

Table of Contents

Document Solutions for Word Overview	4
Key Features	5-6
Getting Started	7-9
Quick Start	10-12
License Information	13-14
Technical Support	15
Contacting Sales	16
Redistribution	17
End-User License Agreement	18
Product Architecture	19-22
Features	23
Comments	24-26
Content Controls	27-31
Glossary Document	32
Document	33
Document Properties	34-36
Document Protection	37-42
Document Formatting Source	43-44
Copy or Move Document Content	45-47
Split or Merge Documents	48-49
Page Settings	50-52
Styles	53-56
Text Effect	57-60
Export	61-63
Fields	64-66
Footnote and Endnote	67-69
Embedded Fonts	70-74
Header and Footer	75-78
Images	79-83
Shapes	84
Textbox	85-87
Canvas Shape	88-89
Group Shape	90-91
Ink Shape	92

Shape	93
Shape Format	94-101
Picture	102
Picture Format	103-107
Online Video	108-109
Presets and Themed Styles	110-111
Shape Styles	112-122
Office Math	123-135
Complex Equation Examples	136-140
Links	141
Hyperlink	142-144
Bookmark	145-147
Lists	148-151
Paragraph	152
Paragraph	153-154
Indentation	155
Line Spacing	156
Borders	157-158
Background Shading	159
Tab Stops	160
Paragraph Numbering	161
Paragraph Styling	162
Range Objects	163-166
Sections	167-170
Table	171
Table	172-177
Cell	178-180
Row	181-182
Text	183-192
Themes	193-194
Helper Methods for Adding Content	195-200
Report Templates	201-207
Template Configuration	208
Template Tags	209-210
List	211-213
Table	214-219

Links	220-221
Loops and Nesting	222-223
Conditionally Hide Blocks	224-227
Calculation	228-231
Formatters	232-235
Restart List Formatter	236-237
Sequence and Follow Formatter	238-239
Block Behavior Formatters	240-241
Data Source Binding	242-247
Template Validation	248-249
Walkthrough	250
Generate Document with Advanced Layout	251-256
Generate Document using JSON Data Source	257-259
Generate Document using Multiple Data Sources	260-261
Samples	262
API Reference	263
Release Notes	264
Breaking Changes	265
Version 7.1.0	266
Version 7.0.0	267
Version 6.2.0	268
Version 6.1.0	269
Version 6.0.0	270
Version 5.2.0.800	271
Version 5.1.0.790	272
Version 5.0.0.762	273
Version 4.2.0.719	274
Version 4.2.0.715	275
Version 4.1.0.658	276
Version 4.0.0.616	277
Version 3.2.0.548	278
Version 3.1.0.508	279
Version 3.0.0.414	280
Version 2.2.0.310	281-283
Version 2.1.0.260	284

Document Solutions for Word Overview

Document Solutions is a cross-platform solution for document management which aims to provide a universal document, editor and viewer solution for all popular document formats.

Document Solutions for Word (DsWord, previously GcWord), is a part of Document Solutions that aims to be a complete solution to program and work with Word documents, without using any external word processor like MS Word. DsWord is a high performance library which is supported on .NET Standard 2.0 and can be used to create, load, modify, and save Word documents programmatically. It offers a rich and comprehensive object model which is simple to use and is based on Microsoft Office API, Word Javascript API and OpenXML SDK. It provides extensive set of features which allows developers to generate Word documents with formatted text, images, tables, hyperlinks, comments, headers, footers, footnotes, endnotes and more, and export Word documents to PDF.


Key Features

DsWord provides many different features that enable the developers to build intuitive and professional-looking Word documents. The main features for DsWord library are as follows:

- **Work with Word documents programmatically**
Using DsWord, you can programmatically create Word documents with simple or complex business requirements in .NET Standard applications without the help of an external word processor. You can also load, modify the Word documents from an external source and save them again.
- **High-Performance Library**
DsWord is a high performance library which is faster alternative to the Microsoft Word automation.
- **Work with Document content**
DsWord allows you to work with inbuilt document properties, text objects, range objects, and preserve macros. You can also copy or move document content and split or merge documents.
- **Document Formatting**
DsWord supports document formatting which includes character, paragraph, list, table, page, and section formatting. You can also detect source of formatting property in Word documents.
- **Advanced document protection**
DsWord lets you protect the contents of a document. DsWord provides properties to control the different types of modifications applied on Word documents.
- **Extensive support for Content controls**
DsWord library provides support for different types of content controls and their mappings. It also provides the flexibility to reuse your content by creating building blocks and maintaining their collection.
- **Supported file formats**
DsWord supports DOCX, DOTM, DOCM, DOTX file formats completely and can read files with FlatOPC, FlatOpcMacroEnabled, FlatOpcTemplate, and FlatOpcTemplateEnabled formats.
- **Export to PDF and Image Formats**
DsWord library allows you to export a Word document to PDF and Image file formats programmatically with just a single line of code. Even, right-to-left text, vertical text, East Asian languages, superscript, subscript etc. can also be exported without impacting their formats.
- **View Options**
DsWord library lets you set the view and zoom options to specify how a document appears when it is opened in an application.
- **Theme**
DsWord library allows you to customize themes applied on a Word document, hence giving it a consistent styling and professional look.
- **Rich set of features**
DsWord library provides a rich set of features that allow you to generate Word documents with content including formatted text and paragraphs, images, shapes, hyperlinks, bookmarks, comments, tables, lists, headers, footers, footnotes, endnotes, textbox and more.
- **Report Templates to Generate Word Documents**
DsWord provides Report Templates with comprehensive API to generate Word Documents with advanced

layouts. You can use various data sources to bind the data. The template layout provides flexible syntax, easy notations and extended reusability making it an ideal solution to generate desired documents.

For additional information about the supported features in DsWord, see [Features](#) topic.

 **Note:** Keep in mind that the following MS Word objects are not yet supported by DsWord object model. These objects are preserved in a load, modify, or save scenario but are not accessible via DsWord object model and are not parsed by DsWord except for removal.

- Charts & diagrams
- VML Objects
- Embedded objects, such as controls and OLE objects
- Revisions
- Ruby Phonetic Guide
- Sub documents references
- Picture Effects

Getting Started

System Requirements


The DsWord packages are fully supported on Visual Studio 2017 or later for Windows, Visual Studio for MAC, and Visual Studio Code for Linux and are compatible with the following:

- .NET 5, [.NET 6](#), and [.NET 7](#)
- .NET Core 2.x and 3.x
- .NET Standard 2.x
- .NET Framework 4.6.1 or higher

Setting up an Application

DsWord references are available through NuGet, a Visual Studio extension that adds the required libraries and references to your project automatically. To work with DsWord, you need to have following references in your application:

Reference	Purpose
DS.Documents.Word	To use DsWord in an application, you need to reference (install) just the DS.Documents.Word package. It pulls in the required infrastructure packages.
DS.Documents.Word.Layout	To enable saving Word documents to PDF, install the DS.Documents.Word.Layout package (DsWordLayout for short). It provides extension methods allowing to save DsWordDocument as PDF.
DS.Documents.Imaging	For image handling, you need to reference (install) the DS.Documents.Imaging package.
DS.Documents.DX.Windows	DS.Documents.DX.Windows is an infrastructure package. You do not need to reference it directly.

 **Note:** With v7.0, GrapeCity.Documents.Word (GcWord) package is renamed to DS.Documents.Word (DsWord). The namespaces and classes within DS.Documents.Word remain the same, which provide the same functionality and are backwards compatible with GrapeCity.Documents.Word, ensuring minimal impact on your existing projects.

To upgrade GcWord package to DsWord package in your existing projects, follow one of the below options:

- Update package using Migration tool:
 1. The migration tool is present in the package downloaded from the website. Follow the instructions displayed in the UI when using the tool for a seamless migration from GcWord to DsWord.
- Update package manually from NuGet package manager:
 1. In **Solution Explorer**, right-click either **Dependencies** or a project and select **Manage NuGet Packages**.
 2. In **Installed** tab, click on **GrapeCity.Documents.Word** package and click **Uninstall** to remove it and its dependencies from the project.
 3. In **Browse** tab, type "ds.documents" or "DS.Documents" in the search text box at the top and find the package "DS.Documents.Word".
 4. Click Install to add the **DS.Documents.Word** package and its dependencies into the project.

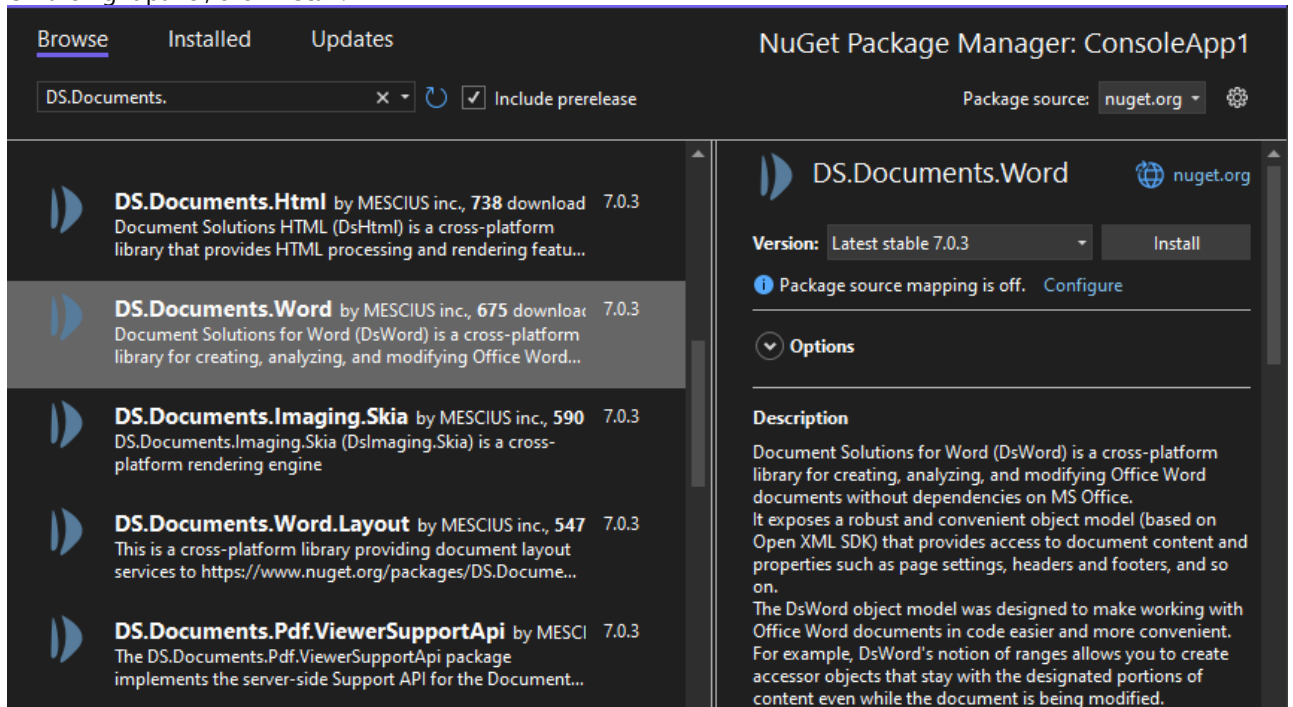
Add reference to DsWord in your application from nuget.org

In order to use DsWord in a .NET Core, ASP.NET Core, .NET Framework application (any target that supports .NET Standard 2.0), install the NuGet packages in your application using the following steps:

Visual Studio for Windows

1. Open Visual Studio.
2. Create any application (any target that supports .NET Standard 2.0).
3. Right-click the project in Solution Explorer and choose **Manage NuGet Packages**.

4. In the **Package source** on top right, select **nuget.org**.
5. Click **Browse** tab on top left and search for "DS.Documents".
6. On the left panel, select **DS.Documents.Word**.
7. On the right panel, click **Install**.



8. In the **Preview Changes** dialog, click **OK** and choose **I Accept** in the next screen.

Visual Studio for Mac

1. Open Visual Studio for Mac.
2. Create any application (any target that supports .NET Standard 2.0).
3. In tree view on the left, right-click **Dependencies** and choose **Add Packages**.
4. In the Search panel, type "DS.Documents".
5. From the list of packages displayed in the left panel, select **DS.Documents.Word** and click **Add Packages**.
6. Click **Accept**.

This automatically adds references of the package and its dependencies to your application.

Visual Studio Code for Linux

1. Open Visual Studio Code.
2. Install **Nuget Package Manager** from **Extensions**.
3. Create a folder "MyApp" in your **Home** folder.
4. In the Terminal in Visual Studio Code, type "cd MyApp"
5. Type command "dotnet new console"
Observe: This creates a .NETCore application with MyApp.csproj file and Program.cs.
6. Press **Ctrl+P**. A command line opens at the top.
7. Type command: ">"
Observe: "**Nuget Package Manager: Add Package**" option appears.
8. Click the above option.
9. Type "**DS**" and press Enter.
Observe: DS packages get displayed in the dropdown.
10. Choose **DS.Documents.Word**.
11. Type following command in the Terminal window: "dotnet restore"

This adds references of the package to your application.

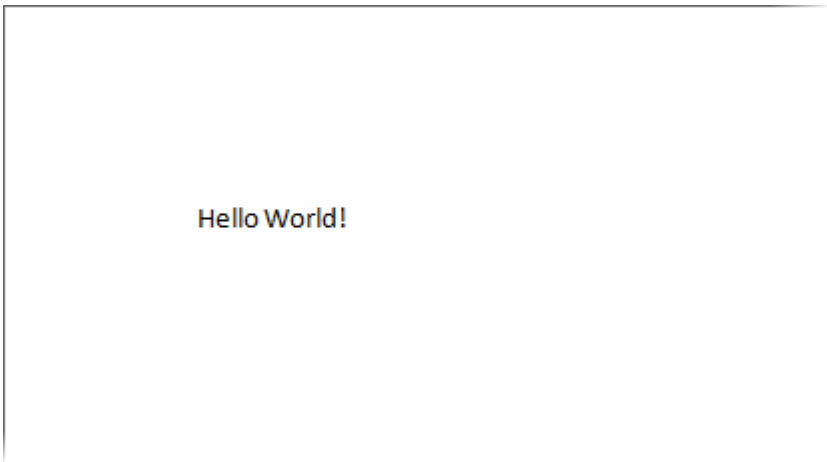
Quick Start

The following quick start sections help you in getting started with the DsWord library.

Create and Save a Word Document

This quick start covers how to create a simple Word document having a single page, add text to it and save it in a .NET Core or .NET Standard application. Follow the steps below to get started:

1. **Create a new DsWord document**
2. **Add text to the document**
3. **Save the document**



Step 1: Create a new DsWord document

1. Create a new application (.NET Core Console App\Windows Forms App) and install the DS.Documents.Word package to add the required dlls to the project. For detailed information on how to add the NuGet packages, see **Getting Started**.
2. Include the following namespace
 - using GrapeCity.Documents.Word;
3. Create a new Word document using an instance of the [GcWordDocument](#) class, through code.

```
C#  
  
// Create a new Word document:  
GcWordDocument doc = new GcWordDocument();
```

Back to Top


Step 2: Add text to the document

DsWord provides two methods to add text to the document:

- Access the first section from the body of the document using the [GetRange](#) method and add a paragraph to the paragraph collection using [Add](#) method of the [ParagraphCollection](#) class.
- Or, Add a paragraph to the paragraph collection using the [AddParagraph](#) method of [Body](#) class.

```
C#  
  
// Add a paragraph with the 'Hello, World!' text to the first section:  
doc.Body.Sections.First.GetRange().Paragraphs.Add("Hello World!");
```

```
// Or
doc.Body.AddParagraph("Hello World!");
```


 **Note:** DsWord provides various content elements helper methods in the classes to add different content elements directly, thus making the code shorter, clearer, and the content elements easily accessible. For more information, see [Helper Methods for Adding Content](#).

Back to Top

Step 3: Save the document

Save the document using [Save](#) method of the **DsWordDocument** class.

```
C#
//Save the created Word file
doc.Save("CreateDoc.docx");
```

 Note that the file is saved to the default location, which is the "bin/Debug" folder of the application.

Back to Top

Load and Modify a Word Document

This quick start covers how to load an existing Word document, modify it and save it using a .NET Core or .NET Standard application. Follow the steps below to get started:

1. **Load an existing document in DsWord**
2. **Modify the document**
3. **Save the document**

Hello World!

This document has been modified. A new paragraph is added to the document.

Step 1: Load an existing document in DsWord

1. Create a new application (.NET Core Console App\Windows Forms App) and install the DS.Documents.Word package to add the required dlls to the project. For detailed information on how to add the NuGet packages, see **Getting Started**.
2. Include the following namespace
 - o using GrapeCity.Documents.Word;
3. Create a new instance of the **DsWordDocument** class to load an existing document.

```
C#
```

```
// Create a new Word document:  
GcWordDocument doc = new GcWordDocument();
```

4. Load an existing document using [Load](#) method of the **DsWordDocument** class. In this example, the document to be loaded is placed in the "bin/Debug" folder of the application. In case you want to load a document from some other location in your system, you can update the location accordingly in the code.

```
C#  
doc.Load("SampleDoc.docx");
```

[Back to Top](#)

Step 2: Modify the document

To modify the document, access the first section from body of the document using the [GetRange](#) method and add a paragraph to the paragraph collection using [Add](#) method of the [ParagraphCollection](#) class.

```
C#  
// Add a new paragraph with the specific content, to the existing document  
Paragraph p = doc.Body.Sections.First.GetRange().Paragraphs.Add("This document" +  
    "has been modified. A new paragraph is added to the document.");
```

[Back to Top](#)

Step 3: Save the document

Save the document using the **Save** method.

```
C#  
//Save the modified Word file  
doc.Save("SampleDoc_Modified.docx", DocumentType.Document);
```

[Back to Top](#)

License Information

Types of Licenses


Document Solutions for Word supports the following types of license:

- **Unlicensed**
- **Evaluation License**
- **Licensed**

Unlicensed

After downloading, the product works in unlicensed mode. The following limitations are imposed when the product is used without license:

- Only 200 paragraphs in a Word file can be saved for analyzing.
- On saving the Word file, a text is displayed on the beginning of the first page of that file:
'Created with unlicensed copy of Document Solutions for Word. The document is limited to 200 paragraphs. Contact us.sales@mescius.com to get your 30-day evaluation key.'

 **Note:** Please note that the above text is inserted as the first paragraph into the document when it is saved, which changes the indices of other paragraphs in the saved file (but not in the current object model).

In case of PDF export, following page header is displayed on all the pages of the PDF file.

'Created with unlicensed copy of Document Solutions for Word. Contact us.sales@mescius.com to get your 30-day evaluation key.'

Evaluation License


DsWord evaluation license is available to users for 30 days to evaluate the product. If you want to evaluate the product, you can ask for the evaluation license key by sending an email to us.sales@mescius.com.

The evaluation version has an expiration date that is determined when an evaluation key is generated. After applying the evaluation license key, you can use the complete product until the license expiry date.

After the expiry date, the product works in unlicensed mode with the above mentioned limitations.

In such case, following watermark is displayed in the Word file:

'Created with expired evaluation copy of Document Solutions for Word. The document is limited to 200 paragraphs. Contact us.sales@mescius.com to purchase license.'

 **Note:** Please note that the above text is inserted as the first paragraph into the document when it is saved, which changes the indices of other paragraphs in the saved file (but not in the current object model).

On exporting a Word document to a PDF, following page header is displayed on all the pages of the PDF file:

'Created with expired evaluation copy of Document Solutions for Word. Contact us.sales@mescius.com to purchase license.'

Licensed

DsWord production license is issued at the time of purchase of the product. If you have a production license, you can access all the features of DsWord without any limitations.

Apply License

To apply evaluation/production license in DsWord, the long string key needs to be copied to code in one of the following two ways.

- Pass it as an argument to the GcWordDocument's ctor:

```
var doc = new GcWordDocument("key")
```

This licenses the instance being created.
- Call a static method on GcWordDocument:

```
GcWordDocument.SetLicenseKey("key");
```

This licenses all the instances while the program is running.

Technical Support

If you have a technical question about this product, consult the following sources:

- Product forum: <https://developer.mescius.com/forums>
- Email: us.sales@mescius.com


Contacting Sales

If you would like to find out more about our products, contact our Sales department using one of these methods:

World Wide Web site	https://developer.mescius.com/
E-mail	us.sales@mescius.com
Phone	(800) 858-2739 or (412) 681-4343 outside the U.S.A.
Fax	(412) 681-4384

Redistribution

To distribute an application containing the DsWord API, it is necessary to have a valid Distribution License and meet any [System Requirements](#).

 DsWord makes it easy to deploy your application to your local servers or cloud offerings such as Azure.

For more information about Distribution License, contact our Sales department using one of these methods:

World Wide Web site	https://developer.mescius.com/
E-mail	us.sales@mescius.com
Phone	1.800.858.2739 or 412.681.4343
Fax	(412) 681-4384

End-User License Agreement

The MESCIUS licensing information, including the MESCIUS end-user license agreement, frequently asked licensing questions, and the MESCIUS licensing model, is available online. For detailed information on licensing, see [MESCIUS Licensing](#). For MESCIUS end-user license agreement, see [End-User License Agreement For MESCIUS Software](#).

Product Architecture

Packaging

DsWord is a collection of cross-platform .NET class libraries written in C#, that provides API to create DOCX/DOCM MS Word files from scratch. The library also allows to load, analyze and modify existing Word documents.

DsWord is compatible with .NET Core 2.x/3.x, .NET Standard 2.x, .NET Framework 4.6.1 or higher, and .NET 6 or higher.

DsWord and supporting packages are available on nuget.org:

Package	Description
DS.Documents.Word	Main package which automatically pulls in the other required infrastructure packages.
DS.Documents.Word.Layout	Enables saving Word documents as PDF.
DS.Documents.Imaging	Provides image handling.
DS.Documents.DX.Windows	Provides access to the native graphics APIs when running on a Windows system.

Document Overview

A Word document in DsWord is represented by an instance of the [GrapeCity.Documents.Word.GcWordDocument](#) class.

The object model of the **GcWordDocument** class corresponds to the structure of a Word document, with the following properties corresponding to major parts of the document:

Property	Description
Body	The main document story
Styles	A collection of document styles to format document content
ListTemplates	A collection of list templates to format list content in the document
Settings	Provides options to control view, compatibility and other settings
Theme	Provides the different formatting options available to a document through a theme
CustomXMLParts	Provides the collection of CustomXMLPart objects.
GlossaryDocument	Provides the supplementary document storage which stores the content for future insertion.

Body

Body is the place where the **content elements** (representing the actual content of a document) are stored.

GcWordDocument.Body represents the main content of the document, but other parts of the document (such as headers/footers, comments, footnotes/endnotes) also have bodies to store their content, the specific body type is indicated by the [GrapeCity.Documents.Word.BodyType](#) enumeration, which has the following members:

Member	Description
Main	Body of main document part
Header	Body of section header

Footer	Body of section footer
Comment	Body of comment
BuildingBlock	Body of building block
Footnote	Body of footnote
FootnoteSeparator	Body of footnote separator
FootnoteContinuationSeparator	Body of footnote continuation separator
FootnoteContinuationNotice	Body of footnote continuation notice
Endnote	Body of endnote
EndnoteSeparator	Body of endnote separator
EndnoteContinuationSeparator	Body of endnote continuation separator
EndnoteContinuationNotice	Body of endnote continuation notice

Unlike other body types, the main body has Sections as the top level content elements. It also contains comments, footnotes and endnotes collections. There are three types of **content elements** that can be stored in a body:

Content Element Type	Description	Content Elements
Block elements	Top level elements	<ul style="list-style-type: none"> • Sections • tables • paragraphs
Inline elements	Elements that must be placed inside another elements	<ul style="list-style-type: none"> • Runs • Texts • Pictures • Simple fields • Hyperlinks • Footnotes • Endnotes
Reference elements	Elements that do not have its own content in the body (except for complex fields, see Complex Fields) but are represented by start/end markers.	<ul style="list-style-type: none"> • Bookmarks • Comments • Complex fields

The following sections explain how to access and work with various content elements of a body.

Range

A range is a sequence of **content elements** in a body. The body itself is a kind of range that holds all the content elements. In DsWord, the [Range](#) class is the main feature providing access to the various content elements in a document.

All content elements have the [GetRange\(\)](#) method, using which it is possible to access and modify collections of elements of specific types inside the content element's range, since the Range object has properties returning collections of specific types of objects included in the range. These collections allow to add/insert elements using the [Add\(\)](#) and [Insert\(\)](#) methods.

Please note that adding or inserting always occurs on one or both (e.g. when replacing a range) of a range's boundary. It is not possible to insert something in the middle of a range without creating a range with a boundary on that position first.

A range provides the following two overloads to get new ranges based on it:

Method	Description
GetRange (ContentObject first, ContentObject last)	Gets a range that extends from the 'first' content object to the 'last'
GetRange(Marker start, Marker end)	Gets a range providing a fine-grained control over the range's bounds, e.g. GetRange(first.End, last.Start). For more information, see DsWord API Reference.

To clear all content in a range use the [Range.Clear\(\)](#) method. Range, being a collection of ContentObject, allows to enumerate the content elements included in it.

ContentObject

Block and inline elements are derived from the [ContentObject](#) class which provides access to the start and end position of an element in a document. Also, it allows to get the parent content element and enumerate the element's children.

In addition, all content objects have the Next and Previous properties which allow to enumerate objects of the same content type through the whole body.

The [Delete\(\)](#) method of the ContentObject class removes the element itself and all its inner content from the body.

ContentRange

Reference elements, bookmarks, comments, and complex fields, are slightly different from simple ContentObject. This kind of elements do not have a parent content since the element can start and end anywhere. For example, it can start in one section and end in another. Instead, reference elements provide a pair of ContentObjects named ContentMark that define the start and end of the element. The ContentMark has Owner property that points to the ContentRange element. Removing a ContentMark from the body also removes its owner element. The Delete() method on a ContentRange usually removes its ContentMarks only. Complex fields are an exception to this as its actual internal content is also deleted.

Complex Fields

Despite the fact that the complex field inherits from ContentRange, it actually is a combination of ContentRange and ContentObject. Bounds of a complex field are defined by special field characters (see the FieldChar class and the associated enum that defines the type of the field character as Begin, Separator or End values). The complex field can contain two ranges, code range and result range, separated by a Separator field character.

The code range usually contains one or several codes (see [FieldCode](#) class) that in turn contain instructions on how to calculate the field's result. The result range contains cached result of the instructions. In the current version, DsWord does not yet calculate instructions, so it does not update the result.

As mentioned above, unlike other ContentRange elements, the Delete() method on a complex field removes not only the field characters from the body but the field codes and the result too.

Section

Sections can only be present in the main body, and any document must have at least one section.

Sections allow to change page formatting for the document parts; PageSetup property and headers or footers collections of a section provide the means to do that. Each section can have its own headers or footers and page

formatting.

Headers and footers display on each page of the section and they have their own bodies to store their content. There are several types of headers or footers in a section (see `HeaderFooterType` enum) and each header or footer can be linked to the same type from a previous section, so you do not have to create identical headers or footers for each section.

Run

A run is a contiguous fragment of a body content with uniform formatting. So, a run is the primary means to change character formatting. It is also a container for all other inline elements (excluding simple fields and hyperlinks).

Nesting elements

The top elements in the main body are the sections. For other body types, the top elements can be paragraphs, tables and content marks (see `ContentRange`).

Usually elements with the same type cannot be nested (for example, a Run cannot be nested within another Run). Only SimpleField and Hyperlink can be nested. Also, a cell in a table can contain another table within its own cells.

Styles

Styles is the main means allowing to apply formatting to a document's content. DsWord provides 375 built-in styles. There are different style types (see `StyleType` enumeration). Each type of style can be applied only to the corresponding content type. You can get any built-in type using `BuiltInStyleId` enumeration.

The `StyleCollection` class has default styles which can be fetched or set using its `GetDefaultStyle(StyleType)` or `SetDefaultStyle(StyleType, Style)` methods. These styles are applied to content that does not have an explicitly specified style. `StyleCollection` provides the `DefaultFont` and `DefaultParagraphFormat` properties which are used by default for the default styles.

Some styles are linked. A linked style is a grouping of a paragraph style and character style which is used in a user interface to allow the same set of formatting properties. For example, if you want to apply Heading 1 paragraph style to a run, you can apply it using `Document.Styles[BuiltInStyleId.Heading1].LinkStyle`.

Formatting inheritance

DsWord allows to get the actual formatting values of elements. It takes into account the formatting inheritance from default document formatting, base style formatting, applied style formatting, parent content formatting and direct formatting of the element.

ListTemplates

DsWord provides 21 built-in list templates to create lists in the document. The formatting of these templates is the same as in Microsoft Word built-in list templates. There is no "list" class in DsWord. To create a list you need to set `ListFormat.Template` and `ListFormat.LevelNumber` (for multilevel lists) properties on each paragraph that should be in the list.

Settings

The `Settings` class allows to set properties that apply to the whole document, add custom document properties, control document variables, detect and remove document macros, and change view options.

Features

This section comprises the features available in DsWord.

Comments

Add, modify, and delete comments and comment replies in DsWord.

Content Controls

Work with content controls in DsWord.

Glossary Document

Maintain the collection of building blocks in Glossary document in DsWord.

Document

Manage various document related operations in DsWord.

Export

Export Word documents to PDF and image file formats in DsWord.

Fields

Add, modify, and delete fields in DsWord.

Footnotes and Endnotes

Add, modify, and delete footnotes and endnotes from a Word document in DsWord.

Font

Define custom fonts, embed them in a Word document and apply them to the runs in DsWord.

Header and Footer

Add, modify, and delete header and footer from a Word document in DsWord.

Images

Add image from file/stream, extract, edit, and delete images from a Word document in DsWord.

Shapes

Add various shapes like textbox, canvas, group, ink etc and apply shape format and shape styles in DsWord.

Office Math

Add and display the Office Math content in DsWord.

Links

Add, modify, and delete hyperlinks and bookmarks from a Word document in DsWord.

Lists

Work with lists and list templates in DsWord.

Paragraph

Work with paragraph and its properties in DsWord.

Range Objects

Work with range objects in DsWord.

Sections

Work with sections and section breaks in DsWord.

