files//PDF/Forms/EvacuationGuide.doc");
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
Stream stream = response.GetResponseStream();
byte[] buffer = ReadFully(stream, 32768);

//Store bytes into the memory stream
MemoryStream ms = new MemoryStream();
ms.Write(buffer, 0, buffer.Length);
ms.Seek(0, SeekOrigin.Begin);
stream.Close();

//Creating a new document.
WordDocument doc = new WordDocument();

//Open the template document from the MemoryStream.
doc.Open(ms, FormatType.Doc);
doc.Save("Sample.doc", FormatType.Doc, Response,
HttpContentDisposition.InBrowser);
}

public static byte[] ReadFully(Stream stream, int initialLength)
{
// If we've been passed an unhelpful initial length, just
// use 32K.
if (initialLength < 1)
{
initialLength = 32768;
}
byte[] buffer = new byte[initialLength];
int read = 0;
int chunk;
while ((chunk = stream.Read(buffer, read, buffer.Length - read)) > 0)
{
read += chunk;
// If we've reached the end of our buffer, check to see if there's
```

```

// any more information
if (read == buffer.Length)
{
    int nextByte = stream.ReadByte();
    // End of stream? If so, we're done
    if (nextByte == -1)
    {
        return buffer;
    }
    // Nope. Resize the buffer, put in the byte we've just
    // read, and continue
    byte[] newBuffer = new byte[buffer.Length * 2];
    Array.Copy(buffer, newBuffer, buffer.Length);
    newBuffer[read] = (byte)nextByte;
    buffer = newBuffer;
    read++;
}
}

// Buffer is now too big. Shrink it.
byte[] ret = new byte[read];
Array.Copy(buffer, ret, read);
return ret;
}

```

[VB.NET]

```

Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click

    Dim request As HttpWebRequest =
CType(WebRequest.Create("http://www.nfpa.org/assets/files//PDF/Forms/EvacuationGuide.doc"), HttpWebRequest)
    Dim response1 As HttpWebResponse = CType(request.GetResponse(), HttpWebResponse)
    Dim stream As Stream = response1.GetResponseStream()
    Dim buffer As Byte() = ReadFully(stream, 32768)
    Dim ms As MemoryStream = New MemoryStream()
    ms.Write(buffer, 0, buffer.Length)
    ms.Seek(0, SeekOrigin.Begin)
    stream.Close()

    'Creating a new document.
    Dim doc As WordDocument = New WordDocument()
    'Open the template document from the filestream.
    doc.Open(ms, FormatType.Doc)
    doc.Save("Sample.doc", FormatType.Doc, response,

```

```

HttpContentDisposition.InBrowser)
End Sub
Public Shared Function ReadFully(ByVal stream As Stream, ByVal initialLength
As Integer) As Byte()
    ' If we've been passed an unhelpful initial length, just
    ' use 32K.
    If initialLength < 1 Then
        initialLength = 32768
    End If

    Dim buffer As Byte() = New Byte(initialLength - 1) {}
    Dim read As Integer = 0

    Dim chunk As Integer

    chunk = stream.Read(buffer, read, buffer.Length - read)
    Do While (chunk > 0)
        read += chunk

        ' If we've reached the end of our buffer, check to see if there's
        ' any more information
        If read = buffer.Length Then
            Dim nextByte As Integer = stream.ReadByte()

            ' End of stream? If so, we're done
            If nextByte = -1 Then
                Return buffer
            End If

            ' Nope. Resize the buffer, put in the byte we've just
            ' read, and continue
            Dim newBuffer As Byte() = New Byte(buffer.Length * 2 - 1) {}
            Array.Copy(buffer, newBuffer, buffer.Length)
            newBuffer(read) = CByte(nextByte)
            buffer = newBuffer
            read += 1
        End If
        chunk = stream.Read(buffer, read, buffer.Length - read)
    Loop

    ' Buffer is now too big. Shrink it.
    Dim ret As Byte() = New Byte(read - 1) {}
    Array.Copy(buffer, ret, read)
    Return ret
End Function

```

5.8 How to set the Page Size of the Document Section?

The page size of the document section is set by using the **PageSize** property of the **PageSetup** object. The following code snippet illustrates how to set this property.

[C#]

```
document.Sections[0].PageSetup.PageSize = PageSize.B6;
```

[VB .NET]

```
document.Sections(0).PageSetup.PageSize = PageSize.B6
```

5.9 How to set OpenType features?

The open type features provide special effects for text, in order to make them more refined and easier to read. This support is provided specifically for Word 2010 documents. Microsoft Word 2010 has new open type features for a font that supports these features, to make your documents look professional when printed. When font designers create fonts, they often add designs for special features.

The OpenType features include:

- Ligatures
- Use Contextual Alternates
- Number spacing
- Number forms
- Stylistic sets

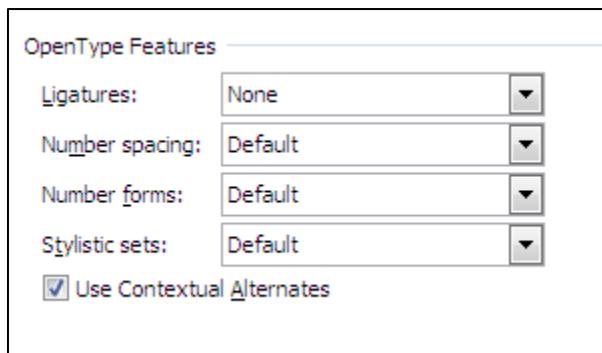


Figure 88: MS Word Open Type Features for Font

Ligatures

A ligature is a combination of characters that is written as a glyph, which is written as though it is a single character. Most often, ligatures are made up of pairs of letters. This enhances readability and attractiveness by providing the best ligature choice for the surrounding text. The OpenType standard specifies four categories of ligatures namely:

- **Standard** - The standard set of ligatures varies by language, but it contains the ligatures that most typographers and font designers agree are appropriate for that language.
- **Contextual** - Contextual ligatures are ligatures that were appropriate for use with that font, but they are not standard.
- **Historical and discretionary** - Historical forms are ligatures that were once standard but are no longer commonly used in the language. They can be used for a “period” effect. Discretionary ligatures are those that the font designer included for specific purposes. In general, you are more likely to want to apply historical or discretionary ligatures to just a portion of your text.
- **All** - All ligature combinations that are available for a font are applied to the text.

Check your fast facts on Fiji Islands

Figure 89: Text with Ligatures

Use Contextual Alternates

Contextual alternates provide fine-tuning of letters or combinations of letters, based on the surrounding characters. This feature can be used to make scripts look more natural and flowing. We can also use contextual alternates to provide specific letter forms at the start or end of a word, next to punctuations, or even at the end of a paragraph.

Contextual Alternates disabled - Text to check contextual alternates.

Contextual Alternates enabled - Text to check contextual alternates.

Figure 90: Contextual alternates

Number spacing

- **Default** - The default number spacing is specified by the font designer of each font.

- **Proportional** - Numbers are spaced more like letters with varying widths. For example, an 8 is wider than a 1. This spacing is easier to read in text.
- **Tabular** - Each number has the same width. This means that in a table column, for example, all three-digit numbers will align vertically. Tabular spacing is useful in Mathematics, which enhances readability for financial information.

Number forms

- **Default** - The default number form is specified by the font designer of each font.
- **Lining** - Lining numbers will apply the same height for all numbers and they don't extend below the baseline of the text. Lining numbers are easier to read in tables, boxes, or forms.
- **Old-style** - In Old-style numbering, the lines of the characters flow above or below the line of the text (it makes the numbers easier to read). For example, some numbers such as 3 and 5 extend below the baseline or are centered higher on the line.

Stylistic sets

Stylistic sets help users to change the look of the text by applying a different stylistic set. A font designer may include up to 20 stylistic sets for a font, and each stylistic set may include any subset of the characters of the font.

For example, the open type font Gabriola supports seven stylistic sets, each of which changes the formatting of text.

Default - *The quick red fox.*

Stylistic set 1 - *The quick red fox.*

Stylistic set 2 - *The quick red fox.*

Stylistic set 3 - *The quick red fox.*

Stylistic set 4 - *The quick red fox.*

Stylistic set 5 - *The quick red fox.*

Stylistic set 6 - *The quick red fox.*

Stylistic set 7 - *The quick red fox.*

Figure 91: Text with Stylistic Sets

The following code snippets illustrate how to set OpenType features:

[C#]