Table

Work with tables, cells and rows in DsWord.

Text

Work with text objects in DsWord.

Themes

Change the theme color and font of a Word document in DsWord.

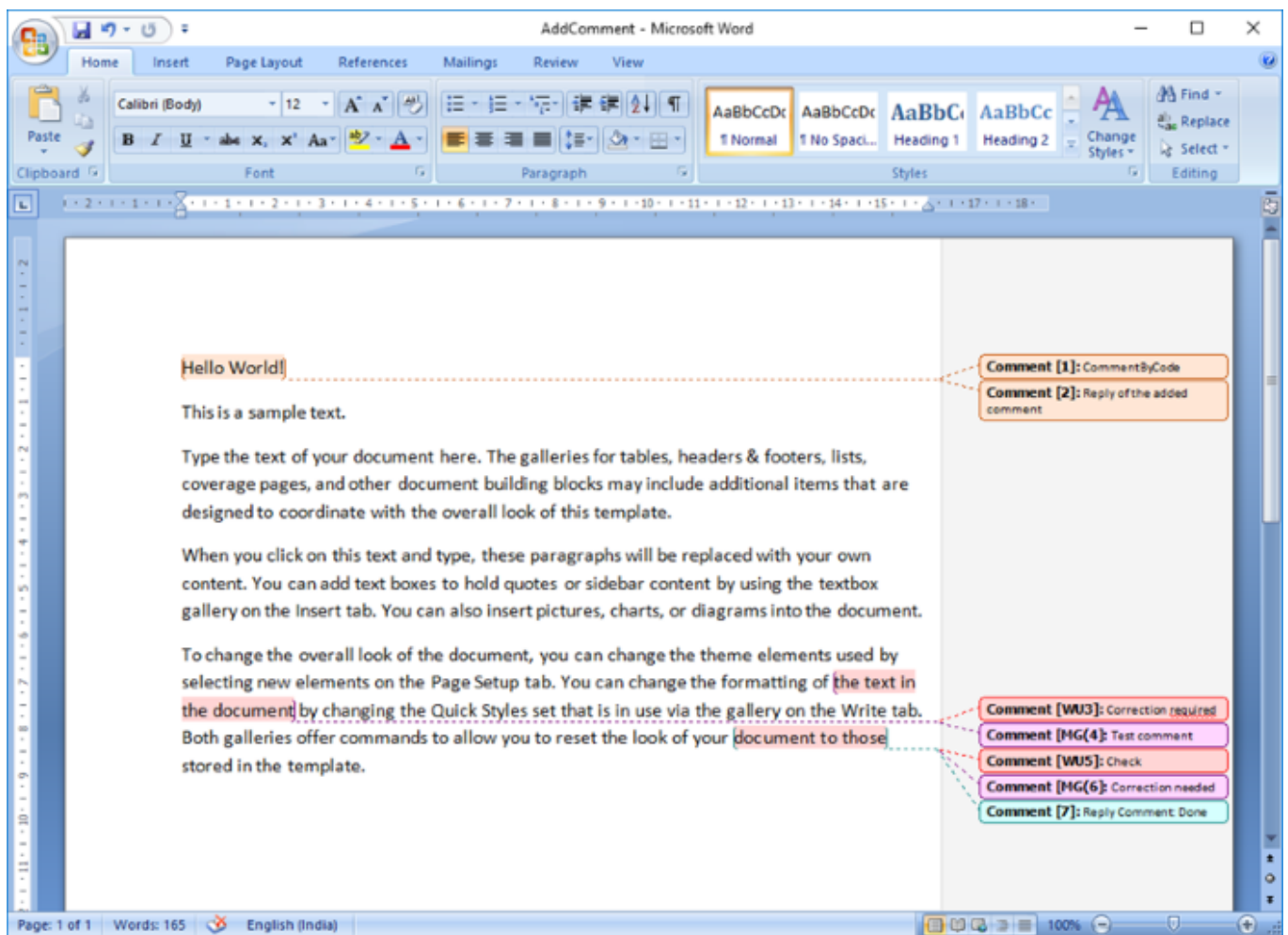
Helper Methods for Adding Content

Add various content elements using helper methods to a Word document in DsWord.

Comments

A comment is a note or a type of annotation which is usually added to a document when a user wants to add remarks, notes, reminder, or a feedback to it. For instance, when multiple users work on the same document, it becomes easy for them to leave a note, remarks or feedback for each other's reference.

In DsWord, comments are represented by the [Comment](#) class. It allows you to modify the content and other properties of the comments, such as, author, date of the comment, etc. DsWord allows you to add or insert comments in a Word document using Add or Insert method of the [CommentCollection](#) class respectively. In addition, it lets you add and delete the comment replies as well. You can use [Reply](#) method of the **Comment** class to add a new comment as a reply into the comments collection and the [Delete](#) method to delete it.



Add Comments

To add comments in a document:

1. Access the content object on which you want to add a comment. For example, access the first paragraph.
2. Access the comment collection using [Comments](#) property of the [RangeBase](#) class.
3. Add a comment to the paragraph using [Add](#) method of the [CommentCollection](#) class.

```
C#
doc.Load("CommentInWord.docx");

//Access the first paragraph
```



```
Paragraph par = doc.Body.Sections.First.GetRange().Paragraphs[0];

//Add comment
Comment c=par.GetRange().Comments.Add("CommentByCode", "GC");

//Save the document
doc.Save("AddComment.docx");
```

[Back to Top](#)

Modify Comments

To modify a comment through code:

1. Access a comment from the comment collection using the **Comments** property. For example, access the first comment.
2. Get the property of the comment you want to modify. For example, access Author and Initials properties of the first comment.
3. Modify the value of the properties. For example, set value of the Author property to DsWordUser and Initials property to G.C.W.

```
C#
doc.Load("AddComment.docx");

//Access the first paragraph
Paragraph par = doc.Body.Sections.First.GetRange().Paragraphs[0];

//Modify the first comment's author and initials
par.GetRange().Comments[0].Author = "DsWordUser";
par.GetRange().Comments[0].Initials = "D.S.W";

//Mark the comment reply as closed
par.GetRange().Comments[0].Done = true;

doc.Save("ModifiedComment.docx");
```

[Back to Top](#)

Delete Comments

To delete a comment:

1. Access a comment from the comment collection using the **Comments** property. For example, access the fourth comment in the document.
2. Delete the comment using **Delete** method of the Comment class.

```
C#
doc.Load("CommentInWord.docx");

//Delete the second comment
doc.Body.Comments[3].Delete();
doc.Save("DeleteComment.docx");
```

[Back to Top](#)

Add Comment Reply

To add a reply on a comment:

1. Access a comment from the comment collection using the **Comments** property. For example, access the second comment.
2. Add a new comment in the comment collection as a reply for the accessed comment using **Reply** method of the Comment class.

```
C#  
  
doc.Load("CommentInWord.docx");  
  
//Add comment reply for the last comment in the document  
doc.Body.Comments.Last.Reply("Reply Comment", "John Doe");  
  
//Add a comment reply  
Comment cReply = c.Reply("Reply of the added comment", "JD");  
  
//Mark the comment reply as closed  
cReply.Done = true;  
  
//Save the document  
doc.Save("AddComment.docx");
```

[Back to Top](#)

Delete Comment Reply

To delete a reply on the comment:

1. Access a reply comment using Replies property of the **Comment** class. For example, access first reply of the first comment.
2. Delete the comment reply using **Delete** method of the Comment class.

```
C#  
  
//Delete first comment's first reply  
doc.Body.Comments.First.Replies[0].Delete();  
  
doc.Save("DelComment.docx");
```

[Back to Top](#)

For more information on how implement comments using DsWord, see [DsWord sample browser](#).

Content Controls

DsWord provides different types of content controls that help in organizing the content of a document in a structured manner. They are often used for creating templates and forms. They provide the flexibility to position the content and deny or allow editing of controls.

For example, consider a scenario of a company event. Thousands of invite forms need to be generated which will have the company address by default. Since the same address needs to be replicated over all the invite forms, this can be done by using Repeating Section content control. Also, in case the company address needs to be changed, the value of address field will be updated only in the Repeating Section control and it will automatically reflect in all the invite forms.

DsWord library supports content controls which are represented by the [ContentControl](#) class. In fact, there are various types of content controls which are represented by the enum [ContentControlType](#) and are explained below:

Type	Description
Plain text	A plain text control is restricted to write plain text only.
Rich text	A rich text control contains formatted text as well as other items like tables, pictures etc.
Picture	A picture control displays a single image in the content control.
ComboBox	A combo box can contain any arbitrary text and also displays a list of values which can be selected from a drop down.
Drop down list	A drop-down list displays a list of items out of which only one can be selected.
Date	A date control provides a calendar for selecting a date.
CheckBox	A check box provides the option to represent the binary state: checked or unchecked.
Building block gallery	A building block gallery allows to select from a list of building blocks to insert into a document.
Group	A group control represents a protected region which users cannot edit or delete.
Repeating Section	A repeating section control acts as a container for repeated items.
RepeatingSectionItem	A repeating section item specifies that the content control is a repeated item.
BuildingBlock	A building block is a pre-designed and pre-formatted block of text and formatting.
Equation	An equation specifies that the content control shall be of type equation.
Bibliography	A bibliography specifies that the content control shall be of type bibliography.
Citation	A citation specifies that the content control shall be of type citation.
ExternalContentEntityPicker	An external content entity picker control allows the user to select an instance of an external content type.

DsWord allows you to add, modify, and delete content controls from a Word document. A content control can be created using the [Add](#) or [Insert](#) method of the [ContentControlCollection](#) class. It can be modified using the [ContentControl](#) class properties, and deleted using [Delete](#) method of the [ContentControl](#) class.

Create Content Control

To create a content control:

1. Create a new Word document by instantiating the GcWordDocument class.
2. Add a paragraph using **Add** method of the ParagraphCollection class.
3. Create a content control of type plain text using the **Add** method of ContentControlCollection class.
4. Configure the content control by setting its properties.
5. Add text to the content control using **Add** method of the ParagraphCollection class.

```
C#
GcWordDocument doc = new GcWordDocument();

doc.Body.Sections.First.GetRange().Paragraphs.Add("This sample demonstrates
creation/insertion of content controls in a Word document");
Style paraStyle = doc.Styles.Add("paraStyle", StyleType.Paragraph);
paraStyle.Font.Bold = true;
paraStyle.Font.Size = 12;

doc.Body.Sections.First.GetRange().Paragraphs.Add("Employee Name: ", paraStyle);

//Create a content control of type plain text and add static text to it
ContentControl cc_plainText =
doc.Body.ContentControls.Add(ContentControlType.Text, false);
cc_plainText.Appearance = ContentControlAppearance.BoundingBox;
cc_plainText.LockContents = false;
cc_plainText.LockControl = true;
cc_plainText.Content.GetRange().Paragraphs.Add("Robert King");

doc.Save("ContentControls.docx");
```

Back to Top

To create a content control using custom placeholder text:

1. Create a new Word document by instantiating the GcWordDocument class.
2. Add a content control of type plain text using the **Add** method of ContentControlCollection class.
3. Create a building block that defines the custom placeholder text using the **Add** method of BuildingBlockCollection class.
4. Add the custom placeholder text to the building block by adding a paragraph to it.
5. Assign the value of the custom placeholder text to the content control's placeholder text by setting the **PlaceholderText** property of the content control.

```
C#
GcWordDocument doc = new GcWordDocument();

doc.Body.Sections.First.GetRange().Paragraphs.Add("Job Title: ", paraStyle);

//Create a content control of type rich text with custom placeholder
ContentControl cc_custom =
doc.Body.ContentControls.Add(ContentControlType.Text);
BuildingBlock placeholder =
doc.GlossaryDocument.BuildingBlocks.Add("placeholder", "General",
BuildingBlockGallery.Placeholder, BuildingBlockInsertOptions.Content,
BuildingBlockType.ContentControlPlaceholder);
```

```
Paragraph p = placeholder.Body.Paragraphs.Add("Enter your job title:");
p.GetRange().Runs.First.Style =
p.ParentBody.Document.Styles[BuiltInStyleId.PlaceholderText];
cc_custom.PlaceholderText = placeholder; // attach the custom placeholder to the
content control

doc.Save("ContentControls.docx");
```

[Back to Top](#)

Modify Content Control

To modify a content control:

1. Load the document using **Load** method of GcWordDocument class.
2. Access the content control which needs to be modified using **ContentControls** property of **RangeBase** class.
3. Clear the existing text in the content control using the **Clear** method of RangeBase class.
4. Add new text to the content control using the **Add** method of the ParagraphCollection class.

```
C#
GcWordDocument doc = new GcWordDocument();
doc.Load("ContentControls.docx");

//Retrieves the content control and modifies its content
ContentControl contentControl = doc.Body.ContentControls[0];
contentControl.Content.GetRange().Clear();
contentControl.Content.GetRange().Paragraphs.Add("Andrew Fuller");

doc.Save("ContentControl_Modified.docx");
```

[Back to Top](#)

Delete Content Control

To delete a content control:

1. Load the document using **Load** method of GcWordDocument class.
2. Access the content control which needs to be deleted using the **ContentControls** property of **RangeBase** class.
3. Delete the content control using **Delete** method of the ContentControl class.

```
C#
GcWordDocument doc = new GcWordDocument();
doc.Load("ContentControls.docx");

//Retrieves the first content control and deletes it
ContentControl contentControl = doc.Body.ContentControls.First();
contentControl.Delete();

doc.Save("ContentControl_Deleted.docx");
```

[Back to Top](#)

Bind Content Controls to Custom XML parts

DsWord supports binding content controls to XML data using custom XML parts. Custom xml parts are used to embed XML data in documents for some Microsoft Office applications. Any change that is made to the text in a content control is saved to the custom XML part and vice versa. DsWord allows you to bind content controls to elements in a custom XML part using the [XmlMapping](#) class.

To bind the content control to custom XML parts:

1. Create and load an XML document using **Load** method of **XmlDocument** class.
2. Use **Add** method of **CustomXmlPartCollection** to embed the XML data to the word document.
3. Create a content control of type plain text which will be bound to the custom XML part.
4. Bind the content control to an element in the custom XML part using the **SetMapping** method of **XmlMapping** class.

C#

```
GcWordDocument doc = new GcWordDocument();

doc.Body.Sections.First.GetRange().Paragraphs.Add("This sample demonstrates the
binding of content controls to custom XML parts ");
Style paraStyle = doc.Styles.Add("paraStyle", StyleType.Paragraph);
paraStyle.Font.Bold = true;
paraStyle.Font.Size = 12;

//Create an XML document
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load("Employees.xml");

//Add the XML document to the custom XML part

string xmlPartId = Guid.NewGuid().ToString();
doc.CustomXmlParts.Add(xmlDoc, xmlPartId);

doc.Body.Sections.First.GetRange().Paragraphs.Add("Employee Name: ", paraStyle);

//Create a content control of type plain text
ContentControl plainTextContentControl =
doc.Body.ContentControls.Insert(ContentControlType.Text, InsertLocation.End);
//bind the content control to element in the custom XML part using XPath
expression
plainTextContentControl.XmlMapping.SetMapping("/employees/employee[1]/name[1]",
null, xmlPartId);

doc.Save("XMLMapping.docx");
```

[Back to Top](#)

Bind Content Controls to Built-in Properties

DsWord allows you to bind content controls to built-in document properties (like Application name, Title, Last saved time, Version, Author, Category etc). The [SetMapping](#) method of **XmlMapping** class takes the property name as a

parameter to establish the binding.

To bind the content control to built-in properties:

1. Create a new Word document by instantiating the GcWordDocument class.
2. Set the value of a built-in property 'Author' using the **BuiltinProperties** property of the **BuiltInPropertyCollection** class.
3. Create a content control of type plain text which will be bound to the built-in property.
4. Map the content control to built-in property using the **SetMapping** method of XmlMapping class.

C#

```
GcWordDocument doc = new GcWordDocument();

doc.Body.Sections.First.GetRange().Paragraphs.Add("Document Author: ",
paraStyle);

doc.Settings.BuiltinProperties.Author = "James Smith"; // set a built-in
property value

// Add a new content control of type plain text for the built-in property
ContentControl cc_builtInProp =
doc.Body.ContentControls.Insert(ContentControlType.Text, InsertLocation.End);
/*map the built-in property value with the content control. Now when a user
changes the content control value the built-in property will be changed
automatically*/
string author = cc_builtInProp.XmlMapping.SetMapping(() =>
doc.Settings.BuiltinProperties.Author);

doc.Save("XMLMapping.docx");
```

Back to Top

For more information on how implement comments using DsWord, see [DsWord sample browser](#).

Glossary Document

A glossary document is a supplementary storage which stores the content, such as AutoText/Buildingblock entries that you do not want to appear in the document, but want to keep it as the part of document for future insertion, if needed. DsWord provides the capability of maintaining a GlossaryDocument for a Word document to store the BuildingBlocks content. It is represented using the [GlossaryDocument](#) class.

GlossaryDocument class maintains the collection of building blocks using the [BuildingBlockCollection](#) class. The BuildingBlockCollection class provides [Add](#) and [Remove](#) method to add and remove the blocks from the glossary document respectively.

Building blocks is an essential feature of word which allows you to insert blocks of information in the document. These blocks of information can be reusable chunks of content or pre-designed and pre-formatted blocks of text and formatting. DsWord allows you to create, modify and delete building blocks using the BuildingBlock class.

Create Building Block in Glossary Document

To create and add building blocks in a glossary document:

1. Create an instance of the GcWordDocument class to create a document. The **GlossaryDocument** property of this class returns the Glossary Document specific to the Word document.
2. Add the building block to the Glossary document using the **Add** method of BuildingBlockCollection.

```
C#
GcWordDocument doc = new GcWordDocument();

//Adds a building block that contains text
BuildingBlock bbl = doc.GlossaryDocument.BuildingBlocks.Add("Mission_Statement",
"CompanyInfo", BuildingBlockGallery.AutoText, BuildingBlockInsertOptions.Content,
BuildingBlockType.None);
Style paraStyle = bbl.Body.Document.Styles.Add("paraStyle", StyleType.Paragraph);
paraStyle.Font.Bold = true;
bbl.Body.Paragraphs.Add("To serve our customers by helping them to achieve their
goals", paraStyle);

//Save the document (Note:Building blocks can be saved in dotx and docx, but in the
docx format, MS Word UI won't show them)
doc.Save("CustomBuildingBlocks.dotx", DocumentType.Template);
```

[Back to Top](#)

Remove Building Block from Glossary Document

To remove building block from glossary document:

1. Load the document using **Load** method of GcWordDocument class.
2. Invoke **Remove** method of the **BuildingBlockCollection** class to remove the building block from glossary document.

```
C#
GcWordDocument doc = new GcWordDocument();
doc.Load("CustomBuildingBlocks.dotx");
doc.GlossaryDocument.BuildingBlocks.Remove(doc.GlossaryDocument.BuildingBlocks.First);
```

[Back to Top](#)

Document

In DsWord, a document is represented by the [GcWordDocument](#) class which is the main class that contains document properties. The **GcWordDocument** class provides access to the main functionality, such as creating, loading, and saving of documents.

In this section, you learn how to work with the following:

- [Document Properties](#)
- [Document Styles](#)
- [Document Protection](#)
- [Document Formatting Source](#)
- [Copy or Move Document Content](#)
- [Split or Merge Documents](#)
- [Page Settings](#)

Document Properties

Apart from content, a Word file holds some additional information in the form of document properties. These properties define various attributes of document as a whole.

DsWord provides following document properties through the GcWordDocument class:

Body

DsWord allows you to access the body of the document using the [Body](#) property, which contains the text that excludes headers, footers, footnotes, text boxes, etc.

List Templates

DsWord provides the [ListTemplates](#) property to get a collection of list templates in the document.

Document Styles

DsWord allows you to access the collection of styles defined in the document using the [Styles](#) property.

Theme

DsWord provides the [Theme](#) property to get a theme that holds all the different formatting options available to a document through the theme.

Settings

DsWord provides the [Settings](#) property which gives you options to control various settings of a Word document, such as:

- **Compatibility options:** These options influence how the document content appears and are handled using the [CompatibilityOptions](#) class which can be accessed using [CompatibilityOptions](#) property of the [Settings](#) class.
- **View Options:** These options in Word documents lets you control the view and layout of a document. They are handled through the [ViewOptions](#) class which can be accessed using [ViewOptions](#) property of the [Settings](#) class.
- **Hyphenation options:** These options are useful to render text in different marginal or justified settings by breaking words in between the lines to bring more consistency in text. The options are handled using the [HyphenationOptions](#) class which can be accessed using [HyphenationOptions](#) property of the [Settings](#) class.

Document Solutions for Word library, referred to as DsWord, is a part of Document Solutions.

[Testtextforhyphenation](#). DsWord aims to be a complete solution to program and work with Word documents, [Testtextforhyphenation](#) without using any external word processor like MS Word. [Testtextforhyphenation](#). DsWord is a [high performance](#) library which is supported on .NET Standard 2.0 and can be used to create, load, modify, and save Word documents programmatically.

Get Document Properties

To get the document properties, for example, compatibility mode and hyphenation options:

1. Get access to the document compatibility options using [Settings.CompatibilityOptions](#) property.

2. Get the document compatibility mode using [CompatibilityMode](#) of the [CompatibilityOptions](#) class. This will give you the version of Word document.
3. Get the maximum number of consecutive lines that can end with hyphens using [ConsecutiveHyphensLimit](#) property of the **HyphenationOptions** class.
4. Display the compatibility mode version and number of consecutive lines that can end with hyphens on the console.

```
C#  
  
doc.Load("SampleDoc.docx");  
  
//get document compatibility mode  
WordVersion version = doc.Settings.CompatibilityOptions.CompatibilityMode;  
  
//Get the maximum number of consecutive lines that can end with hyphens  
ushort limit = doc.Settings.HyphenationOptions.ConsecutiveHyphensLimit;  
  
//Display the compatibility mode version on the console  
Console.WriteLine("\n WordVersion: " + version);  
  
//Display the maximum number of consecutive lines that can end with hyphens  
Console.WriteLine("\n consecutive lines ending with hyphens: " + limit);
```

[Back to Top](#)

Set Document Properties

To set the document properties, for example, compatibility mode and hyphenation options:

1. Get access to the document compatibility options using **Settings.CompatibilityOptions** property.
2. Set the document compatibility mode using **CompatibilityMode** of the **CompatibilityOptions** class, which takes the value from the [WordVersion](#) enumeration.
3. Set the automatic hyphenation for the document using [AutoHyphenation](#) property of the **HyphenationOptions** class by providing a Boolean value.

```
C#  
  
doc.Load("SampleDoc.docx");  
  
//set document compatibility mode  
doc.Settings.CompatibilityOptions.CompatibilityMode = WordVersion.Word2007;  
  
//Enable automatic hyphenation for the document  
doc.Settings.HyphenationOptions.AutoHyphenation = true;  
  
//Save the modified Word file  
doc.Save("SetDocProperties.docx");
```

[Back to Top](#)


Set View Options

DsWord provides various options to control how a document is displayed in an application through the [ViewOptions](#) class, which can be accessed using the [ViewType](#) property. It helps you display the page in different view modes such as master document view, draft view, outline view, print view and webpage view. DsWord also lets you set the zoom

levels using the [ZoomType](#) property of the **ViewOptions** class. The different zoom modes available are **BestFit**, **FullPage**, **TextFit** and **None**. In addition, the ViewOptions class lets you specify the zoom percentage using the [Zoom](#) property.

To set various viewing options, for example, view type, the zoom percentage and zoom type:

1. Access the viewing options using **Settings.ViewOptions** property.
2. Set the view mode using **ViewType** property of the ViewOptions class which accepts value from the [ViewType](#) enumeration.
3. Set the zoom value using **ZoomType** property of the ViewOptions class which accepts value from the [ZoomType](#) enumeration.
4. Set the zoom percentage using **Zoom** property of the ViewOptions class.

 **Note:** Microsoft Word ignores these properties set through the ViewOptions class while displaying a document as it reads the properties directly from the Windows registry.

C#

```
doc.Load("SampleDoc.docx");

//set view type
doc.Settings.ViewOptions.ViewType = ViewType.Print;

//set zoom type
doc.Settings.ViewOptions.ZoomType = ZoomType.Fullpage;

//set zoom percentage
doc.Settings.ViewOptions.Zoom = 150;

doc.Save("ViewOptionsAdded.docx");
```

Back to Top

For more information on how implement document properties using DsWord, see [DsWord sample browser](#).

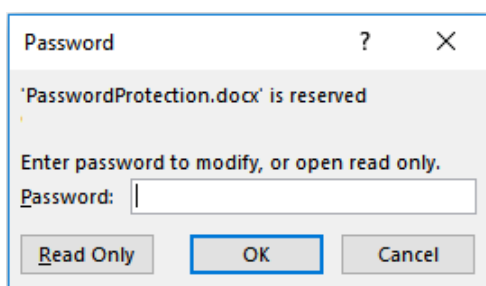
Document Protection

Document protection is an essential feature in Word documents to restrict unauthorized access. The DsWord library provides the [DocumentProtection](#) class to protect the Word documents. With the properties in the **DocumentProtection** class, you can now add editing restrictions, write protections, document theme, style protections etc. to your document.

Password

With Password protection in DsWord, you can perform one of the following:


- Open the document in edit mode in case you provide the password.
- Open the document in ReadOnly mode without providing the password. You can make changes to the document but cannot save them in the same file.



To set a password:

1. Load a document using the **Load** method of GcWordDocument class.
2. Set a password using the **SetPassword** method of **Password** class, which can be accessed using the **WritePassword** property of **DocumentProtection** class.

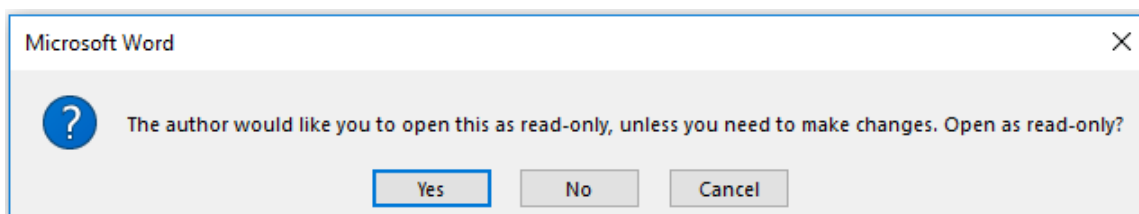
```
C#  
  
//Load the Word document  
GcWordDocument doc = new GcWordDocument();  
doc.Load("SampleDoc.docx");  
  
//Set the write protection password for the document  
doc.Settings.DocumentProtection.WritePassword.SetPassword("abc");  
  
//Save the document  
doc.Save("PasswordProtection.docx");
```

 **Note:** User passwords or protection settled in this way can be removed using raw docx modification in any moment. It is not "strong" cryptographic protection.

Back to Top

ReadOnly

This is a special ReadOnly mode that recommends users to open the document as read-only. When this mode is set, the below dialog appears in Word:



- **Yes** - Opens the file in **ReadOnly** mode. You can make changes to the document but cannot save them in the same file.
- **No** - Opens the document in edit mode.
- **Cancel** - Closes the document.

To set **ReadOnlyRecommended** mode:

1. Load a document using the **Load** method of **GcWordDocument** class.
2. Set the **ReadOnlyRecommended** property of the **DocumentProtection** class to true.

```
C#  
  
//Load the Word document  
GcWordDocument doc = new GcWordDocument();  
doc.Load("SampleDoc.docx");  
  
//Recommends users to open a document as read-only  
doc.Settings.DocumentProtection.ReadOnlyRecommended = true;  
  
//Save the document  
doc.Save("ReadOnlyRecommendedpassword.docx");
```

Back to Top

Mark as Final

Mark as Final mode makes the document read-only. However, you can make changes in the document by clicking the 'Edit Anyway' option in dialog box.

To set a document to Mark as Final:

1. Load a document using the **Load** method of **GcWordDocument** class.
2. Set the **MarkAsFinal** property of the **DocumentProtection** class to true.

```
C#  
  
//Load the Word document  
GcWordDocument doc = new GcWordDocument();  
doc.Load("SampleDoc.docx");  
  
//Marks a document as Final  
doc.Settings.DocumentProtection.MarkAsFinal = true;  
  
//Save the document  
doc.Save("MarkAsFinal.docx");
```

Back to Top

Editing Restrictions

DsWord supports editing restrictions to limit the user's ability to edit text in the document. You can apply certain edit modes to the document by using the **EditProtectionMode** enumeration.

The **IsActive** property of **EditProtection** class should be set to true in order to enforce the editing restrictions. You can also set a password using the **Password** property of **EditProtection** class which enables only the users with password to remove the editing restrictions from the document.

Edit modes	Description
AllowOnlyReading	Allows read-only access to the document except regions(ranges) with defined editor rights.
AllowOnlyComments	Allows only comments to be added to the document.
AllowOnlyFormFields	Allows content to be added to the document only through form fields
NoProtection	Does not apply any protection to the document.

1. Load a document using the **Load** method of **GcWordDocument** class.
2. Set the enum value of **EditProtectionMode** to **AllowOnlyComments** using the **EditMode** property of **EditProtection** class.
3. Set the **IsActive** property of **EditProtection** class to true.
4. Set the password using **SetPassword** method of **Password** class which can be accessed using **EditProtection** property of **DocumentProtection** class

```
C#
//Load the Word document
GcWordDocument doc = new GcWordDocument();
doc.Load("SampleDoc.docx");

//Specify the protection mode for a document
doc.Settings.DocumentProtection.EditProtection.EditMode =
EditProtectionMode.AllowOnlyComments;

//Set a password so that users who know the password can remove the protection and work
on the document
doc.Settings.DocumentProtection.EditProtection.Password.SetPassword("abc123");

//enforce document protection
doc.Settings.DocumentProtection.EditProtection.IsActive = true;

//Save the document
doc.Save("EditingRestrictions.docx");
```

[Back to Top](#)

Editable Ranges

Editable ranges are specific regions in a read-only document, which can be edited or modified by users or groups with editing rights. To enable editable ranges, the **IsActive** property of **EditProtection** class should be set to true and the edit mode should be set to **AllowOnlyReading**. The whole document behaves as **ReadOnly**, except the editable ranges for defined users or groups.