```
// Sets the contextual alternates.
text = paragraph.AppendText("Text to check contextual alternates.");
text.CharacterFormat.FontName = "Segoe Script";
text.CharacterFormat.UseContextualAlternates = true;

// Sets the historical discretionary ligatures.
text = paragraph.AppendText("Check your fast facts on Fiji Islands");
text.CharacterFormat.FontName = "Calibri";
text.CharacterFormat.Ligatures = LigatureType.HistoricalDiscretionary;

// Sets the.old style number format.
text = paragraph.AppendText("3457645");
text.CharacterFormat.FontName = "Calibri";
text.CharacterFormat.NumberForm = NumberFormType.OldStyle;

// Sets the.tabular type number spacing.
text = paragraph.AppendText("53127");
text.CharacterFormat.FontName = "Calibri";
text.CharacterFormat.NumberSpacing = NumberSpacingType.Tabular;

// Sets the.stylistic set option.
text = paragraph.AppendText("The quick red fox.");
text.CharacterFormat.FontName = "Gabriola";
text.CharacterFormat.StylisticSet = StylisticSetType.StylisticSet06;
```

[VB .NET]

```
' Sets the contextual alternates.
text = paragraph.AppendText("Text to check contextual alternates")
text.CharacterFormat.FontName = "Segoe Script"
text.CharacterFormat.UseContextualAlternates = True

' Sets the historical discretionary ligatures.
text = paragraph.AppendText("Check your fast facts on Fiji Islands")
text.CharacterFormat.FontName = "Calibri"
text.CharacterFormat.Ligatures = LigatureType.HistoricalDiscretionary

' Sets the.old style number format.
text = paragraph.AppendText("3457645")
text.CharacterFormat.FontName = "Calibri"
```

```

text.CharacterFormat.NumberForm = NumberFormType.OldStyle

' Sets the.tabular type number spacing.
text = paragraph.AppendText("53127")
text.CharacterFormat.FontName = "Calibri"
text.CharacterFormat.NumberSpacing = NumberSpacingType.Tabular

' Sets the.stylistic set option.
text = paragraph.AppendText("The quick red fox.")
text.CharacterFormat.FontName = "Gabriola"
text.CharacterFormat.StylisticSet = StylisticSetType.StylisticSet06

```

5.10 Does Essential DocIO provide support for Client profile?

Yes, Essential DocIO provides support for Client profile. In order to use Essential DocIO in an application (which is targeted to the Client profile), the user should include the following assemblies:

- Syncfusion.Core.dll
- Syncfusion.Compression.Base.dll
- Syncfusion.DocIO.ClientProfile.dll

5.11 How to insert section breaks?

Essential DocIO provides direct support to insert section breaks to Word documents. You can insert a section break to a Word document by using the InsertSectionBreak method of WParagraph class.

The following code snippets illustrate how to insert a section break:

[C#]

```

//Inserting a section break and it returns a newly created section.
WSection section = paragraph.InsertSectionBreak();

//Inserting a section break of the specified type and it returns a
newly created section.
WSection section =
paragraph.InsertSectionBreak(SectionBreakCode.EvenPage);

```

[VB]

```

'Inserting a section break and it returns a newly created section.
Dim section As WSection = paragraph.InsertSectionBreak()

```

```
'Inserting a section break of the specified type and it returns a newly  
created section.  
Dim section As WSection =  
paragraph.InsertSectionBreak(SectionBreakCode.EvenPage)
```

5.12 Migration from Microsoft Office Automation to Essential DocIO

This section covers information on the following topics:

5.12.1 Mail Merge

Mail merge feature can be used to generate reports and letters in MS word. The following code examples show how to generate an employee report from an MDB data source using office automation and DocIO.

Using MS Office Interop

Office automation performs the mail merge by executing a SQL query on the word document. The output of the mail merge can be sent to a new word document. Alternatively, it can be sent to a printer and a fax machine and forwarded to an e-mail address.

[C#]

```
using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object nullobject = Missing.Value;
object filepath = "EmployeesReportDemo.doc";
object sqlStmt = "SELECT * FROM [Employees]";
string sDBPath = "Northwind.mdb";
// Start the word application
word.Application wordApp = new word.Application();
// Open the word document
word.Document document = wordApp.Documents.Open(ref filepath, ref
nullobject, ref nullobject, ref nullobj
ref nullobj);
wordApp.Visible = false;
// Perform Mail Merge
document.MailMerge.OpenDataSource(sDBPath, ref nullobj
ref nullobj);
document.MailMerge.Execute(ref nullobj);
// Send output of Mail Merge to a new document
document.MailMerge.Destination =
word.WdMailMergeDestination.wdSendToNewDocument;
// Close the document
document.Close(ref nullobj, ref nullobj, ref nullobj);
// Quit the application
wordApp.Quit(ref nullobj, ref nullobj, ref nullobj);
```

[VB]

```

Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullObject As Object = Missing.Value
Dim DataBase As String = "Northwind.mdb"
Dim filePath As Object = "EmployeesReportDemo.doc"
Dim sqlStmt As String = "SELECT * FROM [Employees]"
Dim falseobj As Object = False

' Start the word application
Dim wordApp As word.Application = New word.Application()

' Open the word document
Dim doc As word.Document = wordApp.Documents.Open(filePath,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
falseobj, nullObject, nullObject, nullObject)

wordApp.Visible = False

' Perform Mail Merge
With doc.MailMerge
    .OpenDataSource(DataBase, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, sqlStmt,
nullObject, nullObject, nullObject)
    .Execute(nullObject)
    .Destination =
word.WdMailMergeDestination.wdSendToNewDocument
End With

' Close the document
doc.Close(nullObject, nullObject, nullObject)

' Quit the application
wordApp.Quit(nullObject, nullObject, nullObject)

```

Using DocIO

DocIO performs the mail merge using the *Execute ()* method as illustrated in the following example.

[C#]

```
string DataBase = "Northwind.mdb";

//Open existing template
WordDocument doc = new WordDocument("EmployeesReportDemo.doc",
FormatType.Doc);

// Get Data from the Database.
OleDbConnection conn = new
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
DataBase);
conn.Open();

// Populate data table
DataTable table = new DataTable();
OleDbDataAdapter adapter = new OleDbDataAdapter("select * from
employees", conn);
adapter.Fill(table);
adapter.Dispose();

// Perform Mail Merge
doc.MailMerge.Execute(table);

// Save the document
doc.Save("MailMerge.doc", FormatType.Doc);

// Close the document
doc.Close();
```

[VB]

```

Dim DataBase As String = "Northwind.mdb"

' Open the word document
Dim doc As WordDocument = New
WordDocument("EmployeesReportDemo.doc")

' Create database connection
Dim conn As OleDbConnection = New
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
DataBase)
conn.Open()

' Populate data table
Dim table As DataTable = New DataTable()
Dim adapter As OleDbDataAdapter = New OleDbDataAdapter("select * 
from employees", conn)
adapter.Fill(table)
adapter.Dispose()

' Perform Mail Merge
doc.MailMerge.Execute(table)

' Save the document
doc.Save("MailMerge.doc", FormatType.Doc)

' Close the document
doc.Close()

```



Note: For more information on performing the mail merge using DocIO, you can refer the online documentation link given below:

http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/46mailmerge.htm

5.12.2 Find and Replace

This section illustrates how to perform simple find and replace operation in a word document using MS Office Interop and DocIO.

Using MS Office Interop

The following code example shows how to search a word in a word document, replace it with another word, and save the document in a new name.

[C#]

```
using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object nullobject = Missing.Value;
object filepath = "FindAndReplaceTemplate.doc";
object newPath = "FindAndReplace.doc";
object item = word.WdGoToItem.wdGoToPage;
object whichItem = word.WdGoToDirection.wdGoToFirst;
object replaceAll = word.WdReplace.wdReplaceAll;
object forward = true;
object matchAllWord = true;
object matchCase = false;
object originalText = "Hello";
object replaceText = "World";
object save = true;

// Start the word application
word.Application wordApp = new word.Application();

// Open the word document
word.Document document = wordApp.Documents.Open(ref filepath, ref
nullobject, ref nullobject, ref nullobj, ref nullobj, ref
nullobj, ref nullobj, ref nullobj, ref nullobj, ref
nullobj, ref nullobj, ref nullobj, ref nullobj, ref
nullobj, ref nullobj, ref nullobj);

wordApp.Visible = false;

// Search for text and replace
document.GoTo(ref item, ref whichItem, ref nullobj, ref
nullobj);
foreach (word.Range rng in document.StoryRanges)
{
    rng.Find.Execute(ref originalText, ref matchCase, ref
matchAllWord, ref nullobj, ref nullobj, ref nullobj, ref
forward, ref nullobj, ref nullobj, ref replaceText, ref
replaceAll, ref nullobj, ref nullobj, ref nullobj, ref
nullobj);
}

// Save the document
document.SaveAs(ref newPath, ref nullobj, ref nullobj,
ref nullobj, ref nullobj, ref nullobj, ref nullobj,
ref nullobj, ref nullobj, ref nullobj, ref nullobj,
ref nullobj, ref nullobj, ref nullobj);

// Close the document
document.Close(ref nullobj, ref nullobj, ref nullobj);

// Quit the application
wordApp.Quit(ref nullobj, ref nullobj, ref nullobj);
```


[VB]

```

Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullObject As Object = Missing.Value
Dim filePath As Object = "FindAndReplaceTemplate.doc"
Dim newFilePath As Object = "FindAndReplace.doc"
Dim item As Object = word.WdGoToItem.wdGoToPage
Dim whichItem As Object = word.WdGoToDirection.wdGoToFirst
Dim replaceAll As Object = word.WdReplace.wdReplaceAll
Dim forward As Object = True
Dim matchAllWord As Object = True
Dim matchCase As Object = False
Dim originalText As Object = "Hello"
Dim replaceText As Object = "World"
Dim save As Object = True
Dim falseObj As Object = False

' Start the word application
Dim wordApp As word.Application = New word.Application()

' Open the word document
Dim doc As word.Document = wordApp.Documents.Open(filePath,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
falseObj, nullObject, nullObject, nullObject)

wordApp.Visible = False

' Search for text and replace
doc.GoTo(item, whichItem, nullObject, nullObject)

For Each rng As word.Range In doc.StoryRanges
    rng.Find.Execute(originalText, matchCase, matchAllWord,
nullObject, nullObject, nullObject, forward, nullObject,
nullObject, replaceText, replaceAll, nullObject, nullObject,
nullObject, nullObject)
Next

' Save the document
doc.SaveAs(newFilePath, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject)

' Close the document
doc.Close(nullObject, nullObject, nullObject)

' Quit the application
wordApp.Quit(nullObject, nullObject, nullObject)

```

Using DocIO

The following code example shows how to perform simple find and replace operation using DocIO.

[C#]

```
// Open the word document
WordDocument doc = new WordDocument("FindAndReplaceTemplate.doc",
FormatType.Doc);

// Define replacement text
string replaceText = "World";

// Perform replace
doc.Replace(new Regex("Hello"), replaceText);

// Save the document
doc.Save("Find And Replace.doc");

// Close the document
doc.Close();
```

[VB]

```
' Open the word document
Dim doc As WordDocument = New
WordDocument("FindAndReplaceTemplate.doc")

' Define text to be replaced
Dim replaceText As String = "World"

' Perform replace
doc.Replace(New Regex("Hello"), replaceText)

' Save the document
doc.Save("Find And Replace.doc")

' Close the document
doc.Close()
```

 **Note:** You can also refer the following online documentation link for more information on performing the find and replace operation using DocIO.
http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/45findandreplace.htm

5.12.3 Bookmarks

Bookmarks identify the location of text in a word document that you can name and identify for future reference. For example, you might use a bookmark to identify text that you want to revise later. Instead of scrolling through the document to locate the text, you can identify the text location by using the **Bookmark** dialog box.

Using MS Office Interop

The following code example shows how to insert a bookmark for a range of text using Office Automation.

[C#]

```
using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object nullobject = Missing.Value;
object newPath = "NewFile.doc";

// Start the word application
word.Application wordApp = new word.Application();

// Create a new word document
wordApp.Documents.Add(ref nullobject, ref nullobj, ref
nullobj, ref nullobj);
word.Document doc = wordApp.ActiveDocument;

// Add a paragraph to the document
word.Paragraph oParal;
oParal = doc.Content.Paragraphs.Add(ref nullobj);
oParal.Range.Text = "Bookmark with one word selected";

// Define start and end positions of bookmark range
object start = oParal.Range.Text.IndexOf("word");
object end = oParal.Range.Text.LastIndexOf(" ");
object rng = doc.Range(ref start, ref end);

// Add bookmark
doc.Bookmarks.Add("one_word", ref rng);

// Save the document and quit the application
doc.SaveAs(ref newPath, ref nullobj, ref nullobj, ref
nullobj, ref nullobj, ref nullobj, ref nullobj, ref
nullobj, ref nullobj, ref nullobj, ref nullobj, ref
nullobj, ref nullobj, ref nullobj, ref nullobj, ref
nullobj);

// Close the document
document.Close(ref nullobj, ref nullobj, ref nullobj);

// Quit the application
wordApp.Quit(ref nullobj, ref nullobj, ref nullobj);
```

[VB]

```

Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullobject As Object = Missing.Value
Dim newFilePath As Object = "NewFile.doc"

' Start the word application
Dim wordApp As word.Application = New word.Application()

' Create a new word document
wordApp.Documents.Add(nullobject, nullobject, nullobject,
nullobject)
Dim doc As word.Document = wordApp.ActiveDocument

' Add a paragraph to the document
Dim oPara As word.Paragraph
oPara = doc.Content.Paragraphs.Add(nullobject)
oPara.Range.Text = "Bookmark with one word selected"

' Define the start and end positions of bookmark range
Dim startobj As Object = oPara.Range.Text.IndexOf("word")
Dim endobj As Object = oPara.Range.Text.LastIndexOf(" ")
Dim rng As Object = doc.Range(startobj, endobj)

' Add bookmark
doc.Bookmarks.Add("one_word", rng)

' Save the document
doc.SaveAs(newFilePath)

' Close the document
doc.Close(nullobject, nullobject, nullobject)

' Quit application
wordApp.Quit()

```

Using DocIO

The following code example shows how to insert the bookmark using DocIO. Here, the *AppendBookmarkStart()* and *AppendBookmarkEnd()* methods are used to add the bookmark.

[C#]

```
// Create a new word document
WordDocument doc = new WordDocument();
IWSection section = doc.AddSection();
IWParagraph paragraph = section.AddParagraph();
paragraph.AppendText("Simple Bookmark");
paragraph = section.AddParagraph();
paragraph.AppendText("Bookmark with one ");

// Insert bookmark
paragraph.AppendBookmarkStart("one_word");
paragraph.AppendText("word");
paragraph.AppendBookmarkEnd("one_word");
paragraph.AppendText(" selected");

// Save the document
doc.Save("Bookmarks.doc");

// Close the document
doc.Close();
```

[VB]

```
' Create a new word document
Dim doc As WordDocument = New WordDocument()
Dim section As IWSection = doc.AddSection()
Dim paragraph As IWParagraph = section.AddParagraph()
paragraph.AppendText("Simple Bookmark")
paragraph = section.AddParagraph()
paragraph.AppendText("Bookmark with one ")

' Insert bookmark
paragraph.AppendBookmarkStart("one_word")
paragraph.AppendText("word")
paragraph.AppendBookmarkEnd("one_word")
paragraph.AppendText(" selected")

' Save the document
doc.Save("Bookmarks.doc")

' Close the document
doc.Close()
```

 **Note:** For more information on working with bookmarks using DocIO, you can refer the following online documentation link:

http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/4413bookmark.htm

5.12.4 Page numbers

Page numbers can be inserted to a word document in the header/footer section.

Using MS Office Interop

In the following code example, the page numbers are inserted to the footer of the word document by adding a page number field.

[C#]