To create editable ranges:

1. Create a new document and add a paragraph using the **Paragraphs** property of **RangeBase** class
2. Add editable range for the first paragraph using **Add** method of **EditableRangeCollection** class. To grant editing rights to everyone, **Everyone** value of **EditorGroup** enumeration has been passed to the **GroupEditor** method.
3. Add second paragraph and add editable range to it using **Add** method of **EditableRangeCollection** class. To grant editing rights to a specific user, the user ID has been passed to **UserEditor** method.
4. Add third paragraph to the document.
5. Set the enum value of **EditProtectionMode** to **AllowOnlyReading** using the **EditMode** property of **EditProtection** class.
6. Set the **IsActive** property of **EditProtection** class to true.
7. Set the password using **SetPassword** method of **Password** class which can be accessed using **EditProtection** property of **DocumentProtection** class

```
C#
GcWordDocument doc = new GcWordDocument();

//add first paragraph
var firstPara = doc.Body.Paragraphs.Add("First paragraph");

//add new EditableRange for the first paragraph and allow everyone to modify it
firstPara.GetRange().EditableRanges.Add(new GroupEditor(EditorGroup.Everyone));

//add second paragraph
var secondPara = doc.Body.Paragraphs.Add("Second paragraph");

//add new EditableRange for the second paragraph and allow a specific user to to modify it
```

```
secondPara.GetRange().EditableRanges.Add(new UserEditor("google\\abc.xyz"));

//add third paragraph
doc.Body.Paragraphs.Add("Third paragraph");

//set document region protection mode
doc.Settings.DocumentProtection.EditProtection.EditMode =
EditProtectionMode.AllowOnlyReading;
//enforce document protection
doc.Settings.DocumentProtection.EditProtection.IsActive = true;
//Set EditProtection Password
doc.Settings.DocumentProtection.EditProtection.Password.SetPassword("123");

//now, first paragraph can be edited by everyone.
//Second paragraph can be edited by a specific user
//rest of the document is ReadOnly
doc.Save("EditableRanges.docx");
```

Back to Top

To delete Editable ranges:

1. Load the document and access the editable paragraph using the **Paragraphs** property of **RangeBase** class
2. Delete the editable range using the **Delete** method of **EditableRange** class.

```
C#
GcWordDocument doc = new GcWordDocument();

//Load the document
doc.Load("EditableRanges.docx");

//Get the editable paragraph
var para = doc.Body.Paragraphs[1];

//Delete the EditableRange using the Delete method
para.GetRange().EditableRanges[0].Delete();

//Save the document
doc.Save("EditableRange_Deleted.docx");
```

Back to Top

Formatting Restrictions

Formatting restrictions provide various properties to restrict the formatting changes that can be applied to a document. These properties are described below:

Properties	Description
EditProtection.LimitFormattingToUnlockedStyles	Styles with "locked" attribute are not available for use.
EditProtection.AllowAutoformatToOverrideFormatRestrictions	Auto-formatting can override formatting restrictions.
DocumentProtection.BlockThemeOrSchemeSwitching	Blocks the ability to change the theme.
DocumentProtection.BlockQuickStylesetSwitching	Blocks the ability to change the Style Set.

To set formatting restrictions using the properties of **EditProtection** class:

The **IsActive** property of **EditProtection** class should be set to true to enforce the formatting restrictions. You can also set a password using the **Password** property of **EditProtection** class which enables only the users with password to remove these formatting restrictions from the document.

1. Load a document using the **Load** method of **GcWordDocument** class.
2. Lock the Heading1 and hyperlink style by setting the **Locked** property of **Style** class to true.
3. Set the **LimitFormattingToUnlockedStyles** and **AllowAutoformatToOverrideFormatRestrictions** properties of **EditProtection** class to true.
4. Set the **IsActive** property of **EditProtection** class to true.
5. Set the password using **SetPassword** method of **Password** class which can be accessed using **EditProtection** property of **DocumentProtection** class.

```
C#
//Load the Word document
GcWordDocument doc = new GcWordDocument();
doc.Load("SampleDoc.docx");

//Lock the Heading1 style (Note that this style will not be available for use
//in the Word document when the LimitFormattingToUnlockedStyles property of
//the EditProtection class is set to true)
Style heading1_Style = doc.Styles[BuiltInStyleId.Heading1];
heading1_Style.Locked = true;

//Lock the Hyperlink style (Note that this style will not be available for use
//in the Word document when the LimitFormattingToUnlockedStyles property of
//the EditProtection class is set to true. However this style will be available
//when AllowAutoformatToOverrideFormatRestrictions property of the EditProtection
//class is set to true automatically formatting hyperlinks)
Style hyperlink_style = doc.Styles[BuiltInStyleId.Hyperlink];
hyperlink_style.Locked = true;

//Limit formatting to the unlocked styles
doc.Settings.DocumentProtection.EditProtection.LimitFormattingToUnlockedStyles = true;

//Allow using locked styles when automatically formatting text such as hyperlinks or
automatic bullets
doc.Settings.DocumentProtection.EditProtection.AllowAutoformatToOverrideFormatRestrictions
= true;

//enforce formatting restrictions
doc.Settings.DocumentProtection.EditProtection.IsActive = true;

//Set a password so that users who know the password can remove the protection and work
on the document
doc.Settings.DocumentProtection.EditProtection.Password.SetPassword("123");

//Save the document
doc.Save("FormattingRestrictions.docx");
```

Back to Top

To set formatting restrictions using the properties of **DocumentProtection** class:



Note: These restrictions can be removed by any user as passwords cannot be set on these properties.

1. Load a document using the **Load** method of **GcWordDocument** class.
2. Set the **BlockThemeOrSchemeSwitching** or **BlockQuickStyleSetSwitching** properties of the **DocumentProtection** class to true.

```
C#
//Load the Word document
```

```
GcWordDocument doc = new GcWordDocument();
doc.Load("SampleDoc.docx");

//Prevent users from changing the themes used in the document
doc.Settings.DocumentProtection.BlockThemeOrSchemeSwitching = true;

//Prevent users from changing the current style set
doc.Settings.DocumentProtection.BlockQuickStylesetSwitching = true;

//Save the document
doc.Save("FormattingRestrictions.docx");
```

[Back to Top](#)

Document Formatting Source

Usually, in a Word document, the content formatting is picked from default document style, parent content formatting or direct formatting. In order to detect the source of formatting properties, DsWord library provides the [GetPropertyValueSource](#) method in **GcWordDocument** class. This feature is useful for users, who want to programmatically determine the formatting source of Word objects in documents.

The **GetPropertyValueSource** method takes the formatting "property" as a parameter to find and return the object in the inheritance chain which is the source of the formatting property.

If the target property's value is determined by an object in the inheritance hierarchy, the type of the object returned by the **GetPropertyValueSource** method can be one of:

- Paragraph
- Run
- Table
- Row
- Cell
- ContentControl
- Style
- ConditionalStyle
- StyleCollection
- ListLevel
- PageSetup
- Settings

If the property belongs directly to an object, that object will be returned (and it can be of any type)

Detect Source Object of Formatting Properties

To detect source of a paragraph's formatting properties:

1. Create an instance of **GcWordDocument** and load the Word document having styles.
2. Get a source object of the "Font.Size" property set for a run within a paragraph formatted with Heading 1 style.

```
C#
public static void DetectFormatting()
{
    //Create a new GcWordDocument
    GcWordDocument doc = new GcWordDocument();

    //Get Heading 1 style
    Style style = doc.Styles[BuiltInStyleId.Heading1];

    //Add a run within a paragraph formatted with Heading 1 style
    doc.Body.Paragraphs.Add("text", style);

    //Get a source object of the "Font.Size" property from the created run
    object source = GcWordDocument.GetPropertyValueSource(doc.Body.Runs.First.Font.Size);

    //The source object must be "Heading1" style because the font size is
    defined here.
    Console.WriteLine("Formatting source name: " + ((Style)source).Name);
    Console.ReadLine();
}
```

```
}
```

Back to Top

For more information on how to modify theme colors using DsWord, see [DsWord sample browser](#).

Copy or Move Document Content

Copying or moving content in documents helps in organizing the document content in a relevant manner.

DsWord allows you to perform the copy or move operations on document content within the document or between documents. The [CopyTo](#) and [MoveTo](#) methods in the [RangeBase](#) class are used to achieve the same. While performing the copy or move operations, the formatting settings of the content can also be copied or cleared using the [FormattingCopyStrategy](#) enumeration.

Copy Document Content

To copy content within a document with or without formatting:

1. Load a document using the **Load** method of **GcWordDocument** class.
2. Copy the second paragraph after the third paragraph using the **CopyTo** method of the **RangeBase** class. To keep the formatting of content control, the **Copy** value of **FormattingCopyStrategy** enumeration has been passed as a parameter to the **CopyTo** method.
3. Copy the table after the fourth paragraph in the document. To clear the formatting of table, the **Clear** value of **FormattingCopyStrategy** enumeration has been passed as a parameter to this method.

```
C#
var doc = new GcWordDocument();
doc.Load("Test.docx");

//Copy the second paragraph after the third paragraph with formatting
doc.Body.Paragraphs[1].GetRange().CopyTo(doc.Body.Paragraphs[2].GetRange(), InsertLocation.After,
FormattingCopyStrategy.Copy);
doc.Save("Copy_WithinDoc_WithFormatting.docx");

//Copy table after the fourth paragraph without formatting
doc.Body.Tables[0].GetRange().CopyTo(doc.Body.Paragraphs[3].GetRange(), InsertLocation.After,
FormattingCopyStrategy.Clear);
doc.Save("Copy_WithinDoc_WithoutFormatting.docx");
```

Back to Top

To copy content between documents with or without formatting:

1. Load a document using the **Load** method of **GcWordDocument** class.
2. Create a new document in which you want to copy the content and add a new paragraph using the **Add** method of **ParagraphCollection** class.
3. Copy the table from first document, in the second document, using the **CopyTo** method of the **RangeBase** class. To keep the formatting of table, the **Copy** value of **FormattingCopyStrategy** enumeration has been passed as a parameter to the **CopyTo** method.
4. Copy the paragraph from first document, at the start of second document. To clear the formatting of paragraph, the **Clear** value of **FormattingCopyStrategy** enumeration has been passed as a parameter to this method.

```
C#
var doc1 = new GcWordDocument();
doc1.Load("Test.docx");

var doc2 = new GcWordDocument();
doc2.Body.Sections[0].GetRange().Paragraphs.Add("Using DsWord, content can be copied with or without
formatting in single document or between documents");

//Copy table from first document after the first paragraph of second document with formatting
doc1.Body.Tables[0].GetRange().CopyTo(doc2.Body.Paragraphs.First.GetRange(), InsertLocation.After,
FormattingCopyStrategy.Copy);
doc2.Save("Copy_BetweenDocs_WithFormatting.docx");

//Copy second paragraph of first document at the start of the second document without formatting
doc1.Body.Paragraphs[1].GetRange().CopyTo(doc2.Body, InsertLocation.Start,
FormattingCopyStrategy.Clear);
doc2.Save("Copy_BetweenDocs_WithoutFormatting.docx");
```

Back to Top

To copy content between documents and keep source document's formatting:

1. Define a style in the target document by using various properties like **Font** and **ParagraphFormat**. Add a paragraph using this style by using

Styles method of **StyleCollection** type.

- Similarly, define a style with the same name in the source document but with different set of properties. Add a paragraph using this style by using **Styles** method of **StyleCollection** type.
- Copy the paragraph from source document to target document using **KeepSource** value of **FormattingCopyStrategy** enumeration.

```
C#
// The name of the style used in both documents
const string styleName = "X-style";

// The target document
var doc = new GcWordDocument();

// Add a paragraph style to the target document
var xStyleTgt = doc.Styles.Add(styleName, StyleType.Paragraph);
xStyleTgt.Font.Color.RGB = Color.Blue;
xStyleTgt.Font.Bold = true;
xStyleTgt.ParagraphFormat.Indentation.RightIndent += 72;
// Add a paragraph with that style
var textTgt = $"This paragraph in the target document " +
    $"is associated with a custom paragraph style named \"{styleName}\". " +
    $"That style specifies the font to be blue and bold, and the whole paragraph " +
    $"is indented by 1\" on the right.";
doc.Body.Paragraphs.Add(textTgt, doc.Styles[styleName]);

// The source document, will copy from it to the target document keeping formatting
var docSrc = new GcWordDocument();
// Add a style with the same name to the source
var xStyleSrc = docSrc.Styles.Add(styleName, StyleType.Paragraph);
xStyleSrc.Font.Color.RGB = Color.Red;
xStyleSrc.Font.Italic = true;
xStyleSrc.ParagraphFormat.Alignment = ParagraphAlignment.Right;
xStyleSrc.ParagraphFormat.Indentation.LeftIndent += 72;
// Add a paragraph with the style, this paragraph will be copied to target
var textSrc = "This paragraph is copied from the source to the target DOCX. " +
    $"In the source DOCX, this paragraph was associated with a custom style also named \"
{styleName}\". " +
    $"That style specifies the font to be red and italic, the whole paragraph is right-aligned " +
    $"and indented 1\" from the left. " +
    $"Due to a conflict with the same-named but different style in the target document " +
    $"its name is changed to \"{styleName}1\".";
var paraSrc = docSrc.Body.Paragraphs.Add(textSrc, docSrc.Styles[styleName]);

// Copy paragraph from source to target using FormattingCopyStrategy.KeepSource
paraSrc.GetRange().CopyTo(doc.Body, InsertLocation.End, FormattingCopyStrategy.KeepSource);

//Save target document
doc.Save("keepsourceformatting.docx");
```

Back to Top

The output of above code will look like below in the target Word document:

This paragraph in the target document is associated with a custom paragraph style named "X-style". That style specifies the font to be blue and bold, and the whole paragraph is indented by 1" on the right.

This paragraph is copied from the source to the target DOCX. In the source DOCX, this paragraph was associated with a custom style also named "X-style". That style specifies the font to be red and italic, the whole paragraph is right-aligned and indented 1" from the left. Due to a conflict with the same-named but different style in the target document its name is changed to "X-style1".

Move Document Content

To move content within a document with or without formatting:

1. Load a document using the **Load** method of GcWordDocument class.
2. Move the second paragraph after the third paragraph using the **MoveTo** method of the **RangeBase** class. To keep the formatting of paragraph, the **Copy** value of **FormattingCopyStrategy** enumeration has been passed as a parameter to the MoveTo method.
3. Move table before the first paragraph in the document. To clear the formatting of content control, the **Clear** value of **FormattingCopyStrategy** enumeration has been passed as a parameter to this method.

C#

```
var doc = new GcWordDocument();
doc.Load("Test.docx");

//Move second paragraph after the third paragraph with formatting
doc.Body.Paragraphs[1].GetRange().MoveTo(doc.Body.Paragraphs[2].GetRange(), InsertLocation.After,
FormattingCopyStrategy.Copy);
doc.Save("Move_WithinDoc_WithFormatting.docx");

//Move table before the first paragraph without formatting
doc.Body.Tables[0].GetRange().MoveTo(doc.Body.Paragraphs.First.GetRange(), InsertLocation.Before,
FormattingCopyStrategy.Clear);
doc.Save("Move_BetweenDocs_WithoutFormatting.docx");
```

Back to Top

To move content between documents with or without formatting:

1. Load a document using the **Load** method of GcWordDocument class.
2. Create a new document in which you want to move the content and add a new paragraph using the **Add** method of ParagraphCollection class.
3. Move the table from first document, in the second document, using the **MoveTo** method of the RangeBase class. To keep the formatting of table, the **Copy** value of FormattingCopyStrategy enumeration has been passed as a parameter to the **MoveTo** method.
4. Move an image from first document, at the end of the last paragraph's run of the second document. To clear the formatting of image, the **Clear** value of **FormattingCopyStrategy** enumeration has been passed as a parameter to this method.

C#

```
var doc1 = new GcWordDocument();
doc1.Load("Test.docx");

var doc2 = new GcWordDocument();
doc2.Body.Paragraphs.Add("Using DsWord, content can be moved with or without formatting in single
document or between documents");

//Move table from first document before the first paragraph of second document with formatting
doc1.Body.Tables[0].GetRange().MoveTo(doc2.Body.Paragraphs.First.GetRange(), InsertLocation.Before,
FormattingCopyStrategy.Copy);
doc2.Save("Move_BetweenDocs_WithFormatting.docx");

//Move image from first document at the end of the last paragraph's run of the second document without
formatting
doc1.Body.Pictures.First.GetRange().MoveTo(doc2.Body.Paragraphs.Last.GetRange().Runs.First.GetRange(),
InsertLocation.End, FormattingCopyStrategy.Clear);
doc2.Save("Move_BetweenDocs_WithoutFormatting.docx");
```

Back to Top

For more information on how to copy or move document content using DsWord, see [DsWord sample browser](#).

Split or Merge Documents

DsWord allows you to easily split or merge documents by providing the [SplitDocument](#) and [MergeDocuments](#) methods in **GcWordDocument** class. While performing the split or merge operations, the formatting settings of the document can also be copied or cleared using the Copy or Clear value of [FormattingCopyStrategy](#) enumeration.

The **MergeDocuments** method appends the document content one after the another to create a new merged document and the **SplitDocument** method copies the content from specific ranges to create new documents.

Split Documents

To split a document:

1. Load a document using the **Load** method of **GcWordDocument** class.
2. Define an **IEnumerable** collection of document ranges using the **Range** class. The count of these ranges will determine the number of split documents to be created.
3. Split the document using the **SplitDocument** method of **GcWordDocument** class. To copy the formatting settings, use the **Copy** value of **FormattingCopyStrategy** enumeration.
4. Save the split documents as new documents using the **Save** method of **GcWordDocument** class.

```
C#  
  
//Load the document  
GcWordDocument doc = new GcWordDocument();  
doc.Load("SampleDoc.docx");  
  
//Define an IEnumerable collection of document ranges which are intended to be  
copied to the new document  
Range[] splitRanges = new Range[] { doc.Body.Sections[0].GetRange(),  
doc.Body.Sections[1].GetRange() };  
  
//Split the document into multiple documents  
IEnumerable<GcWordDocument> docs =  
doc.SplitDocument(FormattingCopyStrategy.Copy, splitRanges);  
  
//Save the individual documents  
List<GcWordDocument> splitDocs = docs.ToList<GcWordDocument>();  
splitDocs[0].Save("Doc1.docx");  
splitDocs[1].Save("Doc2.docx");
```

[Back to Top](#)

Merge Documents

To merge two documents:

1. Load the documents which you want to merge using the **Load** method of **GcWordDocument** class.
2. Define an **IEnumerable** collection of the documents you want to merge together.
3. Merge the documents using the **MergeDocuments** method of **GcWordDocument** class. To copy the formatting settings, use the **Copy** value of **FormattingCopyStrategy** enumeration.
4. Save the merged document as a new document using the **Save** method of **GcWordDocument** class.

```
C#  
  
//Load the documents
```



```
GcWordDocument doc1 = new GcWordDocument();
doc1.Load("Document1.docx");

GcWordDocument doc2 = new GcWordDocument();
doc2.Load("Document2.docx");

//Define an IEnumerable collection of documents to be merged
GcWordDocument[] docs = new GcWordDocument[] { doc1, doc2 };

//Merge the documents
GcWordDocument mergedDoc =
GcWordDocument.MergeDocuments(FormattingCopyStrategy.Copy, docs);

//Save the merged document
mergedDoc.Save("MergedDoc.docx");
```

Back to Top

For more information on how to split or merge documents content using DsWord, see [DsWord sample browser](#).

Page Settings

DsWord stores all the page setup attributes, such as page borders, size, margins, etc., as properties in the [PageSetup](#) class. These properties control the structure and layout of pages in a word document. DsWord also allows you to insert a page break which is especially required in case of a long document using [Type](#) property of the [Break](#) class. The [Type](#) property takes **Page** as a value from [BreakType](#) enumeration for specifying a page break. Moreover, DsWord lets you specify how a document is printed using [Type](#) property of the [MultiPagePrinting](#) class. The [Type](#) property takes the value from [MultiPagePrintingType](#) enumeration, so that the printed document can be bound as a booklet.

Set Page Properties

To set the page properties:

1. Access the page setup properties using [PageSetup](#) property of the [Section](#) class.
2. Set the page size properties. For example, set the orientation of the page using the [Orientation](#) property and paper size using the [PaperSize](#) property.
3. Set the page borders of the section using the [Borders](#) property and apply border to the pages using the [Border](#) class properties.

```
C#
GcWordDocument doc = new GcWordDocument();
doc.Load("SampleDoc.docx");
Section first = doc.Body.Sections.First;

first.PageSetup.Size.Orientation = PageOrientation.Landscape;
first.PageSetup.Size.PaperSize = PaperSize.PaperLetter;

first.PageSetup.Borders.AppliesTo = PageBorderAppliesTo.AllPages;
first.PageSetup.Borders.Left.LineStyle = LineStyle.BabyPacifier;
first.PageSetup.Borders.Right.LineStyle = LineStyle.BabyPacifier;
first.PageSetup.Borders.AlwaysInFront = true;

doc.Save("SetPageProperties.docx");
```

[Back to Top](#)

Set Page Number

To set the page numbering:

1. Access the page setup properties using **PageSetup** property of the **Section** class.
2. Set the page numbering properties. For example, set the starting page number using the [StartingNumber](#) property and the page number format using [NumberStyle](#) property of the [PageNumbering](#) class.
This displays the page numbering when you click on the scroll handle, which is the default behavior. However, if you want to display the page number at the bottom of the page, you can insert a footer in the page and use a simple field in it.
3. Append a footer using the [Footers](#) property and add a paragraph to it using the [Add](#) method.
4. Add a simple field in the paragraph using [Add](#) method of the [SimpleFieldCollection](#) class.

```
C#
doc.Load("SampleDoc.docx");
Section first = doc.Body.Sections.First;

//Set page numbering
first.PageSetup.PageNumbering.StartingNumber = 3;
first.PageSetup.PageNumbering.NumberStyle = NumberStyle.NumberInDash;
first.Footers[HeaderFooterType.Primary].Body.Paragraphs.Add("");
first.Footers[HeaderFooterType.Primary].Body.Paragraphs.First.GetRange().SimpleFields.Add("PAGE");

//Save the document
doc.Save("PageNumbering.docx");
```

[Back to Top](#)

Insert Page Break

To insert page break in a Word document:

1. Access a paragraph in a section. For example, access first paragraph of the first section.
2. Add a page break on the desired location in the paragraph using `AddBreak` method of the `TextCollection` class. For example, add the page break after first run of the first paragraph.
3. Set the break type to Page using the `BreakType` enum.

```
C#
doc.Load("SampleDoc.docx");

//Access the first paragraph of the first section
Section first = doc.Body.Sections.First;
Paragraph p1 = first.GetRange().Paragraphs.First;

//Insert page break
Break br1;
br1 = p1.GetRange().Runs.First.GetRange().Texts.AddBreak(BreakType.Page);

//Save the document
doc.Save("SampleDoc_PageBreak.docx");
```

[Back to Top](#)

Remove Page Break

To remove page break:

1. Get a list of all the breaks of type Page from the document.
2. Remove a page break using the `Delete` method. For example, remove the first page break.

```
C#
//Load the document
doc.Load("SampleDoc_PageBreak.docx");

//Get a list of all the breaks of type "Page" that exist in the document
var breaks = new List<Break>();
foreach (var text in doc.Body.Texts)
{
    Break x = text as Break;
    if (x != null && x.Type == BreakType.Page)
        breaks.Add(x);
}

//Remove the first page break
breaks[0].Delete();

//Save the document
doc.Save("SampleDoc_NoPageBreak.docx");
```

[Back to Top](#)

Multipage Printing

To print a multiple page document:

1. Access the options to print the multiple page document using `MultiPagePrinting` property of the `PageSetup` class.
2. Set the type for document printing using `Type` property of the `MultiPagePrinting` class.

```
C#
doc.Load("SampleDoc.docx");
//Two out of the three sheets in the document will be printed on one page
doc.Body.Sections.First.PageSetup.MultiPagePrinting.Type =
    MultiPagePrintingType.TwoPagesPerSheet;
```

```
doc.Save("MultiPagePrinting.docx");
```

Back to Top

For more information on how to work with pages and page breaks using DsWord, see [DsWord sample browser](#).

Styles

Document styles are the pre-defined set of formatting instructions which can be re-used any number of times in a document. For instance, a document style once defined for a list can be used to represent all other similar lists in the document.

DsWord provides you the ability to style a Word document using built-in styles. It offers different style types through [BuiltInStyleId](#) enumeration which can be applied to the corresponding content type. In addition to the built-in styles, DsWord also allows you to define styles on your own using the [Style](#) class. You can add a defined style to the style collection using [Add](#) method of the [StyleCollection](#) class and the style collection can be accessed using [Styles](#) property of the [GcWordDocument](#) class.

Create Document Styles

To create new style for a Word document:

1. Define a new style using the **Style** class and add it to the document using the **Add** method.
2. Access the content object on which a style has to be applied. For example, access first text run from the first paragraph.
3. Apply the defined style on the text run using [Style](#) property of the [Run](#) class.

```
C#
doc.Load("SampleDoc.docx");

//Access the first paragraph
Paragraph p = doc.Body.Sections.First.GetRange().Paragraphs.First;

//Access the text in the first run
Run run = p.GetRange().Runs.First;

// Create a new char style "style1" for the first half
Style style1 = doc.Styles.Add("Style1", StyleType.Character);
style1.Font.Name = "Times New Roman";
style1.Font.Size = 16;
style1.Font.Bold = true;
style1.Font.Italic = true;
style1.Font.Color.RGB = Color.Blue;
style1.Font.Underline = Underline.Thick;

//Apply style to the text run
run.Style = style1;

//Save the document
doc.Save("StyleAdded.docx");
```

[Back to Top](#)

Modify Document Styles

To modify document style:

1. Access the content object on which a style is applied. For example, access first text run of the first paragraph.
2. Modify the applied styles on the text run. For example, font name and color.

3. Apply the modified style on the text run using **Style** property of the **Run** class.

```
C#
doc.Load("StyleAdded.docx");

//Access the first paragraph
Paragraph para1 = doc.Body.Sections.First.GetRange().Paragraphs.First;

//Access the first run
Run run = para1.GetRange().Runs.First;

//Modify the existing style's font name and color
Style s1 = run.Style;
s1.Font.Color.RGB = Color.Brown;
s1.Font.Name = "Arial";

//Apply style to the run
run.Style = s1;

//Save the document
doc.Save("ModifyStyles.docx");
```

[Back to Top](#)

Delete Document Styles

To delete the style applied on a Word document:

1. Access the content object on which a style is applied. For example, access first text run of the first paragraph.
2. Delete the style applied on the accessed range using [Delete](#) method of the [Style](#) class.

```
C#
doc.Load("StyleAdded.docx");

//Access the first paragraph
Paragraph p = doc.Body.Sections.First.GetRange().Paragraphs.First;

//Access the text in the first run
Run run = p.GetRange().Runs.First;

//Delete the style applied on the first half
run.Style.Delete();

//Save the document
doc.Save("StyleDeleted.docx");
```

[Back to Top](#)

Add Linked Styles

Linked styles behave as either a character style or a paragraph style, depending on what you select. So, if you click on a paragraph or select a paragraph and then apply a linked style, the style is applied as a paragraph style. However, if you select a word or phrase in the paragraph and then apply a linked style, the style is applied as a character style,

with no effect on the paragraph as a whole. Linked styles also allow a user to update the style properties of all the paragraphs and words or phrases with the same type of style at once.

DsWord supports Linked Styles by using [AddLinkedStyle](#) method of [StyleCollection](#) class, and also provides [HideLinkedCharacterStyles](#) property in [GcWordDocument](#) class to hide the character style part of the linked style in our API in order to achieve the same behavior as MS Word.

DsWord also allows a user to:

- Use a linked style in [Add](#) or [Insert](#) method of [RunCollection](#) for style argument
- Apply a linked style to the [Style](#) property of [Run](#) class
- Apply a linked style to the [Style](#) property of [ContentControl](#) class
- Apply a linked style to the [Style](#) property of [FormattedMark](#) class
- Apply a linked style to the [Style](#) property of [FindFormatting](#) class.

Refer to the following example code to add paragraph and character linked style:

```
C#
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Change the default value of HideLinkedCharacterStyles.
doc.HideLinkedCharacterStyles = false;

// Create a linked style.
Style linkedStyle = doc.Styles.AddLinkedStyle("Linked Style",
doc.Styles[BuiltInStyleId.Heading1]);

// Change linked style formatting.
linkedStyle.Font.Bold = true;

// Apply linked style to a paragraph.
doc.Body.Paragraphs.Add("This paragraph is formatted with a paragraph linked style.",
linkedStyle);

// Apply linked style to a run.
Paragraph paragraph = doc.Body.Paragraphs.Add("This paragraph ",
doc.Styles[BuiltInStyleId.Normal]);
Run run = paragraph.GetRange().Runs.Add("is formatted with a character linked
style.", linkedStyle);

// Change HideLinkedCharacterStyles property value.
doc.HideLinkedCharacterStyles = true;

// Save the Word document.
doc.Save("LinkedStyles.docx", DocumentType.Document);
```

The output of the above-mentioned example code is shown in the image below:

This paragraph is formatted with a paragraph linked style.

This paragraph **is formatted with a character linked style.**

Refer to the following example code to add linked style for [ContentControl](#), [FormattedMark](#), and [FindFormatting](#):

```
C#  
  
// Initialize GcWordDocument.  
GcWordDocument doc = new GcWordDocument();  
  
// Create a linked style  
Style style = doc.Styles.AddLinkedStyle("linked");  
  
// Add a content control.  
ContentControl cc = doc.Body.ContentControls.Add(ContentControlType.RichText, false);  
  
// Set the linked style to content control.  
cc.Style = style;  
  
// Add a paragraph.  
Paragraph p1 = cc.Content.GetRange().Paragraphs.Add("paragraph text");  
  
// Set the linked style to its mark.  
p1.Mark.Style = style;  
  
// Add a run and set the linked style to it.  
Run run = p1.GetRange().Runs.Add(style);  
Text text = run.GetRange().Texts.Add("text to find");  
  
// Add another paragraph and set the linked style to it.  
Paragraph p2 = cc.Content.GetRange().Paragraphs.Add("paragraph to find", style);  
  
// Create find options and set the linked style to it.  
FindOptions fo = new FindOptions(doc);  
fo.FormattingOptions.Style = style;  
var res = doc.Body.Find(string.Empty, fo).ToArray();  
GrapeCity.Documents.Word.Range found = res[0].Range;  
  
// Save the Word document.  
doc.Save("LinkedStyles.docx");
```

Back to Top

For more information on how to apply different document styles using DsWord, see [DsWord sample browser](#).