```
using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object filepath = "NewFile.doc";
object nullobject = Missing.Value;
// Start the word application
word.Application wordApp = new word.Application();
// Open the word document
word.Document document = wordApp.Documents.Open(ref filepath, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject);
wordApp.Visible = false;
document.Activate();
// Seek the page footer
wordApp.ActiveWindow.ActivePane.View.SeekView =
word.WdSeekView.wdSeekCurrentPageFooter;
```

```
// Apply formatting for the footer  
  
wordApp.Selection.Paragraphs.Alignment =  
word.WdParagraphAlignment.wdAlignParagraphCenter;  
  
wordApp.ActiveWindow.Selection.Font.Name = "Arial";  
  
wordApp.ActiveWindow.Selection.Font.Size = 8;  
  
// Add page numbers in footer  
  
Object currentPage = word.WdFieldType.wdFieldPage;  
  
wordApp.ActiveWindow.Selection.Fields.Add(wordApp.Selection.Range  
, ref currentPage, ref nullobject, ref nullobject);  
  
  
// Save the document  
  
document.Save();  
  
// Close the document  
  
document.Close(ref nullobject, ref nullobject, ref nullobject);  
  
// Quit the application  
  
wordApp.Quit(ref nullobject, ref nullobject, ref nullobject);
```

[VB]

```

Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullobject As Object = Missing.Value
Dim filePath As Object = "NewFile.doc"
Dim falseobj As Object = False

' Start the application
Dim wordApp As word.Application = New word.Application()

' Add a new word document
Dim document As word.Document = wordApp.Documents.Open(filePath,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject, nullobject,
falseobj, nullobject, nullobject, nullobject, nullobject)
wordApp.Visible = False
document.Activate()

' Seek the page footer
wordApp.ActiveWindow.ActivePane.View.SeekView =
word.WdSeekView.wdSeekCurrentPageFooter

' Formatting for the footer
wordApp.Selection.Paragraphs.Alignment =
word.WdParagraphAlignment.wdAlignParagraphCenter
wordApp.ActiveWindow.Selection.Font.Name = "Arial"
wordApp.ActiveWindow.Selection.Font.Size = 8

' Add page numbers in footer
Dim currentPage As Object = word.WdFieldType.wdFieldTypePage
wordApp.ActiveWindow.Selection.Fields.Add(wordApp.Selection.Range
, currentPage, nullobject, nullobject)

' Save document
document.Save()

' Close the document
doc.Close(nullobject, nullobject, nullobject)

' Quit application
wordApp.Quit()

```

Using DocIO

DocIO provides support for adding page numbers to a word document. The page number field can be added to the header or footer of the word document. In the following code example, the page numbers are inserted to the footer of the word document.

[C#]

```
// Open the word document
WordDocument doc = new WordDocument("PageNumbers.docx",
FormatType.Docx);

// Add page number in footer for every section
foreach (WSection sec in doc.Sections)
{
    IWParagraph para = sec.AddParagraph();
    para.AppendField("footer", FieldType.FieldPage);
    para.ParagraphFormat.HorizontalAlignment =
HorizontalAlignment.Center;
    sec.PageSetup.PageNumberStyle = PageNumberStyle.Arabic;
    sec.HeadersFooters.Footer.Paragraphs.Add(para);
}

// Save the document
doc.Save("PageNumbersUpdated.docx", FormatType.Docx);

// Close the document
doc.Close();
```

[VB]

```

' Open the word document

Dim doc As WordDocument = New WordDocument("PageNumbers.docx",
FormatType.Docx)

' Add page number in footer for every section

For Each sec As WSection In doc.Sections

    Dim para As IWParagraph = sec.AddParagraph()

    para.AppendField("footer", FieldType.FieldPage)

    para.ParagraphFormat.HorizontalAlignment =
HorizontalAlignment.Center

    sec.PageSetup.PageNumberStyle = PageNumberStyle.Arabic

    sec.HeadersFooters.Footer.Paragraphs.Add(para)

Next

' Save the document

doc.Save("PageNumbersUpdated.docx", FormatType.Docx)

' Close the document

doc.Close()

```

5.12.5 Document Watermark

Watermark can be text or image which is used to mark the document as private or confidential and to write any text that informs about the usage and credibility of the document. In Microsoft Word, you can quickly insert a watermark using the Insert Watermark command.

Using MS Office Interop

The below code example illustrates how to insert a text watermark as a shape using Office Automation.

[C#]

```
using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object nullobject = Missing.Value;
object newPath = "NewFileWatermark.doc";

// Start the word application
word.Application wordApp = new word.Application();

// Create a new word document
wordApp.Documents.Add(ref nullobject, ref nullobject, ref
nullobject, ref nullobject);
word.Document doc = wordApp.ActiveDocument;

// Seek the current page header
wordApp.ActiveWindow.ActivePane.View.SeekView =
word.WdSeekView.wdSeekCurrentPageHeader;

// Insert watermark
word.Shape watermark =
wordApp.Selection.HeaderFooter.Shapes.AddTextEffect(Microsoft.Off
ice.Core.MsoPresetTextEffect.msoTextEffect1, "Watermark",
"Arial", (float)48, Microsoft.Office.Core.MsoTriState.msoTrue,
Microsoft.Office.Core.MsoTriState.msoFalse, 0, 0, ref
nullobject);

// Set watermark properties
watermark.Fill.Visible =
Microsoft.Office.Core.MsoTriState.msoTrue;
watermark.Line.Visible =
Microsoft.Office.Core.MsoTriState.msoFalse;
watermark.Fill.Solid();
watermark.Fill.ForeColor.RGB = (Int32)word.WdColor.wdColorGray30;

// Set focus back to the document
wordApp.ActiveWindow.ActivePane.View.SeekView =
word.WdSeekView.wdSeekMainDocument;

// Save the document
doc.SaveAs(ref newPath, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref
nullobject);

// Close the document
document.Close(ref nullobject, ref nullobject, ref nullobject);

// Quit the application
wordApp.Quit(ref nullobject, ref nullobject, ref nullobject);
```

[VB]

```
Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullobject As Object = Missing.Value
Dim newFilePath As Object = "NewFileWatermark.doc"

' Start the application
Dim wordApp As word.Application = New word.Application()

' Create a new word document
wordApp.Documents.Add(nullobject, nullobject, nullobject,
nullobject)
Dim doc As word.Document = wordApp.ActiveDocument

' Seek the current page header
wordApp.ActiveWindow.ActivePane.View.SeekView =
word.WdSeekView.wdSeekCurrentPageHeader

' Add text watermark to the document
Dim watermark As word.Shape =
wordApp.Selection.HeaderFooter.Shapes.AddTextEffect(Microsoft.Office.Core.MsoPresetTextEffect.msoTextEffect1, "Watermark",
"Arial", 48, Microsoft.Office.Core.MsoTriState.msoTrue,
Microsoft.Office.Core.MsoTriState.msoFalse, 0, 0, nullobject)

' Set watermark properties
watermark.Fill.Visible =
Microsoft.Office.Core.MsoTriState.msoTrue
watermark.Line.Visible =
Microsoft.Office.Core.MsoTriState.msoFalse
watermark.Fill.Solid()
watermark.Fill.ForeColor.RGB = CType(word.WdColor.wdColorGray30,
Integer)

' Save document
doc.SaveAs(newFilePath, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject)

' Close the document
doc.Close(nullobject, nullobject, nullobject)

' Quit application
wordApp.Quit()
```

Using DocIO

DocIO enables you to add a text watermark and a picture watermark to a word document. The following code example shows how to insert the picture watermark to the word document.

[C#]

```
// Create a new word document
WordDocument doc = new WordDocument();
doc.EnsureMinimal();

// Add picture watermark to the document
PictureWatermark picWatermark = new PictureWatermark();
picWatermark.Scaling = 120f;
picWatermark.Washout = true;
doc.Watermark = picWatermark;
picWatermark.Picture = Image.FromFile(ImagesPath + "Water lilies.jpg");

// Save the document
doc.Save("Watermark.doc", FormatType.Doc);