Text Effect

DsWord lets you set following effects to your text:

- Fill and Line Effects
- Shadow Effects

Fill and Line Effect

DsWord lets you apply effects to your fonts by using the [Font.Fill](#) and [Font.Line](#) properties. You can set both Fill and Line properties to **NoFill**, **Solid** or **Gradient** type only through Type property. The Type property accepts values from **FillType** enumeration.

Font Fill and Line styles

Font.Fill and **Font.Color** properties are interdependent. Following table demonstrates their dependency on each other:

Font.Fill	Font.Color
Not specified	Font.Fill takes the value from Font.Color as Solid type.
Any of Font.Fill properties are specified.	Font.Color uses Font.Fill.SolidFill properties.
Font.Fill type is specified as Gradient .	Font.Fill and Font.Color are independent of each other. DsWord uses Font.Fill to render the font.

In addition, you can also set the Font.Line property to give your fonts an outline effect. Font.Line and Font.Color are independent of each other.

```
C#
// Specify gradient fill for the font:
var fill = style.Font.Fill;
fill.Type = FillType.Gradient;
var gradient = fill.GradientFill;
gradient.Angle = 64;
gradient.Stops[0].Color.RGB = Color.LightSeaGreen;
gradient.Stops[1].Color.RGB = Color.Orange;

// Specify the outline for the font:
var line = style.Font.Line;
line.Width = 1;
fill = line.Fill;
fill.Type = FillType.Solid;
fill.SolidFill.RGB = Color.Blue;

// Add sample text with the new style:
doc.Body.Paragraphs.Add("Font Fill and Line styles.", style);
```

Shadow Effect

The shadow effect adds additional depth to your text. DsWord allows you to apply shadow effects to text using built-in shadow effects and using custom shadow effects. To apply built-in shadow effect to a text, you can use [ApplyBuiltInShadow](#) method of the [TextEffects](#) class that accepts [BuiltInShadowId](#) enum as its parameter. However, for applying custom shadow effect to a text, you can use **OuterShadow** type of the [TextEffects](#) class and set its properties.

The code below shows how to apply built-in shadow to text using the ApplyBuiltInShadow method:

```
C#
// apply top right offset shadow effect to the text
style.Font.Effects.ApplyBuiltInShadow(BuiltInShadowId.OffsetTopRight);
doc.Body.Paragraphs.Add("Font Shadow", style);
```

The code below shows how to apply custom shadow to a text using the OuterShadow type:

```
C#
// apply top right offset shadow effect to the text
OuterShadow shadow = style.Font.Effects.Shadow;
shadow.Alignment = RectangleAlignment.BottomLeft;
shadow.Angle = 315f;
shadow.Blur = 4f;
```

```
shadow.Distance = 3f;  
shadow.ScaleX = 100f;  
shadow.ScaleY = 100f;  
shadow.SkewX = 0f;  
shadow.SkewY = 0f;  
shadow.Color.RGB = Color.Black;  
shadow.Color.Transparency = 0.6f;  
doc.Body.Paragraphs.Add("Font Shadow", style);
```

Reflection Effect

DsWord allows you to apply reflection effects to text using built-in reflection effects and custom reflection effects. To apply built-in reflection effect to a text, you can use [ApplyBuiltInReflection](#) method of the **TextEffects** class that accepts [BuiltInReflectionId](#) enum as its parameter. However, for applying custom reflection effect to a text, you can use **Reflection** type of the TextEffects class and set its properties.

The code below shows how to apply built-in reflection to text using the [ApplyBuiltInReflection](#) method:

```
C#  
  
// apply top half touching reflection effect to the text  
style1.Font.Effects.ApplyBuiltInReflection(BuiltInReflectionId.HalfTouching);  
doc.Body.Paragraphs.Add("Font Reflection.", style1);
```

The code below show how to apply custom reflection to text using the [Reflection](#) type:

```
C#  
  
// Custom text reflection:  
var style0 = doc.Styles.Add("style0", StyleType.Paragraph);  
style0.Font.Size = 48;  
// apply top half touching reflection effect to the text  
Reflection reflection = style0.Font.Effects.Reflection;  
reflection.Distance = 0f;  
reflection.Angle = 90f;  
reflection.Blur = 0.5f;  
reflection.Alignment = RectangleAlignment.BottomLeft;  
reflection.ScaleX = 100f;  
reflection.ScaleY = -100f;  
reflection.SkewX = 0f;  
reflection.SkewY = 0f;  
reflection.StartOpacity = 60f;  
reflection.EndOpacity = 0.9f;  
reflection.StartOpacityPosition = 0f;  
reflection.EndOpacityPosition = 58f;  
reflection.OpacityAngle = 90f;  
doc.Body.Paragraphs.Add("Font Reflection.", style0);
```

To view the code in action, see [Reflection Effect demo](#) sample.

Glow Format

DsWord allows you to apply glow effects to text using built-in glow effects and custom glow effects. To apply built-in glow effect to a text, you can use [ApplyBuiltInGlow](#) method of the **TextEffects** class that accepts [BuiltInGlowId](#) enum as its parameter. However, for applying custom glow effect to a text, you can use **Glow** type of the TextEffects class and set its properties.

The code below shows how to apply built-in glow to a shape using the [ApplyBuiltInGlow](#) method:

```
C#
```

```
// apply 5 point accent 5 glow effect to the text
style.Font.Effects.ApplyBuiltInGlow(BuiltInGlowId.Radius5Accent5);
doc.Body.Paragraphs.Add("Font Glow", style);
```

The code below shows how to apply custom glow effect to a shape:

```
C#
// apply 5 point accent 6 glow effect to the text
Glow glow = style.Font.Effects.Glow;
glow.Radius = 5f;
glow.Color.ThemeColor = ThemeColorId.Accent6;
glow.Color.Transparency = 0.6f;
doc.Body.Paragraphs.Add("Font Glow", style);
```

To view the code in action, see [Glow Effect Demo](#) sample.

3D Effect

The 3D effects refer to the effects applied in three spatial dimensions: width, height, and depth. The 3D effects give the text a unique and artistic look. For applying 3D effects, DsWord provides the [ThreeDFormat](#) and [ThreeDScene](#) classes and the [ApplyEffectsPreset](#) method. The [ApplyEffectsPreset](#) method allows a user to add preset 3D effects. However, the [ThreeDFormat](#) and [ThreeDScene](#) classes allow a user to add custom 3D effects. DsWord also provides an [IFixedFormattingBag](#) interface for all text effects and allows a user to understand and control whether the text has a direct effect or is inherited from its parent.

Preset 3D Effect	Custom 3D Effect
	

Refer to the following example code to add a preset 3D effect to the text:

```
C#
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Add a paragraph to the document.
Paragraph paragraph = doc.Body.Paragraphs.Add();
Run run = paragraph.GetRange().Runs.Add("3D Effects");

// Set font size.
run.Font.Size = 72f;

// Apply preset text 3D effect.
run.Font.ApplyEffectsPreset(FontEffectsPreset.Preset5);

// Save the Word document.
doc.Save("Preset3DEffects.docx");
```

Refer to the following example code to add a custom 3D effect to the text:

```
C#
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Add a paragraph to the document.
Paragraph paragraph = doc.Body.Paragraphs.Add();
Run run = paragraph.GetRange().Runs.Add("3D Effects");

// Set text style.
GrapeCity.Documents.Word.Font font = run.Font;
font.Size = 72f;
font.Bold = true;
```

```
font.Color.ThemeColor = ThemeColorId.Accent3;
font.Line.Fill.Type = FillType.NoFill;

// Apply 3D effect.
ThreeDFormat threeD = font.Effects.ThreeDFormat;
threeD.Material = MaterialType.Matte;
threeD.Depth.Width = 4.5f;
threeD.TopBevel.Type = BevelType.Angle;
threeD.TopBevel.Width = 5f;
threeD.TopBevel.Height = 1f;

// Save the Word document.
doc.Save("3DEffects.docx");
```


Limitation

DsWord does not support export of font fill and line, font shadow, reflection, and glow effects to PDF and images.

Export

Exporting a Word document to PDF and images is a very common use case and is generally required for various business purposes. Some of the advantages are security, cross-platform compatibility, availability of free readers, reduced file size etc.

DsWord allows you to export Word documents to PDF format with the help of [DS.Documents.Word.Layout](#) (DsWordLayout) package. Hence, it is mandatory to install **DS.Documents.Word.Layout** package while using the export functionality in DsWord.

 **Note:** The old **DS.Documents.Layout** package is replaced by the new [DS.Documents.Word.Layout](#) package which has been refactored to improve the functionality.

The **GcWordLayout** class in GcWordLayout package separates the layout of source Word document from the exported PDF formats. You can also use the **WordLayoutSettings** class to perform various customizations of the set of pages being exported.

Export to PDF

The **SaveAsPdf** method of **GcWordLayout** class can be used to export Word documents to PDF. You can also set various PDF options like compression level, back color, metadata, PDF version by using properties of **PdfOutputSettings** class.

To export a Word document to PDF:

1. Load a Word document in **GcWordLayout** instance.
2. Save the Word document as PDF by using **SaveAsPdf** method of GcWordLayout class.
3. Use **PdfOutputSettings** class to specify the **CompressionLevel** as Fastest.

```
C#  
  
var wordDoc = new GcWordDocument();  
wordDoc.Load("SimpleText.docx");  
using (var layout = new GcWordLayout(wordDoc))  
{  
    // save the whole document as PDF  
    layout.SaveAsPdf("SimpleText.pdf", null, new PdfOutputSettings() {  
        CompressionLevel = CompressionLevel.Fastest });  
}
```

Export to Image

You can save Word Documents to TIFF, PNG and JPEG image formats by using **SaveAsTiff**, **SaveAsPng** and **SaveAsJpeg** methods respectively. The properties of **ImageOutputSettings** class can be used to choose from various options like zoom, back color and resolution while saving to image files.

To export a Word document to TIFF image format:

1. Load a Word document in **GcWordLayout** instance.
2. Save the Word document as TIFF image by using **SaveAsTiff** method of GcWordLayout class.
3. Use **ImageOutputSettings** class to specify the **Compression** as Deflate.

```
C#  
  
var wordDoc = new GcWordDocument();  
wordDoc.Load("JsFrameworkExcerpt.docx");
```

```
using (var layout = new GcWordLayout(wordDoc))
{
    // save a few pages of the Word document as TIFF
    layout.SaveAsTiff("JsFrameworkExcerpt.tiff", new OutputRange("2, 6-7"),
        new ImageOutputSettings() { Zoom = 2f }, new TiffFrameSettings() { Compression =
            TiffCompression.Deflate });
}
```

To export a single page of Word document to JPEG image format:

1. Create a new Word document by instantiating the **GcWordDocument** class.
2. Add content to the document by using **Add** method of ParagraphCollection class and specifying various BuiltInStyles by using **BuiltInStyleId** enumeration.
3. Load the document in **GcWordLayout** instance.
4. Save the first page of Word document as JPEG image by using **SaveAsJpeg** method of GcWordLayout class.
5. Use **ImageOutputSettings** class to specify the Zoom and BackColor properties.

```
C#
var doc = new GcWordDocument();
var sec = doc.Body.Sections.First;
var pars = sec.GetRange().Paragraphs;

// Title
pars.Add("Some Common Built-in Styles (Title)",
    doc.Styles[BuiltInStyleId.Title]);

// Subtitle
pars.Add("Demonstration of some of the built-in styles. (Subtitle)",
    doc.Styles[BuiltInStyleId.Subtitle]);

// Headings 1-4
var heading1 = pars.Add("Heading 1", doc.Styles[BuiltInStyleId.Heading1]);
var heading2 = pars.Add("Heading 2", doc.Styles[BuiltInStyleId.Heading2]);
var heading3 = pars.Add("Heading 3", doc.Styles[BuiltInStyleId.Heading3]);
var heading4 = pars.Add("Heading 4", doc.Styles[BuiltInStyleId.Heading4]);

// Character styles
var p = pars.Add("In this paragraph we demonstrate some of the built-in
character styles. ");
var runs = p.GetRange().Runs;
runs.Add("This run uses the 'Strong' style. ",
    doc.Styles[BuiltInStyleId.Strong]);
runs.Add("A run of normal text. ");
runs.Add("This run uses 'Emphasis' style. ",
    doc.Styles[BuiltInStyleId.Emphasis]);
runs.Add("A run of normal text. ");
runs.Add("This run uses 'Intense Emphasis' style. ",
    doc.Styles[BuiltInStyleId.IntenseEmphasis]);
runs.Add("A run of normal text. ");
runs.Add("This run uses 'Subtle Emphasis' style. ",
    doc.Styles[BuiltInStyleId.SubtleEmphasis]);
```

```
pars.Add("The End.");
using (var layout = new GcWordLayout(doc))
{
    layout.Pages[0].SaveAsJpeg("example.jpg", new ImageOutputSettings() { Zoom = 2f,
    BackColor = Color.Yellow });
}
```

For more information on how to convert a Word document into PDF and image formats using DsWord, see [DsWord sample browser](#).

Export to SVG

In addition to above mentioned common image formats, DsWord also lets you save the Word document pages as SVG or its compressed format SVGZ. You can use **SaveAsSvg** and **ToSvgz** methods of the **GrapeCity.Documents.Word.Layout.Page** class to export an instance of word page to SVG file or stream(.svg) or a byte array(.svgz).

C#

```
var wordDoc = new GcWordDocument();
wordDoc.Load("StatementOfWork.docx");
using (var la = new GcWordLayout(wordDoc))
{
    var page = la.Pages[0];

    // Render a Word page to the .svg file
    page.SaveAsSvg("StatementOfWork.svg", new ImageOutputSettings() { Zoom = 1f },
    new XmlWriterSettings() { Indent = true });

    // Render a Word page to the byte array with compressed data in SVGZ format
    var svgData = page.ToSvgz();
    File.WriteAllBytes("StatementOfWork.svgz", svgData);
}
```

Limitations

DsWord has a few limitations while exporting to PDF and image file formats. Some of them, in particular, are:

- While exporting to SVG, text is always rendered as graphics using paths. Hence, resulting .svg files for text pages are large and it is not possible to select or copy text on the SVG images opened in the browsers.
- Objects that are not supported by the DsWord OM are not exported.
- Footnotes are not exported.
- Export of text boxes is supported, but linked text boxes, text outlines, gradients, font fill and line effect, and the "Do not rotate text" option are not supported.
- When exporting shapes, custom shapes, ink shapes, sketched lines, gradient lines, and compound lines are not supported.
- Only stored values in fields are exported (i.e. no recalculation), with the following exceptions:
 - Page numbering is supported.
 - Partial hyperlink field is supported.
- Comments can optionally be exported, but complete formatting is not retained.

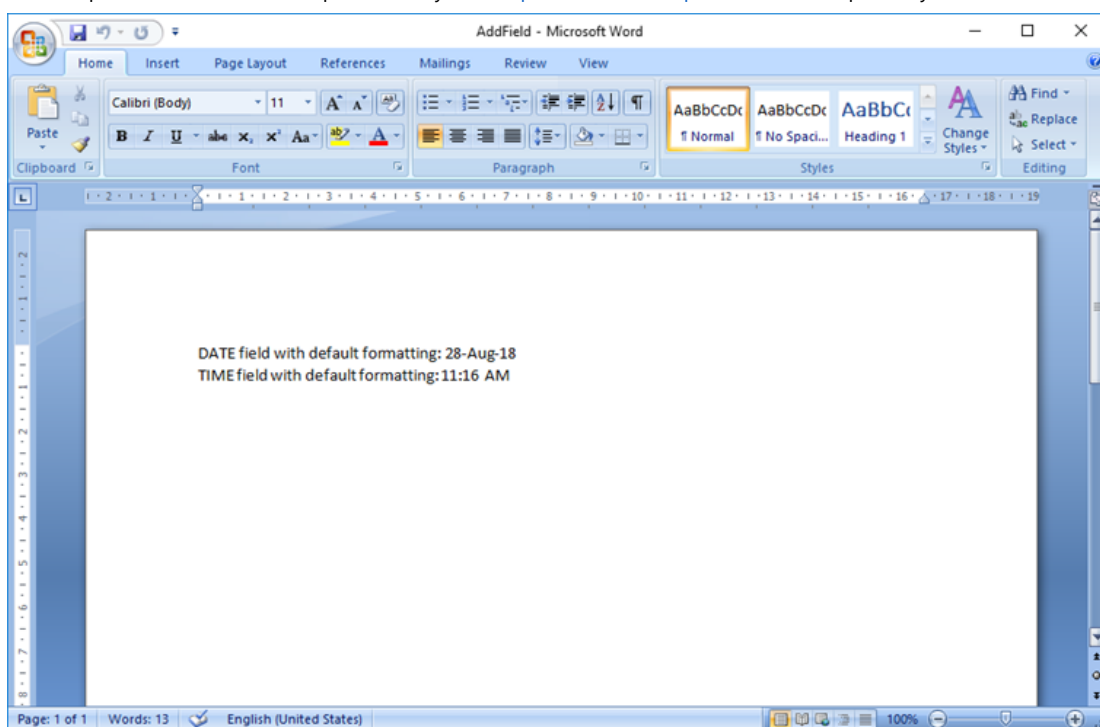
Fields

Fields in word document are used as placeholders to display the dynamic information such as date, page number, table of contents etc. These fields are defined using a combination of a FieldCode and FieldResult. The FieldCode represents a set of instructions which are evaluated to generate the dynamic content. When the document is rendered, the field codes implicitly execute the set of instruction and inserts the text, graphics, etc. in the document. The text or element that is inserted is referred to as FieldResult.

MS Word has a defined set of FieldCodes which are supported in DsWord. All these FieldCodes can be added to the word document either as a SimpleField or as a ComplexField depending on whether the FieldResult is rendered as a single run or multiple runs.

For example, the "DATE" field code inserts the current date in the document, which is a single run of text. When using this FieldCode in a document, the inserted date has the same formatting such as font, font size etc. In such scenario, the DATE field should be implemented as a SimpleField as the FieldResult is to be rendered as a single run. But, in case you need to render each part of the date with different formatting, three runs would be required to render a single date. Hence, in this case, the DATE FieldCode should be implemented as a ComplexField.

DsWord supports two types of field elements, simple field element and complex field element, to hold such dynamic information. These simple and complex field elements are represented by the [SimpleField](#) and [ComplexField](#) class respectively.



Add Simple Field

To add a simple field in a document, you can use [Add](#) method of the [SimpleFieldCollection](#) class. The following sample code adds two Fields i.e DATE and TIME to the Document using the SimpleField class.

```
C#
Section section = doc.Body.Sections.First;

// Add the paragraph:
section.GetRange().Paragraphs.Add("Simple Field Example");
var p0 = section.GetRange().Paragraphs.Add("DATE field with default formatting: ");

//Add Date Field
p0.GetRange().SimpleFields.Add("DATE \@ \"d-MMM-yy\"");

//Add a Run in the paragraph
p0.GetRange().Runs.Add("\rTIME field with default formatting: ");

//Add Time Field
p0.GetRange().SimpleFields.Add("TIME");
```



```
//Save the document
doc.Save("AddSimpleField.docx");
```

[Back to Top](#)

Modify Simple Field

To modify a simple field:

1. Access the field collection using [SimpleFields](#) property of the [RangeBase](#) class and get a reference to a specific field. For example, the first field of the collection.
2. Modify the field code using [Code](#) property of the [SimpleField](#) class.

```
C#
doc.Load("AddSimpleField.docx");

//Modify the simple field
doc.Body.Sections.First.GetRange().Paragraphs[2].GetRange().SimpleFields.First.Code =
    "DATE \@ \"dd-MM-yy\"";

//Save the document
doc.Save("ModifySimpleField.docx");
```

[Back to Top](#)

Delete Simple Field

To delete a simple field, access a simple field from the field collection using [SimpleFields](#) property of the [RangeBase](#) class and delete it using the [Delete](#) method.

```
C#
doc.Load("AddSimpleField.docx");

//Delete a simple field
doc.Body.Sections.First.GetRange().SimpleFields.First.Delete();

//Save the document
doc.Save("DeleteSimpleField.docx");
```

[Back to Top](#)

Add Complex Field

To add a complex field in a document, you can use [Add](#) method of the [ComplexFieldCollection](#) class.

The following sample code creates a ComplexField using IF and DATE fields to render a sentence in the document to notify the users whether today is a new year day or not. Here, we are rendering "not" with bold formatting when DATE field result is not equal to "1-1". Follow the steps below to add the complex field:

1. Add a complex field with instruction using [Add](#) method of the [ComplexFieldCollection](#) class. For example, add a complex field with "IF" instruction.
2. Add a complex field to the complex field code range. For example, add a complex field with "Date" instruction to the complex field code range.
3. Access the field code from the field code collection using the [CodeFields](#) property of the [ComplexField](#) class.
4. Add a field code to the field code collection using [Add](#) method of the [FieldCodeCollection](#) class.

```
C#
Section section = doc.Body.Sections.First;

// create a paragraph and get its range
section.GetRange().Paragraphs.Add("Complex Field Example");
var pr = section.GetRange().Paragraphs.Add().GetRange();

// add a static phrase "It's "
pr.Runs.Add("It's ");

// add a complex field with "IF" instruction
```

```
var f = pr.ComplexFields.Add("IF ");

// add a complex field with "DATE" instruction.
f.GetCodeRange().ComplexFields.Add(" DATE \@ \"M-d\" ");

//Add additional instruction to the "IF" field to compare the nested
//DATE field result with "1-1" and if it's not true - return "not" word.
// also make the "not" word (if visible) bold.
f.CodeFields.Add("<> \"1-1\" \"not \") .ParentRun.Font.Bold = true;

// add a static phrase "new year's day!"
pr.Runs.Add("new year's day!");

//Save the document
doc.Save("AddComplexField.docx");
```

[Back to Top](#)

Modify Complex Field

To modify a complex field:

1. Access a complex field from the field collection using **ComplexFields** property of the RangeBase class.
2. Access a field code from the field code collection using the **CodeFields** property of the ComplexField class. For example, access the second field of the collection.
3. Modify the field code value using the **Value** property.

```
C#
doc.Load("AddComplexField.docx");

//Modify complex field
doc.Body.Sections.First.GetRange().Paragraphs[2].GetRange().ComplexFields.First.CodeFields[1].Value =
    "<> \"12-31\" \"not \")";
doc.Body.Sections.First.GetRange().Paragraphs[2].GetRange().Runs[10].GetRange().Texts.First.Value =
    "year's last day";

//Save the document
doc.Save("ModifyComplexField.docx");
```

[Back to Top](#)

Delete Complex Field

To delete a complex field, access a complex field from the field collection using **ComplexFields** property of the **RangeBase** class and delete it using **Delete** method of the ComplexField class.

```
C#
doc.Load("AddComplexField.docx");

//Delete complex field
doc.Body.Sections.First.GetRange().ComplexFields.First.Delete();

doc.Save("DeleteComplexField.docx");
```

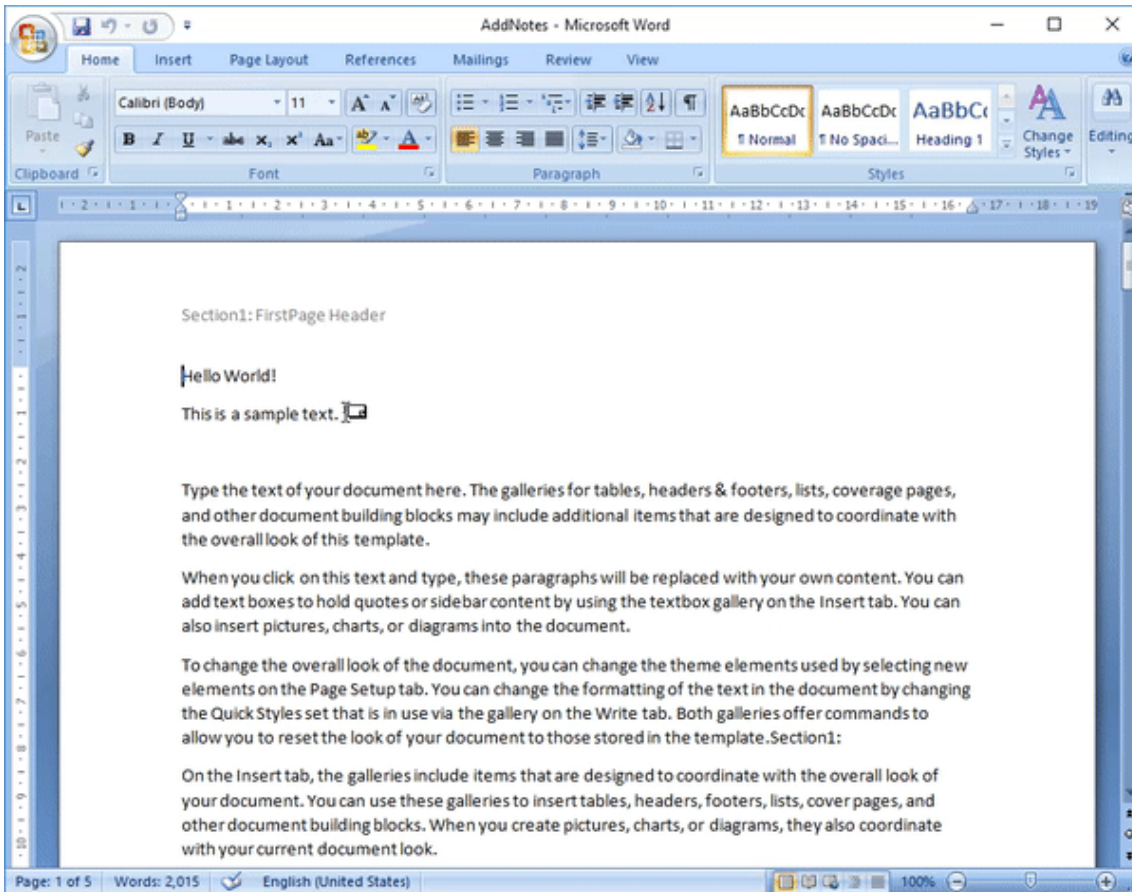
[Back to Top](#)

For more information on how to implement fields using DsWord, see [DsWord sample browser](#).

Footnote and Endnote

The purpose of using footnotes and endnotes in a document is to explain or provide additional information in a document like a reference or credits to something or someone mentioned in a document. The difference between footnotes and endnotes is that a footnote is displayed at the bottom of each page of a document and endnote appears at the end of a section or a document.

DSWord allows you to add a footnote and endnote using `Add` method of the `FootnoteCollection` class and `EndnoteCollection` class respectively. It lets you set the footnote and endnote position, numbering rule and number style for a document or a section using the `FootnoteOptions` and `EndnoteOptions` class respectively. These classes can be accessed using `FootnoteOptions` and `EndnoteOptions` properties of the `Settings` class respectively.



Add Footnote and Endnote

To add a footnote and endnote:

1. Set the footnote and endnote options using the `FootnoteOptions` property and the `EndnoteOptions` property respectively.
2. Access a content object to which a footnote and endnote needs to be added. For example, access a section of the document.
3. Add a footnote using `Add` method of the `FootnoteCollection` class.
4. Add an endnote using `Add` method of the `EndnoteCollection` class.

```
C#
doc.Load("HeaderFooter.docx");

//Set footnote options
doc.Settings.FootnoteOptions.Location = FootnoteLocation.BottomOfPage;
doc.Settings.FootnoteOptions.NumberingRule = FootnoteNumberingRule.Continuous;
```

```
//Add footnote in first section
var section = doc.Body.Sections.First;
section.GetRange().Paragraphs.First.GetRange().Footnotes.Add("This is a test
footnote.");

//Add footnote in second section
var section2 = doc.Body.Sections[1];
var secondRun =
section2.GetRange().Paragraphs.First.GetRange().Runs.Insert(InsertLocation.End);
secondRun.GetRange().Texts.Add("This is a reference text for a footnote.");
secondRun.GetRange().Footnotes.Add("Footnote for second section");

//Set endnote options
doc.Settings.EndnoteOptions.Location = EndnoteLocation.EndOfDocument;
doc.Settings.EndnoteOptions.NumberingRule = EndnoteNumberingRule.Continuous;

//Add endnote at the end of the document
section.GetRange().Paragraphs[1].GetRange().Endnotes.Add("This is a test endnote.");

//Save the document
doc.Save("AddNotes.docx");
```

[Back to Top](#)

Modify Footnote and Endnote

To modify a footnote and endnote:

1. Modify the footnote and endnote option using the **FootnoteOptions** property and the **EndnoteOptions** property respectively. For example, modify the location.
2. Access the footnote and endnote which needs to be modified. For example, access the first footnote and endnote added to the document.
3. Modify text of the footnote and endnote added to the document using **Value** property of the **Text** class.

```
C#
doc.Load("AddNotes.docx");

//Modify footnote options
doc.Settings.FootnoteOptions.Location = FootnoteLocation.EndOfSection;

//Modify the footnote in first section
var section = doc.Body.Sections.First;
section.GetRange().Paragraphs.First.GetRange().Footnotes.First.Body.Texts.First.Value
= "This" +
    " is a modified test footnote.";

//Modify endnote
doc.Settings.EndnoteOptions.Location = EndnoteLocation.EndOfSection;

//Modify the endnote in first section
section.GetRange().Paragraphs[1].GetRange().Endnotes.First.Body.Texts.First.Value =
"This" +
    " is a modified test endnote.";
```

```
//Save the document
doc.Save("ModifyNotes.docx");
```

[Back to Top](#)

Delete Footnote and Endnote

To delete a footnote and endnote, access the footnote and endnote and delete them using [Delete](#) method of the [ContentObject](#) class.

```
C#
doc.Load("AddNotes.docx");

//Add footnote in first section.
var section2 = doc.Body.Sections[1];
section2.GetRange().Paragraphs.First.GetRange().Footnotes.First.Delete();

//Delete EndNote
var section = doc.Body.Sections.First;
section.GetRange().Paragraphs[1].GetRange().Endnotes.First.Delete();

//Save the document
doc.Save("DeleteNotes.docx");
```

[Back to Top](#)

Set Numbering Style

To set numbering style of footnotes and endnotes:

1. Access the footnote and endnote options using the **FootnoteOptions** and **EndnoteOptions** property respectively.
2. Set the numbering style for the footnotes and endnotes using **NumberStyle** property of the FootnoteOptions and EndnoteOptions class respectively, which takes value from the [NumberStyle](#) enumeration.

```
C#
doc.Load("AddNotes.docx");

//Set footnote numbering style
doc.Settings.FootnoteOptions.NumberStyle = NumberStyle.DecimalEnclosedCircle;

//Set endnote numbering style
doc.Settings.EndnoteOptions.NumberStyle = NumberStyle.NumberInDash;

//Save the document
doc.Save("SetNumbering.docx");
```

[Back to Top](#)

For more information on how to implement footnotes and endnotes in a Word document using DsWord, see [DsWord sample browser](#).