// Close the document
doc.Close();
```

[VB.NET]

```
' Create a new word document
Dim doc As WordDocument = New WordDocument()
doc.EnsureMinimal()

' Add picture watermark to the document
Dim picWatermark As PictureWatermark = New PictureWatermark()
picWatermark.Scaling = 120f
picWatermark.Washout = True
doc.Watermark = picWatermark
picWatermark.Picture = Image.FromFile(ImagesPath & "Water lilies.jpg")

' Save the document
doc.Save("Watermark.doc", FormatType.Doc)

' Close the document
doc.Close()
```

The following code example shows how to insert the text watermark to the word document.

[C#]

```
// Create a new word document
WordDocument doc = new WordDocument();

doc.EnsureMinimal();

// Add text watermark to the document

TextWatermark textWatermark = new TextWatermark();

doc.Watermark = textWatermark;

textWatermark.Size = 48;

textWatermark.Layout = WatermarkLayout.Horizontal;

textWatermark.Semitransparent = false;

textWatermark.Color = Color.Black;

textWatermark.Text = "Watermark";

// Save the document

doc.Save("Watermark.doc", FormatType.Doc);

// Close the document

doc.Close();
```

[VB]

```
' Create a new word document
Dim doc As WordDocument = New WordDocument()

doc.EnsureMinimal()

' Add text watermark to the document

Dim TextWatermark As TextWatermark = New TextWatermark()

doc.Watermark = TextWatermark

TextWatermark.Size = 48

TextWatermark.Layout = WatermarkLayout.Horizontal

TextWatermark.Semitransparent = False

TextWatermark.Color = Color.Black

TextWatermark.Text = "Watermark"

' Save the document

doc.Save("Watermark.doc", FormatType.Doc)

' Close the document

doc.Close()
```



Note: For more information on adding watermarks to a word document using DocIO, refer the following online documentation link:

http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/4222watermark.htm

5.12.6 Header/Footer

Headers and footers are displayed at the top and bottom of the document pages, respectively. The headers and footers can be inserted with text, graphics, and nearly any other information that is contained in the document.

Using MS Office Interop

The following code examples illustrate how to add headers and footers to a word document. In this example, page numbers are inserted to the header, and a text is inserted to the footer.

[C#]

```

using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object nullObject = Missing.Value;
object filePath = GetPath("original.doc");
object newFilePath = GetPath("HeaderFooterOffice.doc");

// Start the word application
word.Application wordApp = new word.Application();

// Open the word document
word.Document document = wordApp.Documents.Open(ref filePath, ref
nullObject, ref nullObject, ref nullObject, ref nullObject, ref
nullObject, ref nullObject, ref nullObject, ref nullObject, ref
nullObject, ref nullObject, ref nullObject, ref nullObject);

wordApp.Visible = false;

// Add header and footer for each section in the document
foreach (word.Section section in document.Sections)
{
    object fieldEmpty = word.WdFieldType.wdFieldPage;
    object autoText = "AUTOTEXT \\"Page X of Y\\" ";
    object preserveFormatting = true;

    // Footer
    section.Footers[word.WdHeaderFooterIndex.wdHeaderFooterPrimary].R
ange.Text = "Internal";

    section.Footers[word.WdHeaderFooterIndex.wdHeaderFooterPrimary].R
ange.ParagraphFormat.Alignment =
word.WdParagraphAlignment.wdAlignParagraphLeft;

    // Header
    section.Headers[word.WdHeaderFooterIndex.wdHeaderFooterPrimary].R
ange.Fields.Add(section.Headers[word.WdHeaderFooterIndex.wdHeader
FooterPrimary].Range, ref fieldEmpty, ref autoText, ref
preserveFormatting);

    section.Headers[word.WdHeaderFooterIndex.wdHeaderFooterPrimary].R
ange.ParagraphFormat.Alignment =
word.WdParagraphAlignment.wdAlignParagraphRight;
}

// Save the document
document.SaveAs(ref newFilePath, ref nullObject, ref nullObject,
ref nullObject, ref nullObject, ref nullObject, ref nullObject,
ref nullObject, ref nullObject, ref nullObject, ref nullObject,
ref nullObject);

// Close the document
document.Close(ref nullObject, ref nullObject, ref nullObject);

```


[VB]

```

Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullObject As Object = System.Reflection.Missing.Value
Dim filePath As Object = GetPath("original.doc")
Dim newFilePath As Object = GetPath("HeaderFooterOffice.doc")

' Start the application
Dim wordApp As word.Application = New word.Application()

' Open the document
Dim document As word.Document = wordApp.Documents.Open(filePath,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject)

wordApp.Visible = False

' Add header and footer for each section in the document
For Each section As word.Section In document.Sections
    Dim fieldEmpty As Object = word.WdFieldType.wdFieldPage

    ' Footer
    section.Footers(word.WdHeaderFooterIndex.wdHeaderFooterPrimary).R
ange.Text = "Internal"

    section.Footers(word.WdHeaderFooterIndex.wdHeaderFooterPrimary).R
ange.ParagraphFormat.Alignment =
word.WdParagraphAlignment.wdAlignParagraphLeft

    ' Header
    section.Headers(word.WdHeaderFooterIndex.wdHeaderFooterPrimary).R
ange.Fields.Add(section.Headers(word.WdHeaderFooterIndex.wdHeader
FooterPrimary).Range, fieldEmpty, nullObject, nullObject)

    section.Headers(word.WdHeaderFooterIndex.wdHeaderFooterPrimary).R
ange.ParagraphFormat.Alignment =
word.WdParagraphAlignment.wdAlignParagraphRight

Next

' Save the document
document.SaveAs(newFilePath, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject,
nullObject, nullObject, nullObject, nullObject)

' Close the document
doc.Close(nullObject, nullObject, nullObject)

' Quit the application
wordApp.Quit()

```

Using DocIO

You can set the header and footer by using the **HeadersFooters** property in the word document's section. To access a particular header/footer, you can use the following properties of the **WHeadersFooters** class:

- FirstPageHeader
- FirstPageFooter
- OddHeader
- OddFooter
- EvenHeader
- EvenFooter

[C#]

```
// Open the word document
WordDocument doc = new WordDocument("original.doc");

// Add header and footer for each section in the document

foreach (WSection sec in doc.Sections)

{

    // Header

    WParagraph para = new WParagraph(doc);

    para.AppendField("page", FieldType.FieldPage);

    para.ParagraphFormat.HorizontalAlignment =
HorizontalAlignment.Right;

    sec.HeadersFooters.Header.Paragraphs.Add(para);

    // Footer

    WParagraph para1 = new WParagraph(doc);

    para1.AppendText("Internal");

    para1.ParagraphFormat.HorizontalAlignment =
HorizontalAlignment.Left;

    sec.HeadersFooters.Footer.Paragraphs.Add(para1);

}

// Save the document

doc.Save("HeaderFooterDocIO.doc", FormatType.Doc);

// Close the document

doc.Close();
```

[VB]

```

' Open the word document
Dim doc As WordDocument = New WordDocument("original.doc")

' Add header and footer for each section in the document

For Each sec As WSection In doc.Sections

    ' Header

    Dim para As WParagraph = New WParagraph(doc)

    para.AppendField("page", FieldType.FieldPage)

    para.ParagraphFormat.HorizontalAlignment =
HorizontalAlignment.Right

    sec.HeadersFooters.Header.Paragraphs.Add(para)

    ' Footer

    Dim para1 As WParagraph = New WParagraph(doc)

    para1.AppendText("Internal")

    para1.ParagraphFormat.HorizontalAlignment =
HorizontalAlignment.Left

    sec.HeadersFooters.Footer.Paragraphs.Add(para1)

Next

' Save the document

doc.Save("HeaderFooterDocIO.doc", FormatType.Doc)

' Close the document

doc.Close()

```



Note: For more information on inserting Headers and Footers to a word document using DocIO, refer the online documentation link below:

http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/432headersandfooters.htm

5.12.7 Character Formatting

Character formatting defines the appearance of the text in a word document. This section illustrates how to apply character level formatting to the word document.

Using MS Office Interop

Character formatting can be applied to a word document using Office Automation. You need to set the range of text to which the formatting is to be applied. The following example illustrates how to apply the character formatting to the word document using the *Range* properties.

[C#]