Embedded Fonts

Custom embedded fonts are significant part of document portability. With fonts embedded in it, user can always open the word document with correct fonts even if the fonts do not exist on his system. This feature is especially useful while rendering word documents which use rare fonts not available on a usual system.

DsWord provides extensive API to enable users to define custom fonts, embed them in the Word document, apply them to the runs and so on.

Create and Embed Custom Fonts

DsWord allows you to create custom fonts using the `FontInfo` class which provides various properties to define custom font. The collection of these `FontInfo` objects for a document is represented by `FontInfoCollection` class. Hence, to add a custom font to a document, you need to add it to `FontInfoCollection` using **Add** method. To embed a font into a document, you can use **Add** method of the `EmbeddedFontCollection` class which represents the collection of embedded fonts in a document.

```
C#  
  
// Add the custom font to the document  
// (in this case we simply duplicate "Times New Roman"):  
var font1 = doc.Fonts.Add(myFontName1);  
// Use "Times New Roman" font settings:  
font1.CharSet = FontCharSet.Ansi;  
font1.Family = FontFamily.Roman;  
font1.Pitch = FontPitch.Variable;  
font1.Panose = new List<byte> { 2, 2, 6, 3, 5, 4, 5, 2, 3, 4 };  
font1.Signature.CodePageRange1 = 0x000001ff;  
font1.Signature.CodePageRange2 = 0x00000000;  
font1.Signature.UnicodeRange1 = 0xE0002EFF;  
font1.Signature.UnicodeRange2 = 0xC000785B;  
font1.Signature.UnicodeRange3 = 0x00000009;  
font1.Signature.UnicodeRange4 = 0x00000000;  
// Load the "Times New Roman" font data:  
byte[] data1 = File.ReadAllBytes(@"..\..\Resources\Fonts\times.ttf");  
// Embed font data into the document:  
font1.Embedded.Add(EmbeddedFontType.Regular, FontDataType.ObfuscatedTrueTypeFont,  
data1);
```

Apply Custom Font

To apply custom fonts to runs in your document, you can use **Name** property of the `Font` class.

```
C#  
  
const string myFontName1 = "My Font 1";  
  
GcWordDocument doc = new GcWordDocument();  
  
// Use first of the fonts to be embedded:  
var p = doc.Body.Paragraphs.Add();  
var run = p.GetRange().Runs.Add($"Text rendered using embedded font \"  
{myFontName1}\".");
```

```
// Apply custom font to the run:  
run.Font.Name = myFontName1;
```

Embed Fonts using GrapeCity.Documents.Text Namespace

While creating word documents with DsWord, you can use **FontCollection** and **Font** classes from **GrapeCity.Documents.Text** namespace to embed fonts in a word document. You can embed a single font to the font collection of word document by using **Add** method of the **FontInfoCollection** class or, you can add a list of fonts to the font collection by using **Append** method. The **SaveOptions** class provides various embedded font settings which control how a document is saved.

```
C#  
  
// Load font from a .ttf file and set it as the font of the sample text:  
var font = GrapeCity.Documents.Text.Font.FromFile(Path.Combine("Resources",  
"Fonts", "SEGA.ttf"));  
// Assign the run's font to the font loaded from the file:  
run.Font.Name = font.FontFamilyName;  
// This embeds the font into the document, without this line the document won't  
display correctly  
// on a system that does not have the SEGA.ttf font installed:  
doc.Fonts.Add(font, true);
```

Get Embedded Font

To fetch the type and name of fonts embedded in a document, loop through the embedded fonts inside font collection of the document and use **Type** property of the **EmbeddedFont** and **Name** property of the **FontInfo** object respectively.

```
C#  
  
GcWordDocument doc = new GcWordDocument();  
  
// Load a DOCX that contains embedded font:  
doc.Load(@"..\..\..\EmbedFonts.docx");  
  
var embeddedFont = doc.Fonts.FirstOrDefault(fi_ => fi_.Embedded.Count >  
1)?.Embedded[0];  
  
doc.Body.Paragraphs.Add("Embedded fonts found in this document:",  
doc.Styles[BuiltInStyleId.Heading1]);  
var myListTemplate = doc.ListTemplates.Add(BuiltInListTemplateId.BulletDefault,  
"myListTemplate");  
  
// Enumerate the embedded fonts:  
foreach (var fi in doc.Fonts)  
{  
    foreach (var ef in fi.Embedded)  
    {  
        var p = doc.Body.Paragraphs.Add(doc.Styles[BuiltInStyleId.ListParagraph]);  
        p.ListFormat.Template = myListTemplate;  
        var run = p.GetRange().Runs.Add($"Found embedded font of type {ef.Type} in  
font \"{fi.Name}\".");  
    }  
}
```

```
run.Font.Name = fi.Name;
if (ef.Type == EmbeddedFontType.Bold)
    run.Font.Bold = true;
else if (ef.Type == EmbeddedFontType.Italic)
    run.Font.Italic = true;
else if (ef.Type == EmbeddedFontType.BoldItalic)
    run.Font.Bold = run.Font.Italic = true;
}
}
//Save Document
doc.Save("GetEmbeddedFonts.docx");
```

Find and Remove Embedded Fonts

To find and remove embedded fonts from a document, you need to loop through the fonts collection of the document and call **Clear** method of the **EmbeddedFontCollection** which can be accessed through **Embedded** property of the **FontInfo** class.

```
C#
GcWordDocument doc = new GcWordDocument();
// Load a DOCX that contains embedded font:
doc.Load(@"..\..\..\EmbedFonts.docx");

// Enumerate the embedded fonts:
foreach (var fi in doc.Fonts)
{
    // Remove all embedded fonts:
    fi.Embedded.Clear();
}
//Save Document
doc.Save("FindAndRemoveEmbeddedFont.docx");
```

Use Fonts with Different Culture Languages

DsWord provide Settings class to support **Theme Fonts** for different language cultures to render documents using multi-language Font scripts. You can fetch these theme fonts by using properties such as **LocaleName**, **LocaleNameBi** and **LocaleNameFarEast**. For a particular locale setting, DsWord finds the corresponding font script in **Supplemental** map from **MinorFont** and **MajorFont** theme font collection and marks them as 'in use'. If an appropriate font script is not found, DsWord defines font name from the MinorFont and MajorFont theme font collection. You can get the full list of font being used in the document by using **GetFontsInUse** method. All fonts from this list are candidates to be stored in the document but only fonts that are present in the **SaveOptions.AvailableFonts** are stored.

With this enhanced API support in place, DsWord also supports documents using **cloud fonts**. Following code shows how to embed cloud theme fonts in a Word document and also how to apply different language settings when Fonts from multiple languages are used

```
C#
GcWordDocument doc = new GcWordDocument();

// set default languages for theme fonts
var settings = doc.Settings;
```



```
settings.LocaleName = "en-US";
settings.LocaleNameBi = "ar-AE";
settings.LocaleNameFarEast = "ja-JP";

// set languages for default font
// so you don't need specify them for each run
var defFont = doc.Styles.DefaultFont;
defFont.LocaleName = "en_US";
defFont.LocaleNameBi = "ar-AE";
defFont.LocaleNameFarEast = "ja-JP";

// set minor theme fonts for default font
// so you don't need specify them for each run
defFont.ThemeAscii = ThemeFontType.MinorHighAnsi;
defFont.ThemeBi = ThemeFontType.MinorBidi;
defFont.ThemeFarEast = ThemeFontType.MinorEastAsia;

// add a heading style that uses major theme fonts
Style hstyle = doc.Styles.Add("My Heading", StyleType.Paragraph);
hstyle.Font.Size = 20;
hstyle.Font.ThemeAscii = ThemeFontType.MajorHighAnsi;
hstyle.Font.ThemeBi = ThemeFontType.MajorBidi;
hstyle.Font.ThemeFarEast = ThemeFontType.MajorEastAsia;
hstyle.Font.ThemeOther = ThemeFontType.MajorHighAnsi;

// modern Microsoft Word documents uses cloud fonts:
// https://support.microsoft.com/en-us/office/cloud-fonts-in-office-f7b009fe-037f-45ed-a556-b5fe6ede6adb
// if Office 365 is installed on you computer, make your document readable for every
one.

// specify minor theme fonts
var minorFont = doc.Theme.FontScheme.MinorFont;
minorFont.Latin.Name = "Calibri";
// you can specify fonts for any font script
minorFont.Supplemental["Arab"] = "Aldhabi";
minorFont.Supplemental["Jpan"] = "Yu Mincho";

// specify major theme fonts
var majorFont = doc.Theme.FontScheme.MajorFont;
majorFont.Latin.Name = "Calibri Light";
majorFont.Supplemental["Arab"] = "Traditional Arabic";
majorFont.Supplemental["Jpan"] = "Yu Gothic";

// add English text that will use minor and major Latin theme fonts automatically
var pars = doc.Body.Paragraphs;
pars.Add("Minor font");
pars.Add("Major font", hstyle);

// add Japanese text that will use minor and major East Asian theme fonts
automatically
```

```
pars.Add("マイナーフォント");
pars.Add("メジャーフォント", hstyle);

// add Arabic text that will use minor and major Complex Scrip theme fonts
Paragraph p = pars.Add();
Run run = p.GetRange().Runs.Add("خط ثانوي");
// need to specify that the run is complex script
run.Font.RightToLeft = true;
run.Font.HintType = FontHintType.ComplexScript;
p = pars.Add(hstyle);
run = p.GetRange().Runs.Add("الخط الرئيسي");
run.Font.RightToLeft = true;
run.Font.HintType = FontHintType.ComplexScript;

// replace existing fonts and embed all fonts in use to the document
var saveOpts = doc.Settings.SaveOptions;
saveOpts.SaveFontInfos = SaveFontInfos.SaveAll;
saveOpts.FontEmbedMode = FontEmbedMode.All;
var fontsInUse = saveOpts.GetFontsInUse();

// check whether all fonts that are used in the document are available to embed
foreach (string name in fontsInUse.Keys)
{
    if (saveOpts.AvailableFonts.FindFamilyName(name) == null)
    {
        // add cloud fonts to available font list
        var fonts = new GrapeCity.Documents.Text.FontCollection();
        string cloudFonts =
@"c:\Users\...\AppData\Local\Microsoft\FontCache\4\CloudFonts";
        // Make sure fonts you assign are downloaded, if not just use them in
        Microsoft Word, so they will be loaded automatically
        foreach (string fontFolder in Directory.EnumerateDirectories(cloudFonts))
            fonts.RegisterDirectory(fontFolder);
        saveOpts.AvailableFonts = fonts;
        break;
    }
}

// now this document will be displayed correctly on every machine
doc.Save(@"cloud-theme-fonts.docx");
```

Header and Footer

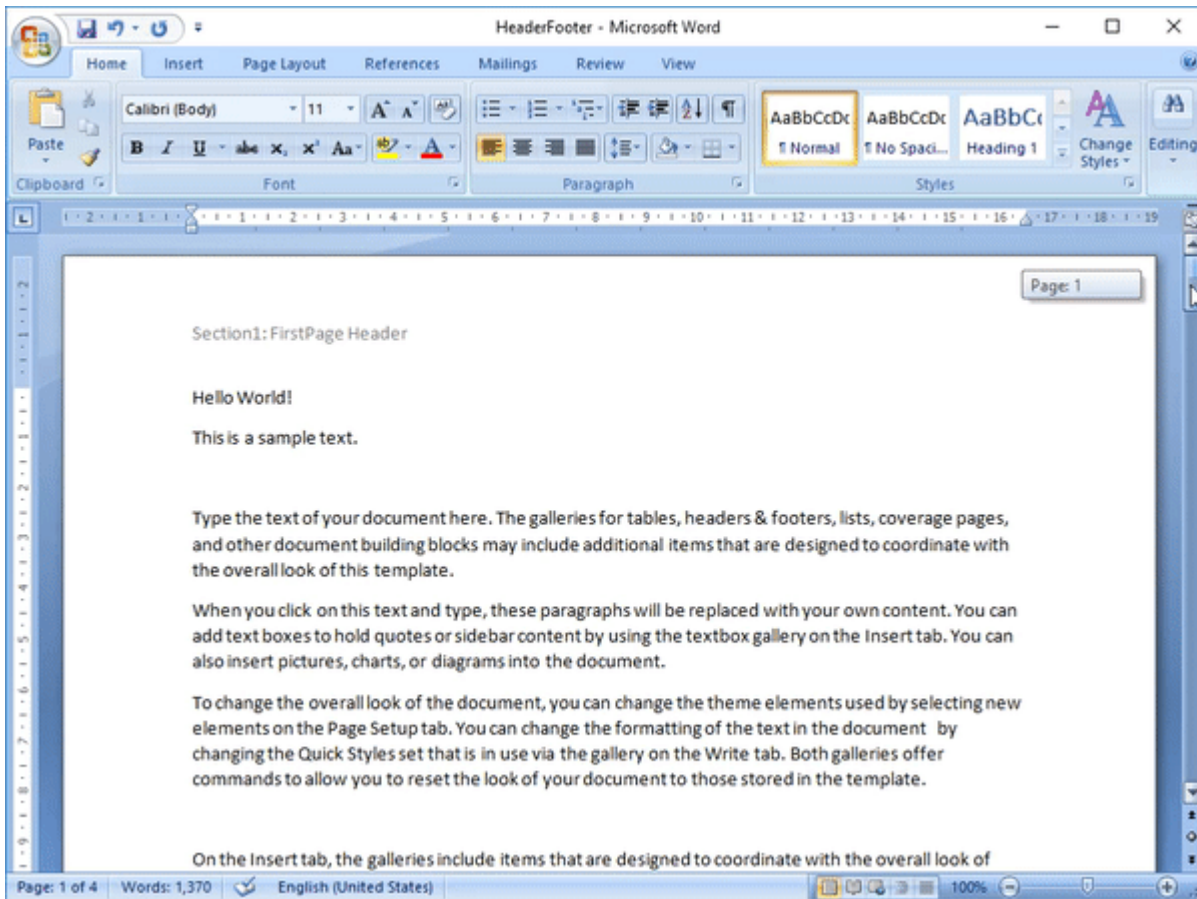
Headers and footers are generally used to display document name, date, page number, etc. In DsWord, headers and footers of a section are represented by the [HeaderFooter](#) class which provides access to the content and settings of header and footer. These headers and footers have their own body which represents the header and footer content. The HeaderFooter is not a part of the document body and it does not get derived from the ContentObject class, hence it is not a content object. The header and footer of a particular section can be accessed using [Headers](#) and [Footers](#) properties of the [Section](#) class which are of type [HeaderFooterCollection](#) class.

In addition, DsWord provides three different types of headers and footers through [HeaderFooterType](#) enumeration, which are listed below:

- **Primary** - The header or footer that appears on the odd pages.
- **EvenPage** - The header or footer that appears on the even pages.
- **FirstPage** - The header that appears only on the first page. When the first page header is not set, the primary header is displayed on the first page.

However, rendering of headers and footers in a document depends on the following properties::

- **LinkToPrevious** - This property is provided by the HeaderFooter class. It accepts boolean value to determine whether the header/footer is linked to the previous section. When the value of this property is set to true for a section, then the section displays the same header and footer as the previous one.
- **DifferentFirstPageHeaderFooter** - This property is provided by the PageSetup class. It accepts boolean value to determine whether the section should have a different header and footer for its first page or not.
- **OddAndEvenPagesHeaderFooter** - This property is provided by the PageSetup class and accepts boolean value. When set to true, Primary headers/footers are displayed for odd-numbered pages and EvenPage headers/footers are displayed for even-numbered pages. When set to false, the Primary type header/footer is displayed on all the pages of a section. Note that changing the value of this property affects all the section in a document.



Add Header and Footer

To add a header and footer to a section:

1. Access a section where you want to add a header and footer. For example, access first section of the document.
2. Use the **Headers** and **Footers** properties of the **Section** class to access the header and footer collection of the section.
3. Add header and footer, by using the [Add](#) method, on different pages, for example, first page, even page, and odd page.

C#

```
var section = doc.Body.Sections.First;

//Set different header and footer for the first page of the section
section.PageSetup.DifferentFirstPageHeaderFooter = true;

//Insert first page header and footer
section.Headers[HeaderFooterType.FirstPage].Body.Paragraphs.Add("Section1:
FirstPage Header");
section.Footers[HeaderFooterType.FirstPage].Body.Paragraphs.Add("Section1:
FirstPage Footer");

//Insert header and footer for odd pages
section.Headers[HeaderFooterType.Primary].Body.Paragraphs.Add("Section1: Odd
Page Header");
section.Footers[HeaderFooterType.Primary].Body.Paragraphs.Add("Section1: Odd
```

```
Page Footer");

//Insert header and footer for even pages
section.Headers[HeaderFooterType.EvenPages].Body.Paragraphs.Add("Section1: Even
Page Header");
section.Footers[HeaderFooterType.EvenPages].Body.Paragraphs.Add("Section1: Even
Page Footer");

//Add a paragraph to the section
for (var p = 1; p <= 50; p++)
{
    section.GetRange().Paragraphs.Add("Section1: Test Paragraph" +
p.ToString());
}

//Add second section
var section2 = doc.Body.Sections.Add();
section2.PageSetup.SectionStart = SectionStart.NewPage;
section2.PageSetup.OddAndEvenPagesHeaderFooter = true;

//Link the header and footer of pages with the previous section
section2.Headers[HeaderFooterType.Primary].LinkToPrevious = false;
section2.Footers[HeaderFooterType.Primary].LinkToPrevious = false;
section2.Headers[HeaderFooterType.EvenPages].LinkToPrevious = false;
section2.Footers[HeaderFooterType.EvenPages].LinkToPrevious = false;

//Insert header and footer for odd and even pages of the second section
section2.Headers[HeaderFooterType.Primary].Body.Paragraphs.Add("Section2: Odd
Page Header");
section2.Footers[HeaderFooterType.Primary].Body.Paragraphs.Add("Section2: Odd
Page Footer");
section2.Headers[HeaderFooterType.EvenPages].Body.Paragraphs.Add("Section2: Even
Page Header");
section2.Footers[HeaderFooterType.EvenPages].Body.Paragraphs.Add("Section2: Even
Page Footer");

//Add a paragraph to the second section
for (var p = 1; p <= 75; p++)
{
    section2.GetRange().Paragraphs.Add("Section2: Test Paragraph" +
p.ToString());
}

//Save the document
doc.Save("HeaderFooter.docx");
```

[Back to Top](#)

Modify Header and Footer

To modify a header and footer in a section:

1. Access the section whose header and footer needs to be modified. For example, access first section of the document
2. Access the header and footer collection using **Headers** and **Footers** properties of the **Section** class.
3. Modify text of the header and footer added to the first page using **Value** property of the **Text** class.

```
C#
doc.Load("HeaderFooter.docx");

//Access the first section
var section = doc.Body.Sections.First;

//Modify the header and footer on the first page
section.Headers[HeaderFooterType.FirstPage].Body.Texts.First.Value =
    "Modified first page header";
section.Footers[HeaderFooterType.FirstPage].Body.Texts.First.Value =
    "Modified first page footer";

//Save the document
doc.Save("ModifiedHeaderFooter.docx");
```

[Back to Top](#)

Delete Header and Footer

To delete the content of a header and/or footer from a section, you can use [Delete](#) method of the [ContentObject](#) class. Alternatively, you can clear content from the header and footer using [Clear](#) method of the [Body](#) class.

```
C#
doc.Load("HeaderFooter.docx");

var section = doc.Body.Sections.First;
//Clear the primary header and footer content in the first section
section.Headers[HeaderFooterType.Primary].Body.Clear();
section.Footers[HeaderFooterType.Primary].Body.Clear();

//Delete the primary header's and footer's first paragraph in the second section
var section2 = doc.Body.Sections[1];
section2.Headers[HeaderFooterType.Primary].Body.Paragraphs.First.Delete();
section2.Footers[HeaderFooterType.Primary].Body.Paragraphs.First.Delete();

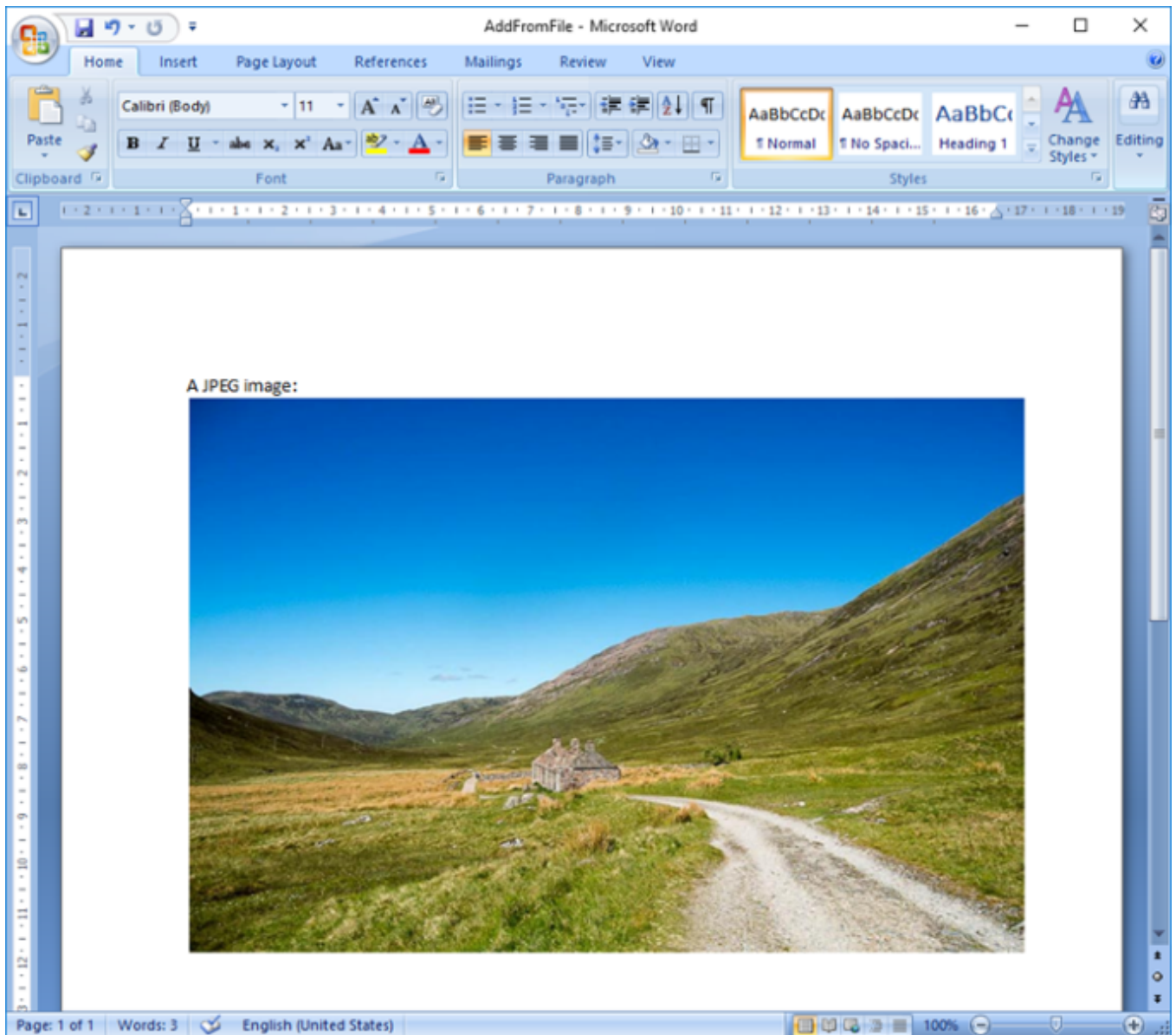
//Save the document
doc.Save("DeletedHeaderFooter.docx");
```

[Back to Top](#)

For more information on how to implement header and footer in a Word document using DsWord, see [DsWord sample browser](#).

Images

Images are generally used to illustrate important information in your document and highlight points raised in the text. In DsWord, an image or a picture element is represented by the [Picture](#) class which allows you to access all the image properties. DsWord allows you to add an image on a word document using [Add](#) method of the [PictureCollection](#) class that represents a collection of picture elements.



Add Image from File

To add an image in a document from file:

1. Create a byte array from the image using the [ReadAllBytes](#) method.
2. Access the picture collection using [Pictures](#) property of the [RangeBase](#) class.
3. Add image to the picture collection using **Add** method of the **PictureCollection** class, which accepts image as a byte array. For example, insert an image in the first paragraph.
4. Set properties of the image. For example, set the height and width of the image using [Height.Value](#) and [Width.Value](#) properties respectively.

```
C#
GcWordDocument doc = new GcWordDocument();

// Load picture data:
var picBytes = File.ReadAllBytes(Path.Combine("Resources", "Images",
"road.jpg"));

//Add a paragraph
var pars = doc.Body.Sections.First.GetRange().Paragraphs;
pars.Add("A JPEG image:");

// Add picture, specifying its mime type:
var pic = pars.First.GetRange().Runs.First.GetRange().Pictures.Add(picBytes,
"image/jpeg");

// Set picture size
pic.Size.Width.Value = 500;
pic.Size.Height.Value = 400;

//Save the document
doc.Save("AddFromFile.docx");
```

Back to Top

To add an image in a document with same size as the input image, you can use **System.Drawing.Image** namespace:

1. Load an image in a document and add it to the picture collection using **Add** method of the **PictureCollection** class.
2. Verify the size of original and added image by retrieving the value of **Height** and **Width** properties.

```
C#
using DrawingImage = System.Drawing.Image;
GcWordDocument doc = new GcWordDocument();
//Load image file
var fileName = Path.Combine(@"cat.png");

using (var image = DrawingImage.FromFile(fileName))
{
    var pic =
doc.Body.Paragraphs.Add("").GetRange().Runs.First.GetRange().Pictures.Add(image);
    //Output: Original Image width is 400 and Added image width is 400
    Console.WriteLine(String.Format(@"Original Image width is {0} and Added image
width is {1}", image.Size.Width, pic.Size.Width.Value));
    //Output: Original Image height is 300 and Added height is 300
    Console.WriteLine(String.Format(@"Original Image height is {0} and Added
height width is {1}", image.Size.Height, pic.Size.Height.Value));
    doc.Save("ImageAdded.docx");
}
```

Limitation

The emf, wmf and ico file formats are not supported while adding images using **System.Drawing.Image** namespace.

Back to Top

Similarly, you can add an image in a document with same size as the input image using [Image](#) class of [GrapeCity.Documents.Drawing](#) namespace as shown below:

```
C#  
  
using GcImage = GrapeCity.Documents.Drawing.Image;  
GcWordDocument doc = new GcWordDocument();  
//Load image file  
var fileName = Path.Combine(@"cat.png");  
  
using (var image = GcImage.FromFile(fileName))  
{  
    var pic2 =  
doc.Body.Paragraphs.Add("").GetRange().Runs.First.GetRange().Pictures.Add(image);  
    //Output: Original Image width is 400 and Added image width is 400  
    Console.WriteLine(String.Format(@"Original Image width is {0} and Added image  
width is {1}", image2.Width, pic2.Size.Width.Value));  
    //Output: Original Image height is 300 and Added height is 300  
    Console.WriteLine(String.Format(@"Original Image height is {0} and Added  
height width is {1}", image2.Height, pic2.Size.Height.Value));  
    doc.Save("ImageAdded.docx");  
}
```

Limitation

The emf and wmf file formats are not supported while adding images using [GrapeCity.Documents.Drawing](#) namespace.

Back to Top

Add Image from Stream

To add an image in a document from memory stream:

1. Access a memory stream loaded with image data.
2. Convert the memory stream to byte array using the `ToArray` method.
3. Add the image to picture collection using **Add** method of the `PictureCollection` class, which accepts image as a byte array.
4. Set some properties of the image. For example, set **Height.Value** and **Width.Value** properties.

```
C#  
  
//Load image to MemoryStream  
MemoryStream ms = new MemoryStream();  
System.Drawing.Image imageIn = System.Drawing.Image.FromFile(Path.Combine(  
    "Resources", "Images", "road.jpg"));  
imageIn.Save(ms, imageIn.RawFormat);  
  
//Convert MemoryStream to byte array  
var picBytes = ms.ToArray();  
  
//Add a paragraph  
var pars = doc.Body.Sections.First.GetRange().Paragraphs;  
pars.Add("A JPEG image:");
```

```
// Add picture, specifying its mime type:
var pic = pars.First.GetRange().Runs.First.GetRange().Pictures.Add(picBytes,
"image/jpeg");

// Set picture size
pic.Size.Width.Value = 500;
pic.Size.Height.Value = 400;

//Save the document
doc.Save("AddFromStream.docx");
```

[Back to Top](#)

Get Image

To get an image:

1. Access an image from the picture collection using Pictures property of the RangeBase class.
2. Get the image data using **ImageBytes** property of the **ImageData** class.
3. Add the fetched image from the document to another document using **Add** method of the **PictureCollection** class.

```
C#
doc.Load("AddFromFile.docx");

//Extract image from existing document
Picture oldpic =
doc.Body.Paragraphs.First.GetRange().Runs.First.GetRange().Pictures[0];
byte[] picBytes = oldpic.ImageData.ImageBytes;

//Add extracted image from old document to new document
GcWordDocument testDocument = new GcWordDocument();

var pars = testDocument.Body.Sections.First.GetRange().Paragraphs;
pars.Add("An old JPEG image:");

// Add picture, specifying its mime type:
var pic = pars.First.GetRange().Runs.First.GetRange().Pictures.Add(picBytes,
"image/jpeg");

// Set picture size
pic.Size.Width.Value = 500;
pic.Size.Height.Value = 400;

//Save the document
testDocument.Save("ExtractImage.docx");
```

[Back to Top](#)

Edit Image

To edit an image:

1. Access the image from picture collection using **Pictures** property of the **RangeBase** class.
2. Change the properties of the image. For example, change the rotation properties such as **Angle** and **VerticalFlip** properties of the **ShapeRotation** class.

```
C#
doc.Load("AddFromFile.docx");

//Edit image from existing document
Picture pic =
doc.Body.Paragraphs.First.GetRange().Runs.First.GetRange().Pictures[0];

//Set the rotation properties
pic.Rotation.Angle = 45;
pic.Rotation.VerticalFlip = true;

//Save the document
doc.Save("EditIma" +
        "ge.docx");
```

[Back to Top](#)

Delete Image

To delete an image from a document, access the image from the picture collection using Pictures property of the RangeBase class and delete it using the **Delete** method.

```
C#
doc.Load("AddFromFile.docx");

//Delete image from existing document
Picture pic = doc.Body.Paragraphs.First.GetRange().Runs.First.GetRange().Pictures[0];
pic.Delete();

//Save the document
doc.Save("DeleteImage.docx");
```

[Back to Top](#)

For more information on how to work with images in a Word document using DsWord, see [DsWord sample browser](#).

Shapes

In DsWord, all the shape types are represented by the [ShapeBase](#) class. This class allows you to work with basic shape properties like alternative text, positioning, rotation, title, wrapping text around shape etc. The [Shape](#) class represent a single shape element and inherits the **ShapeBase** class.

In this section, we will discuss about the following shape:

- [Textbox](#)
- [Canvas Shape](#)
- [Group Shape](#)
- [Ink Shape](#)
- [Shape](#)
- [Shape Format](#)
- [Picture](#)
- [Picture Format](#)
- [Online Video](#)
- [Presets and Themed Styles](#)
- [Shape Styles](#)

Textbox

While adding text in a document, certain kind of formatting and placement restrictions are faced. These restrictions can be overcome by using a textbox which provides the flexibility to add text anywhere in the document and add certain styles and formatting to it. The textbox is also very helpful for creating a blockquote or a sidebar.

DsWord allows you to add textbox in a document which is a combination of text rendered on a shape element. A shape can be added by using the [Shape](#) class and the text can be rendered over it by using the [TextFrame](#) class. Additionally, DsWord also provides [LinkedTextFrame](#) class which provides additional space for textual content if it does not fit into the original text frame. The **LinkedTextFrame** is only visible when it accommodates any additional content. Each shape can have no more than one [TextFrame](#) or [LinkedTextFrame](#).

You can also define the formatting of text frame like its orientation, word wrap, margin, auto-fit shape to text etc. using the [TextFrameFormat](#) class.



Add Textbox

To add textbox in a document:

1. Create a new Word document by instantiating the **GcWordDocument** class.
2. Add a paragraph by using the **Add** method of **ParagraphCollection** class.
3. Add a run to the paragraph and shape to the run by using the **Add** method of [RunCollection](#) and [ShapeCollection](#) class respectively.
4. Add a text frame to the shape by using the [AddTextFrame](#) method of the **Shape** class.
5. Add a linked text frame which will accommodate the extra text, if any, by using the [AddLinkedTextFrame](#) method of **Shape** class.

C#

```
GcWordDocument doc = new GcWordDocument();
Paragraph para = doc.Body.Paragraphs.Add();
Run run = para.GetRange().Runs.Add();

//Add shape
Shape shape = run.GetRange().Shapes.Add(100, 100);

//Add a text frame along with text on shape
TextFrame tf = shape.AddTextFrame("Document Solutions for Word library is a part
```

```
of Document Solutions " +
    "that aims to be a complete solution to program and work with Word
documents");

//Add linked text frame to hold any extra text
LinkedTextFrame ltf =
doc.Body.Paragraphs.Add().GetRange().Runs.Add().GetRange().Shapes.Add(100,
100).AddLinkedTextFrame(ltf);

doc.Save("Textbox.docx");
```

[Back to Top](#)

Modify Textbox

To modify a textbox:

1. Create a new Word document by instantiating the **GcWordDocument** class.
2. Load the document containing textbox using **Load** method of **GcWordDocument** class.
3. Access the first text frame in the document and set its formatting properties using the **Format** property of **TextFrame** class.

```
C#
GcWordDocument doc = new GcWordDocument();
doc.Load("Textbox.docx");

//Set formatting properties for Textbox
doc.Body.TextFrames[1].Format.Orientation = TextOrientation.Vertical;
doc.Body.TextFrames[1].Format.WordWrap = true;
doc.Body.TextFrames[1].Format.VerticalAnchor = TextVerticalAnchor.Bottom;

doc.Save("Textbox.docx");
```

[Back to Top](#)

Delete Textbox

To delete a textbox:


1. Load the document containing textbox using **Load** method of **GcWordDocument** class.
2. Delete the textbox shape by using the **Delete** method of **ContentObject** class.

```
C#
GcWordDocument doc = new GcWordDocument();
doc.Load("Textbox.docx");

//Delete the textbox shape
doc.Body.Paragraphs[1].GetRange().Shapes.First.Delete();

doc.Save("Textbox1.docx");
```

[Back to Top](#)

 **Note:** You can also export a Word document containing textbox to PDF and image formats. To know more, refer [Export](#) topic.

Canvas Shape

A canvas shape acts as a container for multiple drawing objects and can be used to organize them all at once.

DsWord allows you to draw a canvas shape and add multiple drawing objects to it. A canvas shape can be added by using [Add](#) and [Insert](#) methods of [CanvasShapeCollection](#) class. You can also access the previous or next canvas shapes and apply fill format or line format properties on them by using various methods provided by [CanvasShape](#) class.

Add Canvas Shape

To add canvas shape in a document:

1. Create a new Word document by instantiating **GcWordDocument** class.
2. Add a run to a newly added paragraph by using **Add** method of **RunCollection** class.
3. Add a canvas shape by using **Add** method of **CanvasShapeCollection** class.
4. Add two shapes inside canvas shape by using **Add** method of **ShapeCollection** class.
5. Move the horizontal and vertical position of second shape by using **Offset** property of **ShapeHorizontalPosition** class so that it does not overlap the first shape.

```
C#  
  
var doc = new GcWordDocument();  
  
//Add Run element  
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();  
  
//Create canvas shape with initial width 500 and height 500  
var canvas_shape = run.GetRange().CanvasShapes.Add(500, 500);  
  
//Create two shapes inside canvas shape  
var firstShape = canvas_shape.GetRange().Shapes.Add(202, 250, "First shape");  
var secondshape = canvas_shape.GetRange().Shapes.Add(300, 400, "Second shape");  
  
//Move second shape 100 points right and 100 points lower  
secondshape.Position.Horizontal.Offset = 100;  
secondshape.Position.Vertical.Offset = 100;  
  
//Save document  
doc.Save("CanvasShape.docx");
```

[Back to Top](#)

Delete Canvas Shape

To delete canvas shape in a document:

1. Load the document containing canvas shape using **Load** method of **GcWordDocument** class.
2. Delete the canvas shape by using **Delete** method of **ContentObject** class.

```
C#  
  
//Load document  
doc.Load("CanvasShape.docx");  
  
//Access first canvas
```



```
CanvasShape cs = doc.Body.CanvasShapes.First;  
//Delete canvas  
cs.Delete();  
  
doc.Save("CanvasDelete.docx");
```

Back to Top


Limitations

- Unsupported Properties: NonVisualGroupDrawingShapePropsExtensionList, GroupShapeLocks, OfficeArtExtensionList, Scene3DType, ShapePropertiesExtensionList
- Unsupported Effects and derived classes: EffectsDAG, EffectsList, WholeFormattingEffectList, WholeFormattingEffectDag

Group Shape

Group shape is a shape element which can be used to add multiple shapes which need to be grouped together. It provides unified properties for child shapes and can be regarded as a shape container.

DsWord allows you to add a group shape and add multiple shapes to it. A group shape can be added by using [Add](#) and [Insert](#) methods of [GroupShapeCollection](#) class. You can also access the previous or next group shapes, apply fill format or group fill properties and ungroup the grouped shapes by using methods provided by [GroupShape](#) class.

 **Note:** Deleting a group shape deletes all the shapes inside it.

Add Group Shape

To add group shape in a document:

1. Create a new Word document by instantiating **GcWordDocument** class.
2. Add a run to a newly added paragraph by using **Add** method of **RunCollection** class.
3. Add a group shape by using **Add** method of **GroupShapeCollection** class.
4. Add two shapes inside group shape by using **Add** method of **ShapeCollection** class.
5. Move the horizontal and vertical position of second shape by using **Offset** property of **ShapeHorizontalPosition** class so that it does not overlap the first shape.

```
C#  
  
var doc = new GcWordDocument();  
  
//Add run element  
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();  
  
//Create group shape with initial width 500 and height 500  
var group_shape = run.GetRange().GroupShapes.Add(500, 500);  
  
//Create two shapes inside group shape  
var firstShape = group_shape.GetRange().Shapes.Add(202, 250, "First shape");  
var secondshape = group_shape.GetRange().Shapes.Add(300, 400, "Second shape");  
  
//Move second shape 100 points right and 100 points lower  
secondshape.Position.Horizontal.Offset = 100;  
secondshape.Position.Vertical.Offset = 100;  
  
//Save document  
doc.Save("GroupShape.docx");
```

[Back to Top](#)

Delete Group Shape

To delete group shape in a document:

1. Load the document containing group shape using **Load** method of **GcWordDocument** class.
2. Delete the group shape by using **Delete** method of **ContentObject** class.

```
C#  
  
//Load document
```

```
doc.Load("GroupShape.docx");

//Access first group shape
GroupShape gs = doc.Body.GroupShapes.First;
//Delete group shape
gs.Delete();

doc.Save("GroupDelete.docx");
```

Back to Top

Limitations

- Unsupported Properties: NonVisualGroupDrawingShapePropsExtensionList, GroupShapeLocks, OfficeArtExtensionList, Scene3DType, ShapePropertiesExtensionList
- Unsupported Effects and derived classes: EffectsDAG and EffectsList

Ink Shape

Ink shapes are user-drawn shapes which can include lines and curves.

DsWord supports ink shapes which can be added by using [Add](#) and [Insert](#) methods of [InkShapeCollection](#) class. You can also access the previous or next ink shapes and get the xml document containing ink shape's content by using various methods provided by [InkShape](#) class.

In DsWord, you can create a new ink shape by providing an xml document which describes ink drawing or load a document with predefined ink shapes.

Modify Ink Shape

To modify an existing ink shape in a document:

1. Load the document containing ink shape using **Load** method of **GcWordDocument** class.
2. Access the ink shape by using [InkShapes](#) property of **RangeBase** class.
3. Modify ink shape by setting various properties of **ShapeBase** class like AlternativeText, Rotation, Title, Height and Width etc

```
C#
var doc = new GcWordDocument();

//Load doc containing ink shape
doc.Load("ink.docx");

if (doc.Body.InkShapes.Count == 0)
    return;

//Access first ink shape
var ink = doc.Body.InkShapes.First;

//Modify Ink shape
ink.BlackWhiteMode = BlackWhiteMode.LightGray;
ink.AlternativeText = "xyz";
ink.Rotation.HorizontalFlip = true;
ink.Title = "ink shape title";
ink.Size.Height.Value = 300;
ink.Size.Width.Value = 100;

//Save document
doc.Save("InkShapeModified.docx");
```

[Back to Top](#)

Shape

DsWord allows you to add predefined geometrical shapes which are visual figures with filling and outline. A few examples of shapes include rectangle, arrow, ellipse, hexagon etc. A shape can be added in a document by using [Add](#) and [Insert](#) methods of [ShapeCollection](#) class. You can also access the previous or next shapes, apply fill format, line format, group fill, shape style etc. by using methods provided by [Shape](#) class.


DsWord supports 188 geometrical shapes which can be viewed by downloading the [Word document](#).

To add shape in a document:

1. Create a new Word document by instantiating **GcWordDocument** class.
2. Add a run to a newly added paragraph by using **Add** method of **RunCollection** class.
3. Add an arc shape by using **Add** method of **ShapeCollection** class and pass **GeometryType.Arc** as its parameter.
4. Set its fill formatting property and theme color by using [FillType](#) and [ThemeColorId](#) enumeration.

```
C#  
  
var doc = new GcWordDocument();  
  
//Create a Run element  
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();  
  
//Add an arc shape  
var shape = run.GetRange().Shapes.Add(200, 300, GeometryType.Arc);  
shape.Line.Fill.Type = FillType.Solid;  
shape.Line.Fill.SolidFill.ThemeColor = ThemeColorId.Accent4;  
  
doc.Save("shape.docx");
```

Back to Top

 **Note:** You can also export Word document containing shapes to PDF and image formats. To know more, refer [Export](#) topic.

Limitations

Effects and derived classes (EffectsDAG, EffectsList) are not supported.

Shape Format

DsWord allows you to enhance the look of shapes by customizing shape formats. As a shape represents a visual figure containing fill and outline, you can customize its fill format as well as line format. You can also add a shadow to the shape and set the shadow format.

Fill Format

The fill format of a shape can be customized by using [FillFormat](#) class which provides [PatternFill](#), [SolidFill](#), [ImageFill](#), [GradientFill](#) properties that represent relevant fill types. The [FillType](#) enumeration can be used to define the active fill type.

To add gradient type fill format to a shape:

1. Add a rectangle shape to a Word document by using **Add** method of **ShapeCollection** class and pass **GeometryType.Rectangle** as its parameter.
2. Set **FillType** enumeration to **Gradient** to set gradient fill format.
3. Add gradient stops at different positions in the shape by using **Add** method of [GradientStopList](#) class.

```
C#  
  
var doc = new GcWordDocument();  
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();  
var shape = run.GetRange().Shapes.Add(300, 300, GeometryType.Rectangle);  
  
//Set fill format to gradient  
shape.Fill.Type = FillType.Gradient;  
  
//create additional gradient stops  
//position range is 0..100.  
shape.Fill.GradientFill.Stops.Add(ThemeColorId.Light2, 10f);  
shape.Fill.GradientFill.Stops.Add(ThemeColorId.Accent2, 70f);  
shape.Fill.GradientFill.Stops.Add(ThemeColorId.Light2, 87f);  
  
doc.Save("fillformat.docx");
```

Back to Top

Line Format

The line format of a shape defines appearance of a shape's line. It can be customized by using [LineFormat](#) class and can be set to solid, gradient, image, pattern fill types. You can also set various types of line formats like dash type, cap type, join type etc. by using properties of **LineFormat** class.

To add solid type line format to a shape:

1. Add a rectangle shape to a Word document by using **Add** method of **ShapeCollection** class and pass **GeometryType.Rectangle** as its parameter.
2. Set **FillType** enumeration to **Solid** to set solid line format.
3. Set **LineDashType** and **LineJoinType** enumerations to set dash type and join type of line.

```
C#  
  
var doc = new GcWordDocument();  
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();  
var shape = run.GetRange().Shapes.Add(300, 300, GeometryType.Rectangle);
```

```
//Set line format to solid
shape.Line.Fill.Type = FillType.Solid;
//Set line type to dotted
shape.Line.DashType = LineDashType.Dot;
//Set line join type to round
shape.Line.JoinType = LineJoinType.Round;

doc.Save("lineformat.docx");
```

[Back to Top](#)

Shadow Format

The shadow format adds additional depth to a shape and creates a three-dimensional effect. DsWord allows you to apply shadow format to shape using [ApplyBuiltInShadow](#) method of the [ShapeEffects](#) class that accepts [BuiltInShadowId](#) enumeration as its parameter. However, for applying custom shadow effect to a shape, you can use **OuterShadow**, **InnerShadow** or **ShadowPreset** type of the [ShapeEffects](#) class and set their properties.

The code below shows how to apply built-in shadow effect to a shape:

C#

```
// apply lower right prospective offset shadow effect to the shape
shape.Effects.ApplyBuiltInShadow(BuiltInShadowId.ProspectiveLowerRight);
```

The code below shows how to apply custom shadow effect to a shape:

C#

```
var canvas_shape = run.GetRange().CanvasShapes.Add(500, 500);
var shape2 = canvas_shape.GetRange().Shapes.Add(202, 250, "Shape");
shape2.Fill.SolidFill.RGB = Color.LemonChiffon;
shape2.Position.Horizontal.Offset = 4f;

var shape2InnerShadow = shape2.Effects.InnerShadow;
shape2InnerShadow.Blur = 4f;
shape2InnerShadow.Color.RGB = Color.Green;
shape2InnerShadow.Angle = 35;
shape2InnerShadow.Distance = 40f;
```

Blur Format

The blur effect makes the edges of shapes appear fuzzy or out of focus. DsWord allows you to apply blur effect to the shape by using **Blur** property of the [ShapeEffects](#) class. You can also apply this effect through [shape styles](#).



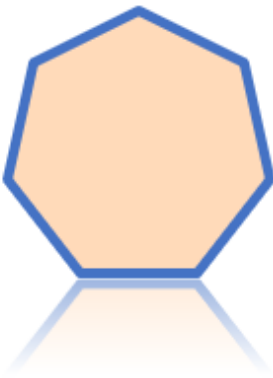
C#

```
// Shape Blur - direct:
Paragraph p = doc.Body.Paragraphs.Add();
Run run = p.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Star4);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.RGB = Color.Yellow;
shape.Line.Width = 4;
shape.Line.Fill.SolidFill.RGB = Color.Red;
// apply 7 point blur effect to the shape
shape.Effects.Blur.Radius = 7f;
p.GetRange().Runs.Add("Shape Blur - direct.", doc.Styles[BuiltInStyleId.Strong]);
```

To view this sample code in action, see [Blur Effect demo](#) sample.

Reflection Format

DsWord allows you to apply reflection format to shape using **ApplyBuiltInReflection** method of the [ShapeEffects](#) class that accepts [BuiltInReflectionId](#) enumeration as its parameter. You can also use [styles](#) to define and apply custom reflection effects to a shape.



The code below shows how to apply built-in reflection to a shape using the `ApplyBuiltInReflection` method:

C#

```
// Shape reflection - direct effects:
Paragraph p = doc.Body.Paragraphs.Add();
Run run = p.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Heptagon);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.RGB = Color.PeachPuff;
shape.Line.Width = 4;
shape.Line.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;
shape.Effects.ApplyBuiltInReflection(BuiltInReflectionId.TightTouching);
```

The code below shows how to apply custom reflection effect to a shape:

C#

```
Paragraph p = doc.Body.Paragraphs.Add();
```



```
Run run = p.GetRange().Runs.Add();
var canvas_shape = run.CanvasShapes.Add(500, 500);
var shape = canvas_shape.GetRange().Shapes.Add(202, 250, "Shape");
shape.Fill.SolidFill.RGB = Color.Red;
shape.Position.Horizontal.Offset = 4f;
var shapeCustomReflection = shape.Effects.Reflection;
shapeCustomReflection.Angle = 35;
shapeCustomReflection.Distance = 40f;
```

To view the code in action, see [Reflection Effect demo](#) sample.

Glow Format

DsWord allows you to apply glow effect to shape using **ApplyBuiltInGlow** method of the [ShapeEffects](#) class that accepts [BuiltInGlowId](#) enumeration as its parameter. You can also use styles to define and apply custom glow effects to a shape.



The code below shows how to apply built-in glow to a shape using the `ApplyBuiltInGlow` method:

```
C#
var shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Star5);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;
// apply 18 point accent 6 glow effect to the shape
shape.Effects.ApplyBuiltInGlow(BuiltInGlowId.Radius18Accent6);
p.GetRange().Runs.Add("Shape glow - direct.", doc.Styles[BuiltInStyleId.Strong]);
```

The code below shows how to apply custom glow effect to a shape:

```
C#
var canvas_shape = run.GetRange().CanvasShapes.Add(900, 900);
var shape = canvas_shape.GetRange().Shapes.Add(102, 102, "Shape");
shape.Fill.SolidFill.RGB = Color.LemonChiffon;
shape.Position.Horizontal.Offset = 9f;
var shapeGlow = shape.Effects.Glow;
shapeGlow.Radius = 30;
shapeGlow.Color.RGB = Color.Red;
```

To view the code in action, see [Glow Effect demo](#) sample.

SoftEdge Format

DsWord allows you to apply SoftEdge effect to the shape by using [SoftEdge](#) property of the [ShapeEffects](#) class. You can also apply this effect through [shape styles](#).



C#

```
// Shape Soft Edge - direct:
Paragraph p = doc.Body.Paragraphs.Add();
Run run = p.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Star7);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.RGB = Color.Yellow;
shape.Line.Width = 8;
shape.Line.Fill.SolidFill.RGB = Color.Red;

// apply 5 point soft edge effect to the shape
shape.Effects.SoftEdge.Radius = 5f;
p.GetRange().Runs.Add("Shape Soft Edge - direct.", doc.Styles[BuiltInStyleId.Strong]);
```

To view this sample code in action, see [SoftEdge Effect demo](#) sample.

FillOverlay Format

DsWord allows you to apply FillOverlay effect to the shape by using [FillOverlay](#) property of the **ShapeEffects** class. You can also apply this effect through [shape styles](#).



C#

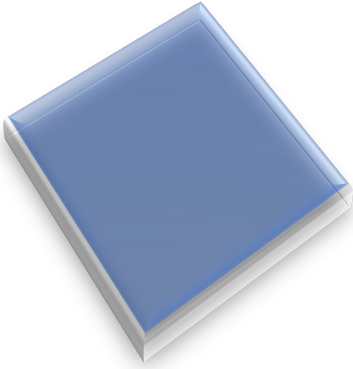
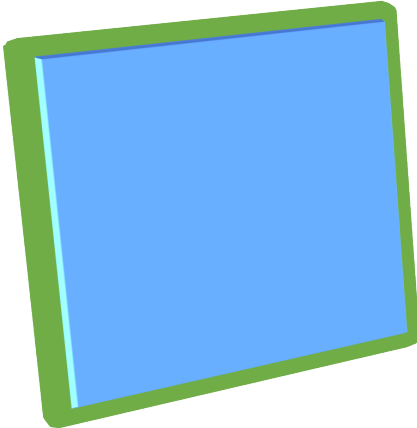
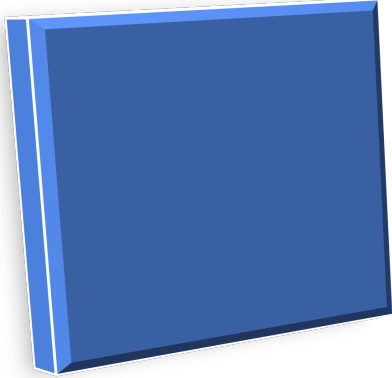
```
// Shape fill overlay - direct:
var p = doc.Body.Paragraphs.Add();
var run = p.GetRange().Runs.Add();
var shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Star8);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.RGB = Color.LightBlue;
// Apply solid fill overlay with another color to mix with the main fill:
FillOverlay overlay = shape.Effects.FillOverlay;
overlay.BlendMode = BlendMode.Multiply;
```

```
overlay.Fill.Type = FillType.Solid;
overlay.Fill.SolidFill.RGB = Color.Yellow;
p.GetRange().Runs.Add("Shape fill overlay - direct.",
doc.Styles[BuiltInStyleId.Strong]);
```

To view this sample code in action, see [FillOverlay Effect demo](#) sample.

3D Format

The 3D effects refer to the effects applied in three spatial dimensions: width, height, and depth. The 3D effects give the shape a unique and artistic look. For applying 3D effects, DsWord provides the [ThreeDFormat](#) and [ThreeDScene](#) classes and the [ApplyEffectsPreset](#) method. The ApplyEffectsPreset method allows a user to add preset 3D effects. However, the ThreeDFormat and ThreeDScene classes allow a user to add custom 3D effects.

Preset 3D Effect	Custom 3D Effect	3D Rotation
		

Refer to the following example code to add a preset 3D effect:

```
C#
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Add a shape to the document.
Paragraph paragraph = doc.Body.Paragraphs.Add();
Run run = paragraph.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add(200, 200, GeometryType.Rectangle);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;

// Apply a preset 3D shape effect.
shape.ApplyEffectsPreset(ShapeEffectsPreset.Preset10);

// Save the Word document.
doc.Save("Preset3DEffects.docx");
```

Refer to the following example code to add a custom 3D effect:

```
C#
```

```
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Add a shape to the document.
Paragraph paragraph = doc.Body.Paragraphs.Add();
Run run = paragraph.GetRange().Runs.Add();
var shape = run.GetRange().Shapes.Add(200, 200, GeometryType.Rectangle);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;

// Apply 3D format to the shape.
ThreeDFormat threeD = shape.Effects.ThreeDFormat;
threeD.TopBevel.ApplyPreset(BevelType.Convex);
threeD.Contour.Color.ThemeColor = ThemeColorId.Accent2;
threeD.Contour.Width = 3f;
threeD.Contour.Color.ThemeColor = ThemeColorId.Accent6;
threeD.Contour.Width = 12f;

// Apply 3D scene to the shape.
ThreeDScene scene = shape.Effects.ThreeDScene;
scene.Camera.Preset = CameraPreset.PerspectiveContrastingRightFacing;
scene.Lighting.Type = LightRigType.BrightRoom;
scene.Lighting.Rotation.Latitude = 90f;
scene.Lighting.Rotation.Revolution = 5f;

// Save the Word document.
doc.Save("3DEffects.docx");
```

Refer to the following example code to add a 3D rotation effect:

```
C#
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Add a shape to the document.
Paragraph paragraph = doc.Body.Paragraphs.Add();
Run run = paragraph.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add(200, 200, GeometryType.Rectangle);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;

shape.Line.Fill.Type = FillType.NoFill;

// Add outershadow to the shape.
OuterShadow shadow = shape.Effects.Shadow;
shadow.Alignment = RectangleAlignment.Center;
shadow.Angle = 87f;
shadow.Blur = 17.75f;
shadow.Distance = 4f;
shadow.Color.RGB = Color.FromArgb(255, 0, 0, 0);
shadow.Color.Transparency = 0.67f;
```

```
// Apply 3D format.
ThreeDFormat threeDFormat = shape.Effects.ThreeDFormat;
threeDFormat.Depth.Width = 20f;
threeDFormat.Contour.Width = 1.5f;
threeDFormat.Contour.Color.RGB = Color.FromArgb(255, 255, 255, 255);

// Add top bevel.
Bevel topbevel = threeDFormat.TopBevel;
topbevel.Type = BevelType.Angle;
topbevel.Width = 6.5f;
topbevel.Height = 3.5f;

// Add bottom bevel.
Bevel bottomBevel = threeDFormat.BottomBevel;
bottomBevel.Type = BevelType.Angle;
bottomBevel.Width = 6.5f;
bottomBevel.Height = 3.5f;

// Apply 3D scene.
Lighting lighting = shape.Effects.ThreeDScene.Lighting;
lighting.Type = LightRigType.Harsh;
lighting.Direction = LightRigDirection.Top;
lighting.Rotation.Latitude = 0f;
lighting.Rotation.Longitude = 0f;
lighting.Rotation.Revolution = 50f;

// Apply rotation to the shape.
Camera camera = shape.Effects.ThreeDScene.Camera;
camera.Preset = CameraPreset.PerspectiveFront;
camera.Perspective = 55f;
camera.Rotation.Latitude = 8.1f;
camera.Rotation.Longitude = 325.5f;
camera.Rotation.Revolution = 2.9f;

// Save the Word document.
doc.Save("Shape3DRotation.docx");
```

Limitation

DsWord does not support the export of 3D effects to PDF or images.

Picture

DsWord allows you to add pictures in documents and apply various settings on them. A picture can be added by using [Add](#) and [Insert](#) methods of [PictureCollection](#) class. You can also access the previous or next pictures, apply fill format, line format, shape style, set alternative text, title etc. by using methods provided by [Picture](#) class.

To add picture in a document:

1. Create a new Word document by instantiating **GcWordDocument** class.
2. Add a run to a newly added paragraph by using **Add** method of **RunCollection** class.
3. Add a picture by using **Add** method of **PictureCollection** class.
4. Set the position, width and outline color of picture by using various properties.

```
C#
var doc = new GcWordDocument();

//Create run element
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();

//Add a picture
var pic = run.GetRange().Pictures.Add();
pic.ImageData.SetImage(new Uri("Resources/picture.png"), "image/png");

//Shift picture position 100 points lower
pic.Position.Vertical.RelativeTo = ShapeVerticalRelativePosition.Page;
pic.Position.Vertical.Offset = 100;

//Set outline width
pic.Line.Width = 2;
//Increase "effective size" of shape by same value as outline width
pic.Size.EffectExtent.AllEdges = 2;
//set outline color
pic.Line.Fill.Type = FillType.Solid;
pic.Line.Fill.SolidFill.RGB = System.Drawing.Color.Red;

doc.Save("Picture.docx");
```

Back to Top

Limitations

Effects and derived classes (EffectsDAG, EffectsList) are not supported.

Picture Format

DsWord allows you to enhance the look of a picture by customizing the picture formats using [EmbeddedImageData](#) and [ImageData](#) classes. ImageData class enables you to apply effects, adjust brightness, contrast, sharpness, etc., and change the color to a preset color or a custom color.

Apply Picture Effects

DsWord allows you to enhance your picture by adding picture effects, such as blur, grayscale, tint, duo tone, fill overlay, etc using [Effects](#) property of ImageData class.