```
using word = Microsoft.Office.Interop.Word;  
-----  
// Initialize objects  
object nullobject = System.Reflection.Missing.Value;  
  
object newPath = "CharacterFormattingOffice.doc";  
  
object falseObj = false;  
  
// Start the word application  
  
word.Application wordApp = new word.Application();  
  
// Create a new word document  
  
wordApp.Documents.Add(ref nullobject, ref nullobject, ref  
nullobject, ref nullobject);  
  
word.Document doc = wordApp.ActiveDocument;  
  
// Define range to which formatting to be applied  
  
object start = 0;  
  
object end = 0;  
  
word.Range rng = doc.Range(ref start, ref end);  
  
rng.Text = "New Text";  
  
rng.Font.Name = "Arial";  
  
rng.Font.Size = 14;  
  
// Save the document  
  
doc.SaveAs(ref newPath, ref nullobject, ref nullobject, ref  
nullobject, ref nullobject, ref nullobj, ref nullobj, ref  
nullobj, ref nullobj, ref nullobj, ref nullobj, ref  
nullobj, ref nullobj, ref nullobj, ref nullobj, ref  
nullobj);  
  
// Close the document  
  
document.Close(ref nullobj, ref nullobj, ref nullobj);  
  
// Quit the application  
  
wordApp.Quit(ref nullobj, ref nullobj, ref nullobj);
```

[VB]

```

Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullobject As Object = System.Reflection.Missing.Value
Dim newFilePath As Object = "CharacterFormattingOffice.doc"
Dim falseObj As Object = False

' Start the word application
Dim wordApp As word.Application = New word.Application()

' Create a new word document
wordApp.Documents.Add(nullobj, nullobj, nullobj,
nullobj)
Dim doc As word.Document = wordApp.ActiveDocument

' Define range to which formatting to be applied
Dim start As Object = 0
Dim endobj As Object = 0
Dim rng As word.Range = doc.Range(start, endobj)

rng.Text = "New Text"
rng.Font.Name = "Arial"
rng.Font.Size = 14

' Save the document
doc.SaveAs(newFilePath, nullobj, nullobj, nullobj,
nullobj, nullobj, nullobj, nullobj, nullobj,
nullobj, nullobj, nullobj, nullobj, nullobj,
nullobj, nullobj)

' Close the document
doc.Close(nullobj, nullobj, nullobj)

' Quit the application
wordApp.Quit()

```

Using DocIO

The following code examples illustrate how to apply the character formatting to the word document using DocIO.

[C#]

```
// Create a new word document  
  
WordDocument doc = new WordDocument();  
  
IWSection section = doc.AddSection();  
  
IWParagraph para = section.AddParagraph();  
  
// Apply formatting  
  
IWTextRange range = para.AppendText("New Text");  
range.CharacterFormat.FontName = "Arial";  
range.CharacterFormat.FontSize = 14;  
  
// Save the document  
  
doc.Save("CharacterFormattingDocIO.doc", FormatType.Doc);
```

[VB]

```
' Create a new word document

Dim doc As WordDocument = New WordDocument()

Dim section As IWSection = doc.AddSection()

Dim para As IWParagraph = section.AddParagraph()

' Apply formatting

Dim range As IWTextRange = para.AppendText("New Text")

range.CharacterFormat.FontName = "Arial"

range.CharacterFormat.FontSize = 14

' Save the document

doc.Save("CharacterFormattingDocIO.doc", FormatType.Doc)
```

5.12.8 Tables

Tables are used to organize information and to display the information with rows and columns. You can also add images or even other tables to the table.

Using MS Office Interop

The following code example illustrates how to insert a table to a word document, where the table contains three rows and two columns.

[C#]

```
using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object nullobject = System.Reflection.Missing.Value;
object newPath = "TableOffice.doc";

// Start the word application
word.Application wordApp = new word.Application();

// Create a new document
wordApp.Documents.Add(ref nullobject, ref nullobject, ref
nullobject, ref nullobject);
word.Document doc = wordApp.ActiveDocument;

// Insert table
object start = 0;
object end = 0;
word.Range tableLocation = doc.Range(ref start, ref end);
doc.Tables.Add(tableLocation, 3, 2, ref nullobject, ref
nullobject);

// Save the document
doc.SaveAs(ref newPath, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject);

// Close the document
document.Close(ref nullobject, ref nullobject, ref nullobject);

// Quit the application
wordApp.Quit(ref nullobject, ref nullobject, ref nullobject);
```

[VB]

```
Imports word = Microsoft.Office.Interop.Word

-----
' Initialize objects

Dim nullobject As Object = System.Reflection.Missing.Value
Dim newFilePath As Object = "TableOffice.doc"

' Start the word application

Dim wordApp As word.Application = New word.Application()

' Create a new document

wordApp.Documents.Add(nullobject, nullobject, nullobj
ect, nullobject)

Dim doc As word.Document = wordApp.ActiveDocument

' Insert table

Dim start As Object = 0
Dim endobj As Object = 0
Dim tableLocation As word.Range = doc.Range(start, endobj)
doc.Tables.Add(tableLocation, 3, 2, nullobject, nullobject)

' Save the document

doc.SaveAs(newFilePath, nullobject, nullobject, nullobj
ect, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject)

' Close the document

doc.Close(nullobject, nullobject, nullobject)

' Quit the application

wordApp.Quit()
```

Using DocIO

The below code example shows how to insert an empty table to a word document. The `ResetCells()` method is used to specify the number of rows and columns present in the table.

[C#]

```
// Create a new word document
IWordDocument doc = new WordDocument();
IWSection section = doc.AddSection();

// Add a table to the document
IWTable table = section.AddTable();
table.ResetCells(3, 2);

// Save the document
doc.Save("TableDocIO.doc");
```

[VB]

```
' Create a new word document

Dim doc As IWordDocument = New WordDocument()

Dim section As IWSection = doc.AddSection()

' Add a table to the document

Dim table As IWTable = section.AddTable()

table.ResetCells(3, 2)

' Save the document

doc.Save("TableDocIO.doc")
```



Note: For more information on creating tables using DocIO, refer the following online documentation link:

http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/433table.htm

5.12.9 Comments

Comments are used to include some additional information to a paragraph or text in a word document. Comments can be added and modified whenever needed and deleted when the comment has served its purpose. The comments are very useful when a document is reviewed. You can insert or delete the comments using DocIO and Office Automation.

Adding Comments Using MS Office Interop

The following code examples illustrate how to add comments to a word document. You need to define the range of text to which the comment is to be added.

[C#]

```
using word = Microsoft.Office.Interop.Word;  
-----  
// Initialize objects  
  
object nullobject = System.Reflection.Missing.Value;  
object newPath = "CommentOffice.doc";  
  
// Start the word application  
  
word.Application wordApp = new word.Application();  
  
// Create a new document  
  
wordApp.Documents.Add(ref nullobject, ref nullobject, ref  
nullobject, ref nullobject);  
  
word.Document doc = wordApp.ActiveDocument;  
  
// Insert text to the word document  
  
object start = 0;  
  
object end = 0;  
  
word.Range rng = doc.Range(ref start, ref end);  
  
rng.Text = "New Text";  
  
// Add comment to the inserted text  
  
object text = "Comment goes here";  
  
doc.Comments.Add(rng, ref text);  
  
// Save the document  
  
doc.SaveAs(ref newPath, ref nullobject, ref nullobject, ref  
nullobject, ref nullobject, ref nullobject, ref nullobject, ref  
nullobject, ref nullobject, ref nullobject, ref nullobject, ref  
nullobject, ref nullobject, ref nullobject, ref nullobject);
```


[VB]

```
Imports word = Microsoft.Office.Interop.Word  
-----  
' Initialize objects  
  
Dim nullobject As Object = System.Reflection.Missing.Value  
Dim newFilePath As Object = GetPath("CommentOffice.doc")  
  
' Start the word application  
  
Dim wordApp As word.Application = New word.Application()  
  
' Create a new document  
  
wordApp.Documents.Add(nullobject, nullobject, nullobject,  
nullobject)  
  