Refer to the following example code to add grayscale effect to the picture:

C#

```
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Read PNG image from a file.
var bytes = File.ReadAllBytes(@"image.png");

// Add PNG picture into the document.
var picture = doc.Body.AddParagraph().AddRun().AddPicture(bytes, @"image/png", 300f,
300f);

// Add grayscale to the picture.
picture.ImageData.Effects.AddGrayscale();

// Save the Word document.
doc.Save(@"picture-effects.docx");
```

Original



Grayscale Effect



Adjust Picture Settings

DsWord allows you to enhance your picture by adjusting the picture settings, such as brightness, contrast, temperature, saturation, and sharpness, using `ImageData` property of `Picture` class.

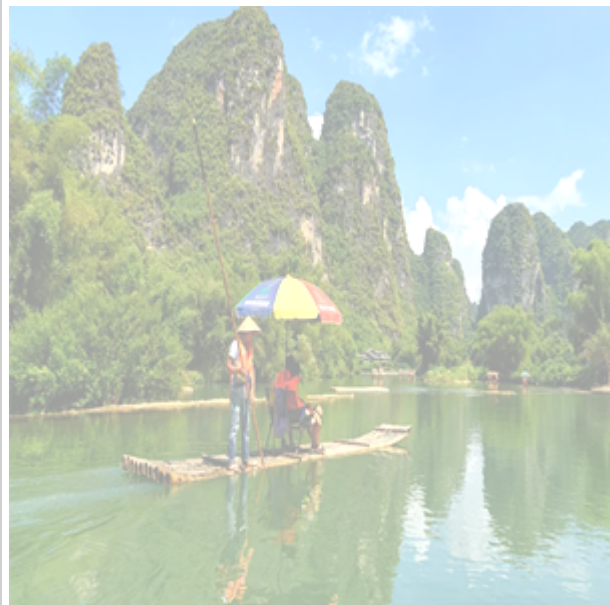
Refer to the following example code to adjust the brightness and contrast of a picture:

```
C#  
  
// Initialize GcWordDocument.  
GcWordDocument doc = new GcWordDocument();  
  
// Read PNG image from a file.  
var bytes = File.ReadAllBytes(@"image.png");  
  
// Add PNG picture into the document.  
var picture = doc.Body.AddParagraph().AddRun().AddPicture(bytes, @"image/png", 300f,  
300f);  
  
// Increase the picture brightness.  
picture.ImageData.Brightness = 0.5f;  
  
// Decrease the picture contrast.  
picture.ImageData.Contrast = -0.5f;  
  
// Save the Word document.  
doc.Save(@"picture-settings.docx");
```

Original



Brightness and Contrast Adjusted



Change Picture Color

DsWord allows you to enhance your picture by adjusting the picture color to a preset color or desired colors using `Recolor` method of `ImageData` class and `AddColorChange` method of `ImageEffectList` class.

Refer to the following example code to adjust the picture color to a preset color:

C#

```
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Read PNG image from a file.
var bytes = File.ReadAllBytes(@"image.png");

// Add PNG picture into the document.
var picture = doc.Body.AddParagraph().AddRun().AddPicture(bytes, @"image/png", 300f,
300f);

// Recolor the picture to grayscale.
picture.ImageData.Recolor(RecolorType.Accent1Light);

// Save the Word document.
doc.Save(@"recolor-preset.docx");
```

Original**Preset Color: Accent1Light**

Refer to the following example code to adjust the picture color to the desired color:

C#

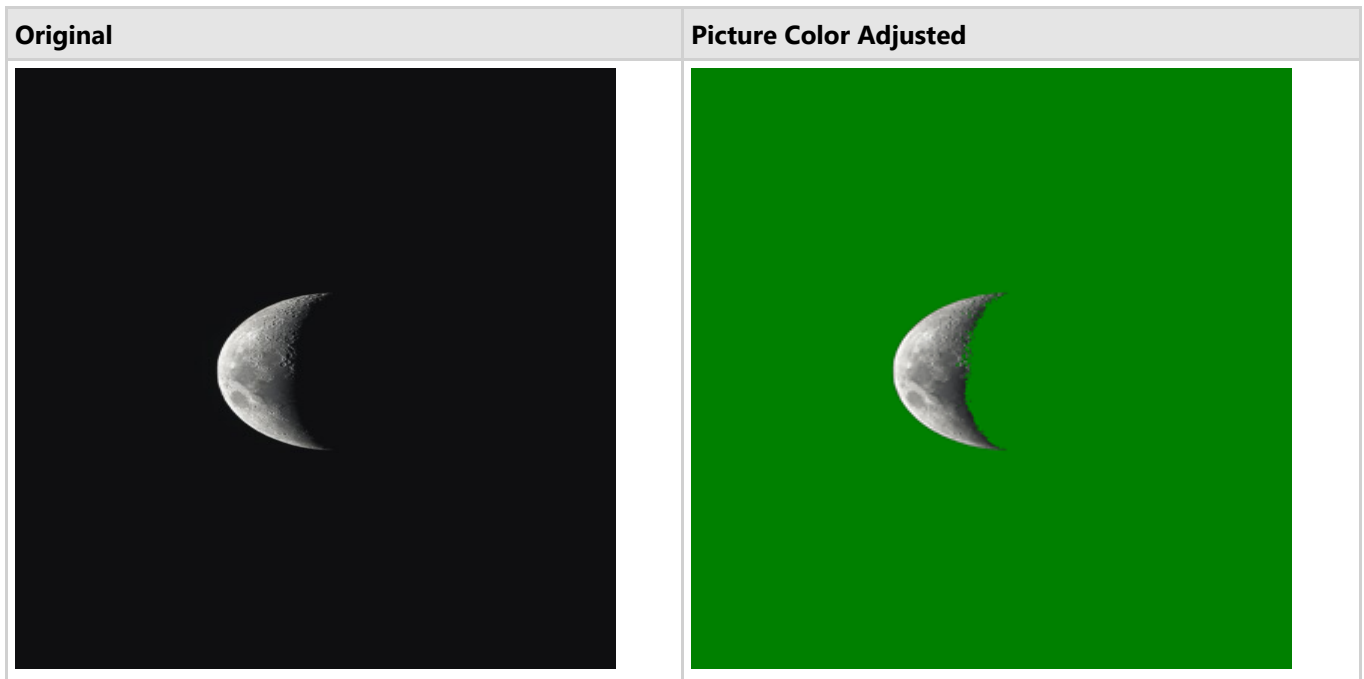
```
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Read JPEG image from a file.
var bytes = File.ReadAllBytes(@"image.png");

// Add JPEG picture into the document.
var picture = doc.Body.AddParagraph().AddRun().AddPicture(bytes, @"image/png", 300f,
300f);

// Change blue to green color in the picture.
picture.ImageData.Effects.AddColorChange(new UserColor(Color.FromArgb(15,17,19)), new
```

```
UserColor(Color.Green));  
  
// Save the Word document.  
doc.Save(@"recolor-desired-color.docx");
```

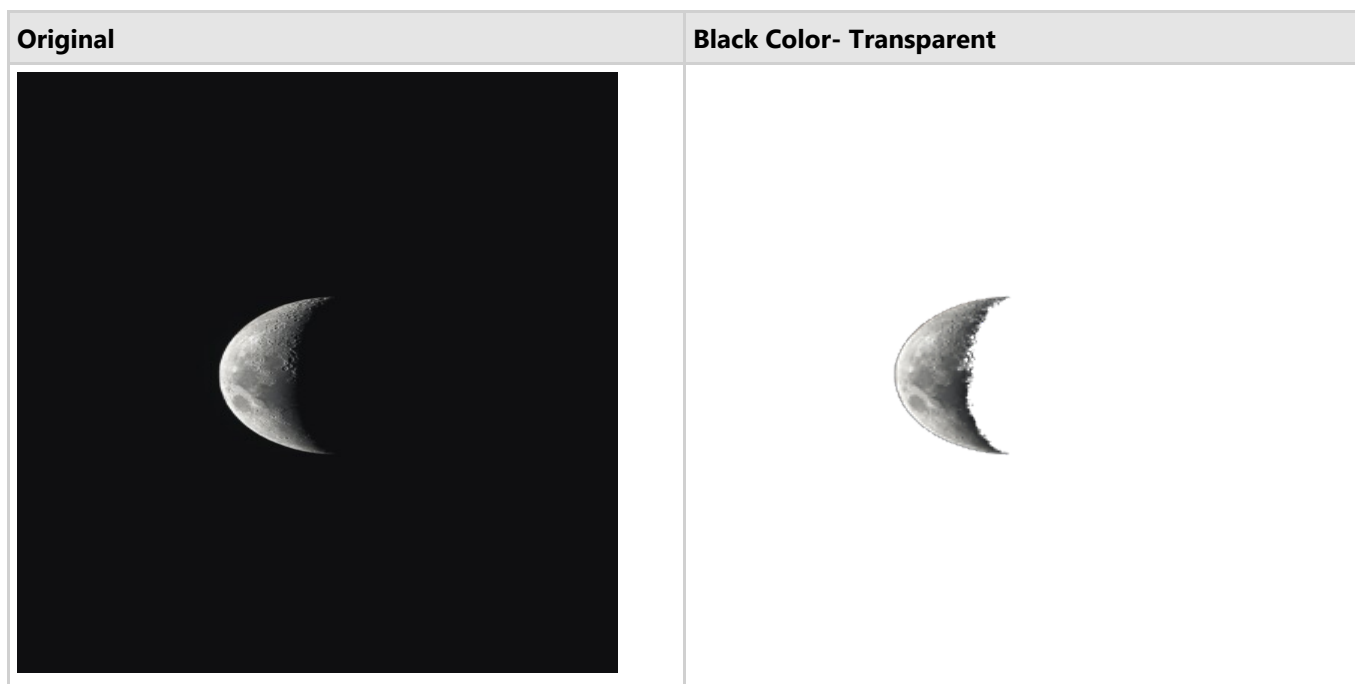


Set Picture Color Transparency

DsWord allows you to set picture color transparency using [SetColorTransparent](#) method of ImageData class.

Refer to the following example code to set the black color to transparent:

```
C#  
  
// Initialize GcWordDocument.  
GcWordDocument doc = new GcWordDocument();  
  
// Read PNG image from a file.  
var bytes = File.ReadAllBytes(@"image.png");  
  
// Add PNG picture into the document.  
var picture = doc.Body.AddParagraph().AddRun().AddPicture(bytes, @"image/png", 300f,  
300f);  
  
// Set black color transparent in the picture.  
picture.ImageData.SetColorTransparent(new UserColor(Color.FromArgb(15, 17, 19)));  
  
// Save the Word document.  
doc.Save(@"set-transparent-color.docx");
```




Limitations

DsWord does not support picture effects in exporting to PDF and images at this time.

Online Video

DsWord allows you to insert and play online videos directly from the document without leaving the document. You can add an online video to the document by inserting the URL of the video into the picture created with **Picture** class using **SetUrl** method of **WebVideoProperties** class.

SetUrl method has three overloads, allowing users to specify the input URL, title, height, and width of the video. This method generates the appropriate **EmbeddedHtml** property string internally, providing a convenient way for users to define these parameters that may be challenging to write directly.

 **Note:** EmbeddedHtml property will receive width and height either from parameters or directly from **Width** and **Height** properties of WebVideoProperties class when the user uses an overload without them. EmbeddedHtml, Width, and Height properties are serialized as distinct OpenXML tags.

Refer to the following example code to insert the URL of the video to add online video to the document:

```
C#  
  
// Initialize GcWordDocument.  
var doc = new GcWordDocument();  
  
// Add picture to document.  
var image = doc.Body.Paragraphs.Add().AddRun().AddPicture();  
  
// Set poster frame image.  
byte[] imageBytes1 = File.ReadAllBytes("sunrise.jpg");  
image.ImageData.SetImage(imageBytes1, contentType: "image/jpeg");  
  
// Add online video Url and set title, width, and height.  
image.ImageData.WebVideoProperties.SetUrl("https://www.youtube.com/watch?v=gsnqXt7dlmU&list=PL4Gr5tOAPttLOY9IrWVjJlv4CtkYI5cI_&index=2", "Sunrise", 600, 300);  
  
// Save the document.  
doc.Save("OnlineVideo.docx");
```



Limitations

Microsoft Word ignores the parameters of SetUrl method (width, height, and title) and Width and Height properties. The effects of these parameters and properties may be visible in alternative applications that also work with Word files.

Presets and Themed Styles

In DsWord, shape formatting can be applied by using either of the two ways below:

- LineFormat and FillFormat properties
- Shape styles

If LineFormat and FillFormat properties of a shape are not defined, the fill and line styles (shape styles) are used. But these shape styles are not picked up automatically (in absence of LineFormat and FillFormat properties). The shape styles are only applied when [ApplyThemedStyle](#) method is used.

Hence, **ApplyThemedStyle** method resets original shape formatting and force shape styles to be applied instead. Whereas [ApplyPreset](#) method overwrites its FillFormat and LineFormat properties to define a new formatting of shape.

DsWord provides a wide range of predefined preset and themed styles, of which:

- Presets can be applied on shapes and pictures both
- Themed styles can be applied only on shapes

Apply Presets

Shape class provides overloaded **ApplyPreset** methods where **LineShapePreset** parameter defines presets for non-fillable shapes like lines, curves etc. and **ShapePreset** parameter defines preset of colored shape and outline fills.

To apply preset on a non-fillable shape:

1. Add an arc to a Word document by using **Add** method of **ShapeCollection** class and pass **GeometryType.Arc** as its parameter.
2. Apply a preset on shape by using **ApplyPreset** method and set **LineShapePreset** enumeration to **Accent2ColorSolidDashArrowTail**.

```
C#  
  
var doc = new GcWordDocument();  
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();  
var shape = run.GetRange().Shapes.Add(200, 300, GeometryType.Arc);  
  
//Apply LineShapePreset as ShapePreset is not applicable to Arc  
shape.ApplyPreset(LineShapePreset.Accent2ColorSolidDashArrowTail);  
  
doc.Save("ShapePreset.docx");
```

Back to Top

Picture class provides **ApplyPreset** method to set preset on a picture.

To apply preset on a picture:


1. Add a picture to a Word document by using **Add** method of **PictureCollection** class and set image's path as a parameter to **SetImage** method.
2. Apply a preset on picture by using **ApplyPreset** method and set **PicturePreset** enumeration to **BeveledOvalBlack**.

```
C#  
  
var doc = new GcWordDocument();  
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();  
var pic = run.GetRange().Pictures.Add();  
pic.ImageData.SetImage(new Uri("Resources/folder.png"), "image/png");
```

```
//Apply picture preset
pic.ApplyPreset(PicturePreset.BeveledOvalBlack);

doc.Save("PicturePreset.docx");
```

Back to Top

 **Note:** Picture presets use heavy Effects, which are not supported currently. Hence, its results do not look like Word.

Apply Themed Styles

Shape class provides overloaded **ApplyThemedStyle** methods where **ThemeLineStyle** parameter defines themed styles for non-fillable shapes like lines, curves etc. and **ThemeShapeStyle** parameter defines themed styles of colored shape and outline fills.

To apply themed style on a shape:

1. Add a rounded rectangle shape to a Word document by using **Add** method of **ShapeCollection** class and pass **GeometryType.RoundRectangle** as its parameter.
2. Apply predefined themed style on shape by using **ApplyThemedStyle** method and set **ThemeShapedStyle** enumeration to **Light1SolidFillAccent5WiderOutline**.


```
C#
var doc = new GcWordDocument();
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();
var shape = run.GetRange().Shapes.Add(200, 300, GeometryType.RoundRectangle);

//Apply predefined themed style
shape.ApplyThemedStyle(ThemedShapeStyle.Light1SolidFillAccent5WiderOutline);

//Now shape has Fill.SolidFill.ThemeColor == ThemeColorId.Light1
//And outline Line.Fill.SolidFill colored to ThemeColor.Accent5

doc.Save("ApplyShapeStyle.docx");
```

Back to Top

 **Note:** In case of applying presets and themed styles on shapes, the use of appropriate overload method depends on the GeometryType of the Shape. If selected overload is not applicable to the shape's geometry, it will return false.

Limitations

Effects and derived classes (EffectsDAG, EffectsList) and custom geometry of shapes are not supported.

Shape Styles


DsWord allows you to define or customize shape styles to enhance the appearance of a shape. You can apply following effects to the shapes as well as pictures:

Fill and Line Effect

DsWord provides **Fill** and **Line** properties of the **ShapeStyle** class to get the fill and line style of a shape. The **StyleFill** class provides **ThemeFill** property which defines the fill style and **PlaceholderColor** property which overrides the predefined color of a theme fill style. Similarly, the **StyleLine** class provides **ThemeLine** property which defines the line style and **PlaceholderColor** property which overrides the predefined color of a theme line style. The **PlaceholderColor** can be of the type **SolidColor** or **GradientColor**.

The properties of **FormatScheme** class can be used to define the background fill styles, effect styles, fill styles, and line styles which define the style matrix for a theme.

To apply themed style on a shape, refer [Apply Themed Styles](#).

 **Note:** If **FillFormat** or **LineFormat** properties are not defined on a shape, their value is taken from Shape style (if present) by using **ApplyThemedStyle** method.

To change existing style on a shape:

1. Add a rectangle shape to a Word document by using **Add** method of **ShapeCollection** class and pass **GeometryType.Rectangle** as its parameter.
2. Apply themed style on shape by using **ApplyThemedStyle** method and set **ThemeShapedStyle** enumeration to **Accent6SolidFillLight1WidestOutline**.
3. Set **ThemeColor** to **Accent3** to change the placeholder color of predefined themed style.
4. Set the width of **ThemeLine** to 100 by using **Width** property of **LineFormat** class.
5. Set the line color of **ThemeLine** to yellow by using **RGB** property of **SolidColor** class.

```
C#
var doc = new GcWordDocument();
var run = doc.Body.Paragraphs.Add().GetRange().Runs.Add();
var shape = run.GetRange().Shapes.Add(300, 300, GeometryType.Rectangle);

//Apply style to shape which will have Accent6 color and Light1 colored outline
shape.ApplyThemedStyle(ThemedShapeStyle.Accent6SolidFillLight1WidestOutline);

//Change Shape style placeholder color to Accent3
//As it is Placeholder color, it will affect ONLY this shape
shape.Style.Fill.PlaceholderColor.ThemeColor = ThemeColorId.Accent3;

//Change Shape style line width. Now all shapes which apply this style will have
outline of width 100 pixels.
shape.Style.Line.ThemeLine.Width = 100;

//Change Style line color. Now it will ignore Placeholder color and will always
be yellow for every shape and ThemedStyle presets it used
shape.Style.Line.ThemeLine.Fill.SolidFill.RGB = System.Drawing.Color.Yellow;

doc.Save("ChangeShapeStyle.docx");
```


Shadow Effect

The shadow effect is used to add additional depth and definition to the foreground objects from the background. DsWord styles allow you to apply custom shadow effects defined in **FormatScheme** to your shape. Properties of FormatScheme class define the style matrix for a theme applied to the shadow. To use a specific color for the shadow, you can either set color directly in format scheme or you can use **PlaceholderColor** property which is used only if the color of shadow in FormatScheme is set to **None**.

Apply Shadow Styles using Direct Colors

DsWord allows you to apply theme styles to the shape by directly embedding the color in the format scheme.

See the code below to apply shadow styles to a shape using direct colors:

```
C#  
  
var fmtEffect = doc.Theme.FormatScheme.Effects.Add();  
// set properties for the PresetShadow  
fmtEffect.PresetShadow.Angle = 35f;  
  
// set direct color to PresetShadow  
fmtEffect.PresetShadow.Color.RGB = Color.Yellow;  
fmtEffect.PresetShadow.Distance = 50;  
fmtEffect.PresetShadow.Type = PresetShadowType.BackRightPerspectiveShadow;  
  
var para = doc.Body.Paragraphs.Add();  
var run = para.GetRange().Runs.Add();  
var shape = run.GetRange().Shapes.Add();  
shape.Fill.SolidFill.RGB = Color.Red;  
  
// set shape style using object of FormatScheme  
shape.Style.Effects.ThemeEffects = fmtEffect;
```

Apply Shadow Style using Placeholder Color

DsWord allows you to apply theme styles to the shape by referring to the external placeholders which are defined explicitly. This is to be noted that multiple styles can share a single format scheme, or you can define individual placeholders for each style.


See the code below to apply shadow styles to a shape using placeholder color:

```
C#  
  
var fmtEffect = doc.Theme.FormatScheme.Effects.Add();  
  
// set properties of PresetShadow  
fmtEffect.PresetShadow.Angle = 35f;  
  
// theme color is set to None, so this shadow will be colored by styles that use this  
// Effect using their PlaceholderColors.  
fmtEffect.PresetShadow.Color.ThemeColor = ThemeColorId.None;  
fmtEffect.PresetShadow.Distance = 50;  
fmtEffect.PresetShadow.Type = PresetShadowType.BackRightPerspectiveShadow;  
var para = doc.Body.Paragraphs.Add();  
var run = para.GetRange().Runs.Add();
```

```
var shape = run.GetRange().Shapes.Add();

// set the PlaceholderColor
shape.Style.Effects.PlaceholderColor.RGB = Color.Yellow;

// set shape style using object of FormatScheme
shape.Style.Effects.ThemeEffects = fmtEffect;
shape.Fill.SolidFill.RGB = Color.Red;
```

 **Note:** When you change fill, line or shadow properties of a predefined themed style, it may affect other styles as well because many themed style presets point to same fill, line and shadow properties.

Blur Effect

Blur effect refers to smoothing of images and its edges so that the image or a part of it appears out of focus. In DsWord, you can apply blur effect to image [directly](#) or through a defined format. This section talks about format or shape styles to apply this effect.

You can define a custom blur effect in FormatScheme of the shape and apply the effect through **FormatScheme** properties which define style matrix of the theme.



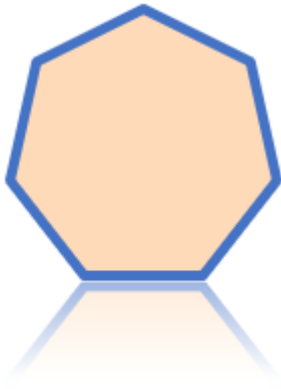
C#

```
// Shape Blur - shapes style:
p = doc.Body.Paragraphs.Add();
p.Style.ParagraphFormat.Spacing.SpaceBefore = 30;
run = p.GetRange().Runs.Add();
shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Star4);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.RGB = Color.Yellow;
shape.Line.Width = 4;
shape.Line.Fill.SolidFill.RGB = Color.Red;
// apply 7 point blur effect to the shape
var fmtEffect = doc.Theme.FormatScheme.Effects.Add();
fmtEffect.Blur.Radius = 7f;
//
shape.Style.Effects.ThemeEffects = fmtEffect;
p.GetRange().Runs.Add("Shape Blur - shapes style.",
doc.Styles[BuiltInStyleId.Strong]);
```

To view this sample code in action, see [Blur Effect demo](#) sample.

Reflection Effect

DsWord styles allow you to apply custom reflection effects defined in **FormatScheme** to your shape. Properties of **FormatScheme** class define the style matrix for a theme applied to the reflection. You can also set effects directly in the shape using **Shape.Effects.Reflection**. For more information, see [Reflection Format](#).



The code below show how to apply custom reflection to a shape using shape styles:

```
C#  
  
// Shape reflection - style effects:  
p = doc.Body.Paragraphs.Add();  
p.Style.ParagraphFormat.Spacing.SpaceBefore = 50;  
run = p.GetRange().Runs.Add();  
shape = shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Heptagon);  
shape.Fill.Type = FillType.Solid;  
shape.Fill.SolidFill.RGB = Color.PeachPuff;  
shape.Line.Width = 4;  
shape.Line.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;  
// Apply tight touching reflection effect to the new effect record:  
var fmtEffect = doc.Theme.FormatScheme.Effects.Add();  
fmtEffect.ApplyBuiltInReflection(BuiltInReflectionId.TightTouching);  
// Apply new effect style to shape:  
shape.Style.Effects.ThemeEffects = fmtEffect;
```

To view the code in action, see [Reflection Effect demo](#) sample.

Glow Effect

DsWord styles allow you to apply custom glow effects defined in **FormatScheme** to your shape. Properties of **FormatScheme** class define the style matrix for a theme applied to the glow. To use a specific color for the glow, you can either set color directly in format scheme or you can use **PlaceholderColor** property which is used only if the color of glow in **FormatScheme** is set to **None**.



Apply Glow Styles using Direct Colors

DsWord allows you to apply theme styles to the shape by directly embedding the color in the format scheme. The code below shows how to apply glow styles to a shape using direct color in the FormatScheme's effect:

C#

```
var shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Cloud);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;
// apply 18 point accent 6 glow effect to the shape's style
var fmtEffect = doc.Theme.FormatScheme.Effects.Add();
fmtEffect.ApplyBuiltInGlow(BuiltInGlowId.Radius18Accent6);
shape.Style.Effects.ThemeEffects = fmtEffect;
p.GetRange().Runs.Add("Shape Glow - shapes style - direct color in format scheme's effect.", doc.Styles[BuiltInStyleId.Strong]);
```

Apply Glow Styles using Placeholder Color

DsWord allows you to apply theme styles to the shape by referring to the external placeholders which are defined explicitly. This is to be noted that multiple styles can share a single format scheme, or you can define individual placeholders for each style.

The code below shows how to apply glow styles to a shape using placeholder color:

C#

```
Shape shape = run.GetRange().Shapes.Add();
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;

var fmtEffect = doc.Theme.FormatScheme.Effects.Add();
fmtEffect.Glow.Color.ThemeColor = ThemeColorId.None;
// apply 18 point accent 6 glow effect to the shape's style
fmtEffect.Glow.Radius = 18f;
shape.Style.Effects.PlaceholderColor.ThemeColor = ThemeColorId.Accent6;
shape.Style.Effects.PlaceholderColor.Transparency = 0.6f;

shape.Style.Effects.ThemeEffects = fmtEffect;
```

To view the code in action, see [Glow Effect demo](#) sample.

SoftEdge Effect

In DsWord, you can apply SoftEdge effect to image [directly](#) or through a defined format. This section talks about format or shape styles to apply this effect.

You can define a custom SoftEdge effect in FormatScheme of the shape and apply the effect through **FormatScheme** properties which define style matrix of the theme.



C#

```
// Shape Soft Edge - shapes style:
Paragraph p = doc.Body.Paragraphs.Add();
p.Style.ParagraphFormat.Spacing.SpaceBefore = 30;
Run run = p.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add(100, 100, GeometryType.Ellipse);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.RGB = Color.Yellow;
shape.Line.Width = 8;
shape.Line.Fill.SolidFill.RGB = Color.Red;

// apply 5 point soft edge effect to the style
var fmtEffect = doc.Theme.FormatScheme.Effects.Add();
fmtEffect.SoftEdge.Radius = 5f;
//
shape.Style.Effects.ThemeEffects = fmtEffect;

p.GetRange().Runs.Add("Shape Soft Edge - shapes style.",
doc.Styles[BuiltInStyleId.Strong]);
```

To view this sample code in action, see [SoftEdge Effect demo](#) sample.

FillOverlay Effect

DsWord styles allow you to apply custom FillOverlay effects defined in **FormatScheme** to your shape. Properties of FormatScheme class define the style matrix for a theme applied to the FillOverlay. To use a specific color for the FillOverlay, you can either set color directly in format scheme or you can use **PlaceholderColor** property which is used only if the color of FillOverlay in FormatScheme is set to **None**.



Apply FillOverlay Styles using Direct Colors

DsWord allows you to apply theme styles to the shape by directly embedding the color in the format scheme.

The code below shows how to apply FillOverlay styles to a shape using direct color in the FormatScheme's effect:

C#

```
// Shape FillOverlay - style effects, fixed color:
Paragraph p = doc.Body.Paragraphs.Add();
p.Style.ParagraphFormat.Spacing.SpaceBefore = 30;
Run run = p.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add(100, 100, GeometryType.AccentCallout3);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.RGB = Color.LightBlue;
// apply solid fill overlay with another color to mix with the main fill
var fmtEffect = doc.Theme.FormatScheme.Effects.Add();
FillOverlay overlay = fmtEffect.FillOverlay;
overlay.BlendMode = BlendMode.Multiply;
overlay.Fill.Type = FillType.Solid;
overlay.Fill.SolidFill.RGB = Color.Yellow;
shape.Style.Effects.ThemeEffects = fmtEffect;
p.GetRange().Runs.Add("Shape FillOverlay - style effects, fixed color.",
doc.Styles[BuiltInStyleId.Strong]);
```

Apply FillOverlay Styles using Placeholder Color

DsWord allows you to apply theme styles to the shape by referring to the external placeholders which are defined explicitly. This is to be noted that multiple styles can share a single format scheme, or you can define individual placeholders for each style.