Dim doc As word.Document = wordApp.ActiveDocument  
  
' Insert text to the word document  
  
Dim startobj As Object = 0  
Dim endobj As Object = 0  
  
Dim rng As word.Range = doc.Range(startobj, endobj)  
rng.Text = "New Text"  
  
' Add comment to the inserted text  
  
Dim text As Object = "Comment goes here"  
doc.Comments.Add(rng, text)  
  
' Save document and quit application  
  
doc.SaveAs(newFilePath, nullobject, nullobject, nullobject,  
nullobject, nullobject, nullobject, nullobject,  
nullobject, nullobject, nullobject, nullobject,  
nullobject, nullobject)
```

Adding Comments Using DocIO

You can insert the comments to a paragraph or text in a word document using DocIO. The following code examples show how to insert the comments to the word document.

[C#]

```
// Create a new word document
WordDocument doc = new WordDocument();
IWSection section = doc.AddSection();

// Add a paragraph to the document
IWParagraph para = section.AddParagraph();
para.AppendText("New Text");

// Add comment to the paragraph
para.AppendComment("Comment goes here");

// Save the document
doc.Save("CommentDocIO.doc", FormatType.Doc);
```

[VB]

```
' Create a new word document

Dim doc As WordDocument = New WordDocument()

Dim section As IWSection = doc.AddSection()

' Add a paragraph to the document

Dim para As IWParagraph = section.AddParagraph()

para.AppendText("New Text")

para.AppendComment("Comment goes here")

' Save the document

doc.Save("CommentDocIO.doc", FormatType.Doc)
```

Removing Comments Using MS Office Interop

The comments can be removed from the word document when they have served their purpose. The following code examples show how to remove all comments in the word document using Office Automation.

[C#]

```
using word = Microsoft.Office.Interop.Word;  
-----  
// Initialize objects  
  
object nullobject = System.Reflection.Missing.Value;  
  
object filepath = "Comments.doc";  
  
object newPath = "CommentsRemovedOffice.doc";  
  
// Start the word application  
  
word.Application wordApp = new word.Application();  
  
// Open the word document  
  
word.Document document = wordApp.Documents.Open(ref filepath, ref  
nullobject, ref nullobject, ref nullobj  
ref nullobject, ref nullobject, ref nullobject, ref  
nullobject, ref nullobject, ref nullobject, ref  
nullobject, ref nullobject, ref nullobject);  
  
wordApp.Visible = false;  
  
// Delete all comments  
  
document.DeleteAllComments();  
  
// Save the document  
  
document.SaveAs(ref newPath, ref nullobject, ref nullobject,  
ref nullobject, ref nullobject, ref nullobject,  
ref nullobject, ref nullobject, ref nullobject,  
ref nullobject, ref nullobject, ref nullobject,  
ref nullobject);  
  
// Close the document  
  
document.Close(ref nullobject, ref nullobject, ref nullobject);
```

[VB]

```
Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullobject As Object = System.Reflection.Missing.Value
Dim filepath As Object = "Comments.doc"
Dim newFilePath As Object = "CommentsRemovedOffice.doc"

' Start the word application
Dim wordApp As word.Application = New word.Application()

' Open the word document
Dim document As word.Document = wordApp.Documents.Open(filepath,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject)
wordApp.Visible = False

' Delete all comments
document.DeleteAllComments()

' Save the document
document.SaveAs(newFilePath, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject, nullobject)

' Close the document
doc.Close(nullobject, nullobject, nullobject)

' Quit the application
wordApp.Quit()
```

Removing Comments Using DocIO

The *CommentsCollection* property holds all the comments present in the word document. The *Clear()* method can be called to remove all the comments. The following code examples illustrate how to remove the comments from the word document using DocIO.

[C#]

```
// Open the word document  
  
WordDocument doc = new WordDocument("Comments.doc");  
  
// Get all comments in the document  
  
CommentsCollection comments = doc.Comments;  
  
// Remove all comments from the document  
  
comments.Clear();  
  
// Save the document  
  
doc.Save("CommentsRemovedDocIO.doc", FormatType.Doc);
```

[VB]

```
' Open the word document  
Dim doc As WordDocument = New WordDocument("Comments.doc")  
  
' Get all comments in the document  
Dim Comments As CommentsCollection = doc.Comments  
  
' Remove all comments from the document  
Comments.Clear()  
  
' Save the document  
doc.Save("CommentsRemovedDocIO.doc", FormatType.Doc)
```

 **Note:** For more information on working with the comments using DocIO, you can refer the following online documentation link:

http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/44143comment.htm

5.12.10 Document Protection

You can protect word documents with or without a password to prevent anyone from accidentally or deliberately modifying the word documents. You can specify the protection type for preserving the word documents.

Using MS Office Interop

WdProtectionType property is used to specify the type of protection that can be set to the word document. This property uses the following values:

- wdAllowOnlyComments - Allow only comments to be added to the document.
- wdAllowOnlyFormFields - Allow content to be added to the document only through form fields.
- wdAllowOnlyReading - Allow read-only access to the document.
- wdAllowOnlyRevisions - Allow only revisions to be made to the existing content.
- wdNoProtection - Do not apply protection to the document.

[C#]

```
using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object nullobject = System.Reflection.Missing.Value;
object filepath = "Document.doc";
object newFilePath = "FileProtectionOffice.doc";
object noReset = false;
object password = System.String.Empty;
object useIRM = false;
object enforceStyleLock = false;

// Start the word application
word.Application wordApp = new word.Application();

// Open the word document that is to be protected
word.Document doc = wordApp.Documents.Open(ref filepath, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject);

wordApp.Visible = false;

// Set "Allow only Comments" protection to word document
doc.Protect(word.WdProtectionType.wdAllowOnlyComments, ref
noReset, ref password, ref useIRM, ref enforceStyleLock);

// Save the document
doc.SaveAs(ref newFilePath, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject);

// Close the document
document.Close(ref nullobject, ref nullobject, ref nullobject);

// Quit the application
wordApp.Quit(ref nullobject, ref nullobject, ref nullobject);
```

[VB]

```
Imports word = Microsoft.Office.Interop.Word

' Initialize objects

Dim nullobject As Object = System.Reflection.Missing.Value

Dim filepath As Object = "Document.doc"

Dim newFilePath As Object = "FileProtectionOffice.doc"

Dim noReset As Object = False

Dim password As Object = System.String.Empty

Dim useIRM As Object = False

Dim enforceStyleLock As Object = False

' Start the word application

Dim wordApp As word.Application = New word.Application()

' Open the word document that is to be protected

Dim doc As word.Document = wordApp.Documents.Open(filepath,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject)

wordApp.Visible = False

' Set "Allow only Comments" protection to word document

doc.Protect(word.WdProtectionType.wdAllowOnlyComments, noReset,
password, useIRM, enforceStyleLock)

' Save the document

doc.SaveAs(newFilePath, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject)
```

Using DocIO

DocIO uses `ProtectionType` property to specify the protection type to the word document. This property uses the following values:

- `AllowOnlyComments` - Allow only comments to be added to the document.
- `AllowOnlyFormFields` - Allow content to be added to the document only through form fields.
- `AllowOnlyRevisions` - Allow only revisions to be made to existing content.
- `NoProtection` - Do not apply protection to the document.

[C#]

```
// Open the word document

WordDocument doc = new WordDocument("Document.doc");

// Set "Allow only Comments" protection to word document
doc.Protect(ProtectionType.AllowOnlyComments, "");

// Save the document
doc.Save("FileProtectionDocIO.doc");
```

[VB]

```
' Open the word document  
  
Dim doc As WordDocument = New WordDocument("Document.doc")  
  
' Set "Allow only Comments" protection to word document  
doc.Protect(ProtectionType.AllowOnlyComments, "")  
  
' Save the document  
doc.Save("FileProtectionDocIO.doc")
```

Refer the following online documentation link for more details about how to protect the word documents using DocIO:

http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/47security.htm

5.12.11 Table of Contents

Table of contents can be generated by applying the heading styles to text in a word document. To create the table of contents in MS Word, click **Table of Contents** from the **Table of Contents** group on the **References** tab.

Using MS Office Interop

The following code example shows how to insert and update table of contents in a word document.

[C#]

```

using word = Microsoft.Office.Interop.Word;
-----
// Initialize objects
object nullobject = System.Reflection.Missing.Value;
object filepath = "TOCDocument.doc";
object newPath = "TOCUpdatedOffice.doc";
object trueobj = true;

// Start the word application
word.Application wordApp = new word.Application();

// Open the word document
word.Document document = wordApp.Documents.Open(ref filepath, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref nullobject, ref nullobject);

wordApp.Visible = false;

// Define range for TOC in document
object tocstart = 0;
object tocend = 0;
word.Range rngToc = document.Range(ref tocstart, ref tocend);

// Add TOC
word.TableOfContents toc = document.TablesOfContents.Add(rngToc,
ref trueobj, ref nullobject, ref nullobject, ref nullobject, ref
nullobject, ref trueobj, ref trueobj, ref trueobj, ref trueobj,
ref trueobj, ref trueobj);

// Update TOC
toc.Update();

// Save the document
document.SaveAs(ref newPath, ref nullobject, ref nullobject,
ref nullobject, ref nullobject, ref nullobject, ref nullobject,
ref nullobject, ref nullobject, ref nullobject, ref nullobject,
ref nullobject, ref nullobject, ref nullobject, ref nullobject,
ref nullobject);

// Close the document
document.Close(ref nullobject, ref nullobject, ref nullobject);

// Quit the application
wordApp.Quit(ref nullobject, ref nullobject, ref nullobject);

```

[VB]

```
Imports word = Microsoft.Office.Interop.Word
-----
' Initialize objects
Dim nullobject As Object = System.Reflection.Missing.Value
Dim filepath As Object = "TOCDocument.doc"
Dim newFilePath As Object = "TOCUpdatedOffice.doc"
Dim trueobj As Object = True

' Start the application
Dim wordApp As word.Application = New word.Application()

' Open the document
Dim document As word.Document = wordApp.Documents.Open(filepath,
nullobject, nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobject, nullobject,
nullobject, nullobject, nullobj)

wordApp.Visible = False

' Define range for TOC in document
Dim tocstart As Object = 0
Dim tocend As Object = 0
Dim rngToc As word.Range = document.Range(tocstart, tocend)

' Add TOC
Dim TOC As word.TableOfContents =
document.TablesOfContents.Add(rngToc, trueobj, nullobj,
nullobj, nullobj, trueobj, trueobj, trueobj,
trueobj, trueobj, trueobj)

' Update TOC
TOC.Update()

' Save the document
document.SaveAs(newFilePath, nullobj, nullobj, nullobj,
nullobj, nullobj, nullobj, nullobj, nullobj,
nullobj, nullobj, nullobj, nullobj)

' Close the document
doc.Close(nullobj, nullobj, nullobj)

' Quit the application
wordApp.Quit()
```

Using DocIO

The following code example illustrates how to insert and update the table of contents in a word document using DocIO.

[C#]

```
// Open the word document
WordDocument doc = new WordDocument("TOCDocument.doc",
FormatType.Doc);
IWSection sec = doc.Sections[0];

// Append TOC to the first paragraph of the document
WParagraph para = new WParagraph(doc);
TableOfContent toc = para.AppendTOC(1, 3);
sec.Paragraphs.Insert(0, para);

// Update table of contents
doc.UpdateTableOfContents();

// Save the document
doc.Save("TOCUpdatedDocIO.doc", FormatType.Doc);
```

[VB]

```
' Open the word document
Dim doc As WordDocument = New WordDocument("TOCDocument.doc",
FormatType.Doc)
Dim sec As IWSection = doc.Sections(0)

' Append TOC to the first paragraph of the document
Dim para As WParagraph = New WParagraph(doc)
Dim TOC As TableOfContent = para.AppendTOC(1, 3)
sec.Paragraphs.Insert(0, para)

' Update table of contents
doc.UpdateTableOfContents()

' Save the document
doc.Save("TOCUpdatedDocIO.doc", FormatType.Doc)
```

Refer the following online documentation link for more information about adding the table of contents to the word document using DocIO:

http://help.syncfusion.com/Ug_101/Reporting/DocIO/Windows%20Forms/default.htm#!documents/4418tableofcontents.htm

5.13 How to Attach a Template to a Word Document?

You can use the **AttachedTemplate** property to attach a template to a Word document using DocIO. The following code example illustrates this.

[C#]

```
//Set the location of the template
document.AttachedTemplate.Path = @"D:\Test.dot";

//Set the UpdateStylesOnOpen to 'true' to automatically update the
//styles from the attached template each time the document is opened with
//MS Word
document.UpdateStylesOnOpen = true;
```

[VB]

```
//Set the location of the template document
document.AttachedTemplate.Path = @"D:\Test.dot"

//Set the UpdateStylesOnOpen to 'true' to automatically update the
//styles from the attached template each time the document is opened with
//MS Word
document.UpdateStylesOnOpen = true
```

Index

A

Accessing Styles 205
Additional Mail Merge Features 278
API Changes 55
ASP.NET 35
ASP.NET MVC 45

B

Basic Concepts 56
Bookmark 163
Bookmark Navigator 168
Bookmarks 338
Break 193

C

Can Essential DocIO be used to read MS Word files? What are the MS Word versions that Essential DocIO can read? 317
Character Formatting 359
Character Styles and Formats 206
CheckBox 145
Class Diagram 23
Cloning and Merging 100
Comment 182
Comments 367
Concepts and Features 56
Content Control 126
Conversion 283
Creating Platform Application 25

D

Deploying Essential DocIO 30
Deployment Requirements 21
DLLs 21
Doc to EPub 299

Doc to HTML 286
Doc to PDF 290
Doc to RTF 285
Document Background 72
Document Color 73
Document Properties 66
Document Protection 376
Document Variable 158
Document Watermark 346
Documentation 9
Docx Support 83, 306
Does DocIO require MS Word to be installed on the report generation machine / server? 317
Does Essential DocIO provide support for Client profile? 328
DropDown 148
E
Embed Field 141
Encryption and Decryption 282
F
Feature Summary 53
Fields 135
Fields Updating Engine 161
Find 254
Find and Replace 248, 333
Font Substitution for Word Documents 92
Footnote and Endnote 187
Footnote and Endnote Separators 190
Form Field 143
Frequently Asked Questions 317
G
Getting Started 23
H
Header/Footer 352
Headers and Footers 103

How to Attach a Template to a Word Document?	Paragraph Item 130
384	Paragraph Styles 212
How to format a Hyperlink by using DocIO? 318	Picture 174
How to format a Table? 319	Prerequisites and Compatibility 8
How to insert section breaks? 328	Properties 190
How to modify the Built-in styles? 320	R
How to open a Document from Stream by using DocIO? 321	Replace 257
How to set OpenType features? 324	Row Format 122
How to set the Page Size of the Document Section? 324	RTF to Doc 296
Hyperlink 155	S
I	Samples and Location 11
Importing XHTML 241	Saving the Word Document 51
Installation 11	Section 94
Installation and Deployment 11	Security 281
Introduction to Essential DocIO 6	Seq Field 142
L	Shapes 173
List Format 230	Silverlight 41
Lists 222	Stream Support 87
M	Styles and Formatting 204
Macro-enabled Document Support 88	Supported Elements 309
Mail Merge 261, 329	Symbol 191
Mail Merge Events 272	T
Merge Field 139	Table 111
Migration from Microsoft Office Automation to Essential DocIO 329	Table Cell 117
N	Table Of Contents 196, 380
Nested Mail Merge 266	Table Row 115
O	Table Styles 124
OLE Object 199	Tables 363
Overview 6	Tabs 220
P	Text 150
Page numbers 342	Text Box Format 235
Paragraph 126	Text Range 132
Paragraph Formats 216	TextBodyPart 250
	TextBodySelection 252
	TextBox 178

TextSelection 249

Track Changes 90

W

Watermark 77

Windows 30

Word 2010 Support 86

Word Document 59

Would it be possible to view the generated report
[DOC] using Essential DocIO? 317

WPF 38