The code below shows how to apply FillOverlay styles to a shape using placeholder color:

C#

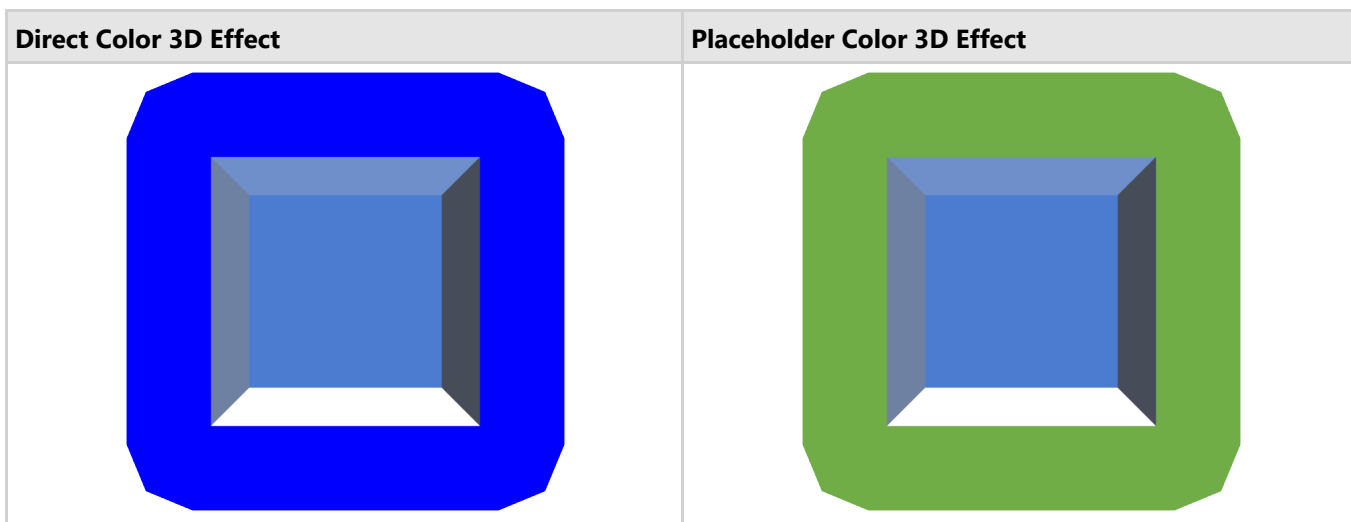
```
// Shape FillOverlay - style effects, fixed and placeholder color usage, pattern
color
// (BackColor uses fixed color, ForeColor uses placeholder color):
Paragraph p = doc.Body.Paragraphs.Add();
p.Style.ParagraphFormat.Spacing.SpaceBefore = 30;
Run run = p.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add(100, 100, GeometryType.CircularArrow);
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.RGB = Color.LightBlue;
// Apply pattern fill overlay with another color to mix with the main fill:
```

```
var fmtEffect = doc.Theme.FormatScheme.Effects.Add();
FillOverlay overlay = fmtEffect.FillOverlay;
overlay.BlendMode = BlendMode.Multiply;
//
var foFill = overlay.Fill;
foFill.Type = FillType.Pattern;
foFill.PatternFill.Type = PatternFillType.DashedHorizontal;
foFill.PatternFill.ForeColor.ThemeColor = ThemeColorId.None;
foFill.PatternFill.BackColor.ThemeColor = ThemeColorId.Accent6;
//
shape.Style.Effects.PlaceholderColor.ThemeColor = ThemeColorId.Accent3;
shape.Style.Effects.PlaceholderColor.Transparency = 0.6f;
shape.Style.Effects.ThemeEffects = fmtEffect;
p.GetRange().Runs.Add("Shape FillOverlay - style effects, fixed and placeholder color usage, pattern color.", doc.Styles[BuiltInStyleId.Strong]);
```

To view the code in action, see [FillOverlay Effect demo](#) sample.

3D Effect

DsWord styles allow you to apply custom 3D effects defined in [FormatScheme](#) to your shape. Properties of [FormatScheme](#) class define the style matrix for a theme applied to the 3D effect. To use a specific color for the 3D effect, you can either set color directly in format scheme or you can use **PlaceholderColor** property which is used only if the color of 3D effect in [FormatScheme](#) is set to **None**. You can also set the 3D effect directly in the shape using [ThreeDFormat](#) and [ThreeDScene](#) classes and the [ApplyEffectsPreset](#) method. For more information, see [3D Format](#).



Apply 3D Styles using Direct Colors

DsWord allows you to apply theme styles to the shape by directly embedding the color in the format scheme. Refer to the following example code to add a direct color 3D effect:

```
C#
// Initialize GcWordDocument.
var doc = new GcWordDocument();
```

```
// Add a shape to the document.
Paragraph p = doc.Body.Paragraphs.Add();
Run run = p.GetRange().Runs.Add();
Shape shape = run.GetRange().Shapes.Add();
shape.Fill.Type = FillType.Solid;
shape.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;

// Set the shape format.
var shapeFormat = shape.Effects.ThreeDFormat;

var schemeEffects = doc.MainDocument.Theme.FormatScheme.Effects.Add();
var schemeFormat = schemeEffects.ThreeDFormat;
var schemeScene = schemeEffects.ThreeDScene;

// Apply top bevel.
schemeFormat.TopBevel.Type = BevelType.Angle;
schemeFormat.TopBevel.Width = 18.5f;
schemeFormat.TopBevel.Height = 058.5f;

// Apply bottom bevel.
schemeFormat.BottomBevel.Type = BevelType.ArtDeco;
schemeFormat.BottomBevel.Width = 11.5f;
schemeFormat.BottomBevel.Height = 21.5f;

// Set the depth of the shape.
schemeFormat.Depth.Width = 33.5f;
schemeFormat.Depth.Color.ThemeColor = ThemeColorId.Accent3;

// Add contour to the shape.
schemeFormat.Contour.Width = 43.5f;
schemeFormat.Contour.Color.RGB = Color.Blue;

// Set the shape material.
schemeFormat.Material = MaterialType.Metal;

// Set the DistanceFromGround.
schemeFormat.DistanceFromGround = 48.5f;

// Set the 3D scene.
schemeScene.Camera.Preset = CameraPreset.OrthographicFront;
schemeScene.Lighting.Direction = LightRigDirection.Top;
schemeScene.Lighting.Type = LightRigType.ThreePoints;

// Apply the effect to the shape.
shape.Style.Effects.ThemeEffects = schemeEffects;

// Save the Word document.
doc.Save("SampleShapeStyleThreeDEffectsDirectColors.docx");
```

Apply 3D Styles using Placeholder Color

DsWord allows you to apply theme styles to the shape by referring to the external placeholders which are defined

explicitly. This is to be noted that multiple styles can share a single format scheme, or you can define individual placeholders for each style.

Refer to the following example code to add a placeholder color 3D effect:

```
C#  
  
// Initialize GcWordDocument.  
var doc = new GcWordDocument();  
  
// Add a shape to the document.  
Paragraph p = doc.Body.Paragraphs.Add();  
Run run = p.GetRange().Runs.Add();  
Shape shape = run.GetRange().Shapes.Add();  
shape.Fill.Type = FillType.Solid;  
shape.Fill.SolidFill.ThemeColor = ThemeColorId.Accent1;  
  
// Set the shape format.  
var shapeFormat = shape.Effects.ThreeDFormat;  
  
var schemeEffects = doc.MainDocument.Theme.FormatScheme.Effects.Add();  
var schemeFormat = schemeEffects.ThreeDFormat;  
var schemeScene = schemeEffects.ThreeDScene;  
  
// Apply top bevel.  
schemeFormat.TopBevel.Type = BevelType.Angle;  
schemeFormat.TopBevel.Width = 18.5f;  
schemeFormat.TopBevel.Height = 058.5f;  
  
// Apply bottom bevel.  
schemeFormat.BottomBevel.Type = BevelType.ArtDeco;  
schemeFormat.BottomBevel.Width = 11.5f;  
schemeFormat.BottomBevel.Height = 21.5f;  
  
// Set the depth of the shape.  
schemeFormat.Depth.Width = 33.5f;  
schemeFormat.Depth.Color.ThemeColor = ThemeColorId.Accent1;  
  
// Add contour to the shape.  
schemeFormat.Contour.Width = 43.5f;  
  
// Set ThemeColor to none. ThemeColor will be retrieved from the placeholder.  
schemeFormat.Contour.Color.ThemeColor = ThemeColorId.None;  
  
// Set the shape material.  
schemeFormat.Material = MaterialType.Metal;  
  
// Set the DisnatceFromGround.  
schemeFormat.DistanceFromGround = 48.5f;  
  
// Set the 3D scene.  
schemeScene.Camera.Preset = CameraPreset.OrthographicFront;  
schemeScene.Lighting.Direction = LightRigDirection.Top;
```

```
schemeScene.Lighting.Type = LightRigType.ThreePoints;  
  
// Set the placeholder color.  
shape.Style.Effects.PlaceholderColor.ThemeColor = ThemeColorId.Accent6;  
  
// Apply the effect to the shape.  
shape.Style.Effects.ThemeEffects = schemeEffects;  
  
// Save the Word document.  
doc.Save("SampleShapeStyleThreeDEffectsPlaceholderColor.docx");
```

Limitation

DsWord does not support the export of 3D effects to PDF or images.

Office Math

Office Math in a Word document is a combination of mathematical symbols or text. For example, $A = \pi r^2$ can be converted and written in a Word document as:

$$A = \pi r^2$$

DsWord allows you to read, add, edit, and display Office Math content in a Word document using [OMathParagraph](#) and [OMath](#) classes. [OMathParagraph](#) represents a paragraph that contains Office Math content, while [OMath](#) represents an inline Office Math zone and can be included in an [OMathParagraph](#) or a regular paragraph. Also, you can add [OMathParagraph](#) and [OMath](#) to a **SimpleField**, **Hyperlink**, **BidirectionalOverride**, and **ControlContent** in an inline content control.

DsWord provides specialized classes such as [OMathBar](#), [OMathBorderBox](#), [OMathBox](#), [OMathDelimiter](#), [OMathEquationArray](#), [OMathFraction](#), [OMathFunction](#), etc. to represent the various math structures inside an [OMath](#) zone. These classes are derived from the common abstract [OMathStruct](#) base class. Each class implementation has its own strong structure of **OMathElements**. [OMathElement](#) represents an element inside the [OMathStruct](#) class instance. You can add [OMathElement](#) only to [OMathDelimiter](#) and [OMathEquationArray](#) class instances manually; in other [OMathStruct](#) implementations, their position is predefined.

Besides Office Math content, [OMath](#) and [OMathElement](#) classes can contain the following objects: [SimpleField](#), [Hyperlink](#), and [ContentControl](#). Also, these classes and [OMathParagraph](#) class can contain [ContentMark](#) and [Run](#) objects. When [Run](#) object is inside the Office Math zone (in [OMath](#) or [OMathElement](#)), its new property [IsOMathRun](#) sets to true. In this case, the new property [OMathFormat](#) became available for additional mathematical formatting through the [OMathFormat](#) class. The default appearance and behavior of Office Math content are available with the new [OMathOptions](#) property in [Settings](#) class through [OMathOptions](#) class.

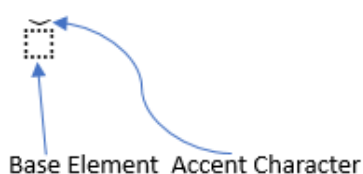
You can access the Office Math content using [OMathParagraphs](#), [OMaths](#), [OMathStructs](#), [OMathElements](#) and [OMathMatrixRows](#) properties of [RangeBase](#) class.

Working with OMathElements

OMathElements allow you to create various complex equations with different structures using accent, bar, border-box, box, delimiter, and so on. The following sections demonstrate the usage of such [OMathElements](#).

Accent

An accent is a single Unicode symbol present over a base element. You can add an accent symbol to an equation using [AddAccent](#) method of [OMath](#) class.



Refer to the following example code to add an accent over a fraction:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add accent to the equation.
var accent = om.AddAccent("", "\u0308");
accent.Base.AddFraction("2", "5", OMathFractionType.Bar);

// Save the Word document.
```

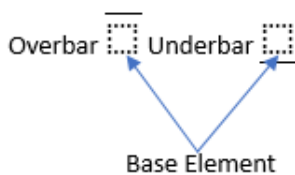
```
sampleDoc.Save("Accent.docx");
```

$$\bar{2}$$

$$\underline{5}$$

Bar

A bar is a horizontal line present over or below a base element. You can add a bar to an equation using [AddBar](#) method of OMath class. By default, the bar is added below the base element.



Refer to the following example code to add a bar over a function:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

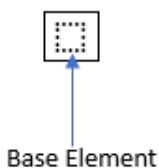
// Add bar to the equation.
var bar = om.AddBar();
bar.Base.AddFunction("cos", "2x");
bar.Position = OMathVerticalJustification.Top;

// Save the Word document.
sampleDoc.Save("Bar.docx");
```

$$\overline{\cos 2x}$$

Border Box

A border box is a border present around a base element with four sides. You can add a border box to an equation using [AddBorderBox](#) method of OMath class. By default, a box is drawn around the base element, but you can also add strikethrough in horizontal, vertical, and diagonal directions.



Refer to the following example code to add a border box around an equation:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
```

```
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add borderbox to the equation.
var box = om.AddBorderBox("2+2*");
box.Base.AddFraction("3", "3", OMathFractionType.Bar);
box.Base.AddRun(" = 5");

// Save the Word document.
sampleDoc.Save("BorderBox.docx");
```

$$2 + 2 * \frac{3}{3} = 5$$

Box

A box represents a box that groups components of an equation or other instance of mathematical text. You can add a box to an equation using [AddBox](#) method of OMath class.



Refer to the following example code to add two runs with a box element between them:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

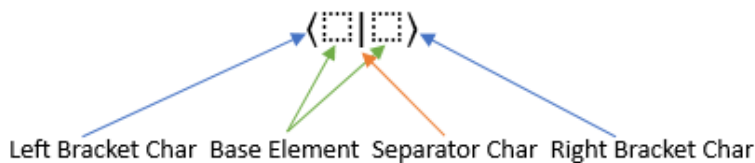
// Add box to the equation.
om.AddRun("a");
var box = om.AddBox("==");
om.AddRun("b");
box.IsOperatorEmulator = true;

// Save the Word document.
sampleDoc.Save("Box.docx");
```

$$a == b$$

Delimiter

A delimiter is a mark or symbol that represents the beginning or end of separate elements of an equation with a designated separator between each element. You can add a delimiter to an equation using [AddDelimiter](#) method of OMath class. By default, the beginning and end characters will be parentheses, and the separator will be a pipe symbol.



Refer to the following example code to add delimiters to an equation:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add delimiter to the equation.
var delimiter = om.AddDelimiter(new string[] { "5", "7" }, "[", "]", "\\");
delimiter.Items.Add().AddRadical("n", "7");

// Save the Word document.
sampleDoc.Save("Delimiter.docx");
```

$$[5\7\sqrt{n}]$$

Equation Array

An equation array is a one-dimensional vertical array consisting of one or more equations or expression elements. You can add an equation array using [AddEquationArray](#) method of OMath class.



Refer to the following example code to add an equation array:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add equation array.
var ea = om.AddEquationArray();
ea.Items.Add().AddDelimiter(new string[] { "x", "y", "z" });
ea.Items.Add().AddFraction("n", "2", OMathFractionType.Skewed);

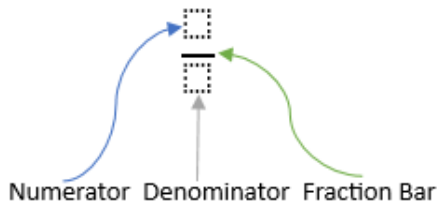
// Save the Word document.
sampleDoc.Save("EquationArray.docx");
```

$$(x|y|z)$$

$$\pi/2$$

Fraction

A fraction consists of a numerator element and a denominator element separated by a fraction bar. You can add a fraction using [AddFraction](#) method of OMath class.



Refer to the following example code to add a fraction:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add fraction.
var fr = om.AddFraction();
var func = fr.Numerator.AddFunction("sin", "");
func.Argument.AddFraction("π", "3", OMathFractionType.Bar);
fr.Denominator.AddRadical("e", "3");

// Save the Word document.
sampleDoc.Save("Fraction.docx");
```

$$\frac{\sin \frac{\pi}{3}}{\sqrt[3]{e}}$$

Function

A function consists of a function name element and an argument element. You can add a function using [AddFunction](#) method of OMath class.



Refer to the following example code to add a function:

C#

```
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

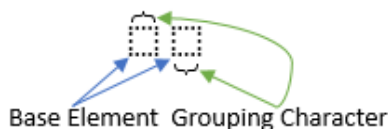
// Add function.
var f = om.AddFunction("cos", "");
f.Argument.AddFraction("π", "2", OMathFractionType.Bar);

// Save the Word document.
sampleDoc.Save("Function.docx");
```

$$\cos \frac{\pi}{2}$$

Group Character

A group character is a Unicode symbol present above or below the base element. You can add a group character using [AddGroupCharacter](#) method of OMath class.



Refer to the following example code to add a group character to the function:

C#

```
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add group character.
var gc = om.AddGroupCharacter("xyz", "\u23DE");
gc.Position = OMathVerticalJustification.Top;
gc.VerticalJustification = OMathVerticalJustification.Bottom;

// Save the Word document.
sampleDoc.Save("GroupCharacter.docx");
```

$$\overline{xyz}$$

Lower Limit

A lower limit is the lowest value of any value or expression. You can add a lower limit using [AddLimitLower](#) method of OMath class.



Refer to the following example code to add a lower limit to the function:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

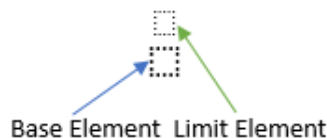
// Add lower limit.
om.AddLimitLower("abc", "z");

// Save the Word document.
sampleDoc.Save("LowerLimit.docx");
```

$$\underset{z}{abc}$$

Upper Limit

An upper limit is the highest value of any value or expression. You can add an upper limit using [AddLimitUpper](#) method of [OMath](#) class.



Refer to the following example code to add an upper limit to the function:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

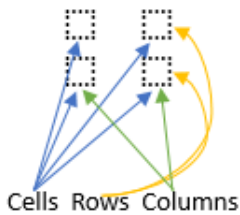
// Add upper limit.
om.AddLimitUpper("abc", "y");

// Save the Word document.
sampleDoc.Save("UpperLimit.docx");
```

$$\overset{y}{abc}$$

Matrix

A matrix is a rectangular array of elements in one or more rows and columns. DsWord provides [OMathMatrix](#) class, which is a special type of [OMathStruct](#). [OMathMatrix](#) represents a math matrix where you can add or remove rows and columns represented by [OMathMatrixRow](#) and [OMathMatrixColumn](#), respectively. Each cell in [OMathMatrix](#) is [OMathElement](#) instance. You can add a matrix using [AddMatrix](#) method of [OMath](#) class.



Refer to the following example code to add a matrix:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add matrix.
om.AddMatrix(new int?[2, 2] { { 1, 2 }, { 3, 4 } });

// Save the Word document.
sampleDoc.Save("Matrix.docx");
```

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

N-Array

An N-Array consists of an n-ary operator character, a base element (or operand), and optional upper and lower limit elements. You can add an N-Array using [AddNary](#) method of OMath class.



Refer to the following example code to add an N-Array:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add N-Array.
om.AddNary("∫", "0", "∞", "fff");

// Save the Word document.
sampleDoc.Save("Nary.docx");
```

$$\iiint_0^{\infty} x dx$$

Phantom

A phantom structure adds the spacing of the phantom base element without displaying that base and suppresses part of the glyph for spacing considerations. You can add a phantom structure using [AddPhantom](#) method of OMath class.



Refer to the following example code to add a phantom structure:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

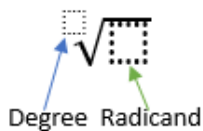
// Add phantom structure.
var del = om.AddDelimiter(null, "<", ">", "");
var ph = del.Items.Add().AddPhantom("argument", false);
ph.Show = false;

// Save the Word document.
sampleDoc.Save("Phantom.docx");
```



Radical

A radical symbol denotes the square root or nth root, and it creates a radical expression comprising a radicand and a degree element. You can add a radical using [AddRadical](#) method of OMath class.



Refer to the following example code to add a radical expression:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add radical expression.
var radical = om.AddRadical();
radical.Degree.AddFraction("2", "3", OMathFractionType.Skewed);
radical.Radicand.AddFunction("cos", "2x");
```

```
// Save the Word document.
sampleDoc.Save("Radical.docx");
```

$$\sqrt[2/3]{\cos 2x}$$

Subscript

A subscript is a character, such as a number or letter, placed slightly below the normal line. You can add a subscript using [AddSubscript](#) method of OMath class.



Refer to the following example code to add a subscript:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add subscript.
var func = om.AddFunction("", "2x+1");
func.Name.AddSubscript("log", "3");

// Save the Word document.
sampleDoc.Save("Subscript.docx");
```

$$\log_3 2x + 1$$

Superscript

A superscript is a character, such as a number or letter, placed slightly above the normal line. You can add a superscript using [AddSuperscript](#) method of OMath class.



Refer to the following example code to add a superscript:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add superscript.
var func = om.AddFunction();
func.Name.AddSuperscript("cos", "2");
var frac = func.Argument.AddFraction();
```

```
frac.Numerator.AddRun("7x");
frac.Denominator.AddRun("2π");

// Save the Word document.
sampleDoc.Save("Superscript.docx");
```

$$\cos^2 \frac{7x}{2\pi}$$

PreSubSuperscript

A pre-sub-superscript structure consists of a base element, a reduced-size script element placed below and to the left, and a reduced-size script element placed above and to the left. You can add a pre-sub-superscript structure using [AddPreSubSuperscript](#) method of OMath class.



Refer to the following example code to add a pre-sub-superscript structure:

```
C#
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add pre-sub-superscript structure.
var pss = om.AddPreSubSuperscript("", "-x", "2x");
pss.Base.AddRadical("n", "").HideDegree = true;

// Save the Word document.
sampleDoc.Save("PreSubSuperscript.docx");
```

$$\frac{2x}{-x} \sqrt{n}$$

SubSuperscript

A SubSuperscript consists of a base element, a reduced-size script element placed below and to the right, and a reduced-size script element placed above and to the right. You can add a SubSuperscript using [AddSubSuperscript](#) method of OMath class.



Refer to the following example code to add a SubSuperscript:

```
C#
// Initialize GcWordDocument.
```

```
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add SubSuperscript.
var subs = om.AddSubSuperscript("", "-x", "2x");
subs.Base.AddRadical("n", "").HideDegree = true;

// Save the Word document.
sampleDoc.Save("SubSuperscript.docx");
```

$$\sqrt[n]{-x}^{2x}$$

BuiltIn Equations

You can add built-in equations supported by MS Word using Add method of RangeBase, OMathParagraph, OMath, and OMathElement classes that accept [OMathBuiltInEquation](#) enumeration value identifying the desired equation.

Add BuiltIn Equation

1. Access Office Math paragraph collection using **OMathParagraphs** property of RangeBase class.
2. Add a builtIn equation using [Add](#) method of [OMathParagraphCollection](#) class and pass the builtIn equation as a parameter by choosing the appropriate value from OMathBuiltInEquation enumeration .

```
C#
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Add an Office Math paragraph with a built-in area of a circle equation.
doc.Body.Paragraphs.Add().GetRange().OMathParagraphs.Add(OMathBuiltInEquation.AreaOfCircle);

// Save the Word document.
doc.Save("AreaofCircleBuiltin.docx");
```

MathML Converter

You can convert Office Math content to MathML and vice versa using **MathMLConverter** class. Also, it allows you to insert MathML directly into a RangeBase and converts the MathML to Office Math automatically.

1. Access the Office Math paragraph collection using **OMathParagraphs** property of **RangeBase** class and add Office Math paragraph using **Add** method of **OMathParagraphCollection** class.
2. Implement a text reader to read from a string using **StringReader** class.
3. Create an XML reader to read MathML from the string using **Create** method of **XmlReader** class.
4. Create a MathML converter and add the content to OMathParagraph using [FromMathML](#) method of [MathMLConverter](#) class.

```
C#
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Add an Office Math paragraph.
OMathParagraph oMathParagraph = doc.Body.Paragraphs.Add().GetRange().OMathParagraphs.Add();
```

```
// Create an XML reader from the MathML area of the circle string.
StringReader stringReader = new StringReader("<mml:math
xmlns:mml=\"http://www.w3.org/1998/Math/MathML\"\" +
    \" xmlns:m=\"http://schemas.openxmlformats.org/officeDocument/2006/math\">
<mml:mi>A</mml:mi><mml:mo>=</mml:mo>\" +
    \"<mml:mi>\pi</mml:mi><mml:msup><mml:mrow><mml:mi>r</mml:mi></mml:mrow><mml:mrow>
<mml:mn>2</mml:mn></mml:mrow></mml:msup></mml:math>");
XmlReader xmlReader = XmlReader.Create(stringReader);

// Create a MathML converter.
MathMLConverter converter = new MathMLConverter();

// Add the MathML area of the circle.
converter.AddMathML(xmlReader, oMathParagraph);

// Save the Word document.
doc.Save("AreaofCircleMathml.docx");
```

Limitation

Export of Office Math content to PDF or images is not yet supported.

Complex Equation Examples

The following example codes demonstrate various complex equations that can be added using Office Math APIs of DsWord library:

Equation 1

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right)$$

C#

```
// Initialize GcWordDocument.
var doc = new GcWordDocument();
var om = doc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add f(x)=a0+ to the paragraph.
om.AddRun("f");

// Add delimiter. It inserts ( and ) before and after its items.
om.AddDelimiter(new string[] { "x" });
om.AddRun("=");
om.AddSubscript("a", "0");
om.AddRun("+");

// Add sum from 1 till infinity.
var nary = om.AddNary("", "n=1", "∞", "Σ");
nary.Grow = true;

// Add delimiters "()".
var deli = nary.Base.AddDelimiter();
var item = deli.Items.Add();

// Add an cos nπx/L.
item.AddSubscript("a", "n");

var func = item.AddFunction();
func.Name.AddRun("cos").OMathFormat.IsNormalText = true;
func.Argument.AddFraction("nπx", "L", null);

// Add + bn sin nπx/L.
item.AddRun("+");
item.AddSubscript("b", "n");

func = item.AddFunction("", "");
func.Name.AddRun("sin").OMathFormat.IsNormalText = true;
func.Argument.AddFraction("nπx", "L", null);
```



```
// Save the Word document.
doc.Save("Equation.docx");
```

Equation 2

$$\exists x \left(\text{Person}(x) \wedge \forall y (\text{Time}(y) \rightarrow \text{Happy}(x, y)) \right)$$

C#

```
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add ∃x.
om.AddRun("∃x");

// Add delimiter. It inserts ( and ) before and after its items.
var item = om.AddDelimiter().Items.Add();

// Add equation.
item.AddRun("Person").OMathFormat.IsNormalText = true;
item.AddDelimiter(new string[] { "x" });
item.AddRun("∧∀y");
item = item.AddDelimiter().Items.Add();
item.AddRun("Time").OMathFormat.IsNormalText = true;
item.AddDelimiter(new string[] { "y" });
item.AddRun("-");
item.AddRun("Happy").OMathFormat.IsNormalText = true;
item.AddDelimiter(new string[] { "x,y" });

// Save the Word document.
sampleDoc.Save("Equation.docx");
```

Equation 3

$$\sin \alpha \pm \sin \beta = 2 \sin \frac{1}{2}(\alpha \pm \beta) \cos \frac{1}{2}(\alpha \mp \beta)$$

C#

```
// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add equation.
```

```

var func = om.AddFunction("sin", "α");
func.Name.GetRange().Runs.First.Font.Italic = false;
om.AddRun("±");
func = om.AddFunction("sin", "β");
func.Name.GetRange().Runs.First.Font.Italic = false;
om.AddRun("=2");
func = om.AddFunction("sin", "");
func.Name.GetRange().Runs.First.Font.Italic = false;
func.Argument.AddFraction("1", "2", null);
func.Argument.AddDelimiter(new string[] { "α±β" });
func = om.AddFunction("cos", "");
func.Name.GetRange().Runs.First.Font.Italic = false;
func.Argument.AddFraction("1", "2", null);
func.Argument.AddDelimiter(new string[] { "α∓β" });

// Save the Word document.
sampleDoc.Save("Equation.docx");

```

Equation 4

$$Z(v) = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} \frac{e^{-t^2} dt}{t - iv} = i \int_0^{\infty} e^{-vt - t^2/4} dt$$

C#

```

// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add equation.
om.AddRun("Z");
om.AddDelimiter(new string[] { "v" });
om.AddRun("=");
var frac = om.AddFraction("1", "", null);
var radical = frac.Denominator.AddRadical();
radical.Radicand.AddRun("π");
radical.HideDegree = true;
var nary = om.AddNary("", "-∞", "∞", "f");
frac = nary.Base.AddFraction();
var superscript = frac.Numerator.AddSuperscript("e", "");
superscript.Superscript.AddSuperscript("-t", "2");
frac.Numerator.AddRun("dt");
frac.Denominator.AddRun("t-i");
frac.Denominator.AddRun("v").Font.Bidi = true;
om.AddRun("=i");
nary = om.AddNary("", "0", "∞", "f");
superscript = nary.Base.AddSuperscript("e", "-vt-");
superscript.Superscript.AddSuperscript("t", "2");

```

```

superscript.Superscript.AddRun("/4");
nary.Base.AddRun("dt");

// Save the Word document.
sampleDoc.Save("Equation.docx");

```

Equation 5

$$\begin{pmatrix} U(t) \\ V(t) \\ W(t) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos Rt & -\sin Rt \\ 0 & \sin Rt & \cos Rt \end{pmatrix} \begin{pmatrix} U(0) \\ V(0) \\ W(0) \end{pmatrix}$$

C#

```

// Initialize GcWordDocument.
var sampleDoc = new GcWordDocument();
var om = sampleDoc.Body.AddParagraph().AddOMathParagraph().AddOMath();

// Add first delimiter.
var delimiter = om.AddDelimiter();
var item = delimiter.Items.Add();

// Add first matrix.
var matrix = item.AddMatrix(3, 1);

foreach (var column in matrix.Columns)
    column.HorizontalAlignment = OMathHorizontalAlignment.Center;
matrix.Rows[0].Cells[0].AddRun("U");
matrix.Rows[0].Cells[0].AddDelimiter(new string[] { "t" });
matrix.Rows[1].Cells[0].AddRun("V");
matrix.Rows[1].Cells[0].AddDelimiter(new string[] { "t" });
matrix.Rows[2].Cells[0].AddRun("W");
matrix.Rows[2].Cells[0].AddDelimiter(new string[] { "t" });

om.AddRun("=");

// Add second delimiter.
delimiter = om.AddDelimiter();
item = delimiter.Items.Add();

// Add second matrix.
matrix = item.AddMatrix(3, 3);
foreach (var column in matrix.Columns)
    column.HorizontalAlignment = OMathHorizontalAlignment.Center;

matrix.Rows[0].Cells[0].AddRun("1");
matrix.Rows[0].Cells[1].AddRun("0");
matrix.Rows[0].Cells[2].AddRun("0");

matrix.Rows[1].Cells[0].AddRun("0");

```

```
matrix.Rows[1].Cells[1].AddFunction("cos",
"Rt").Name.GetRange().Runs.First.Font.Italic = false;
matrix.Rows[1].Cells[2].AddRun("-");
matrix.Rows[1].Cells[2].AddFunction("sin",
"Rt").Name.GetRange().Runs.First.Font.Italic = false;

matrix.Rows[2].Cells[0].AddRun("0");
matrix.Rows[2].Cells[1].AddFunction("sin",
"Rt").Name.GetRange().Runs.First.Font.Italic = false;
matrix.Rows[2].Cells[2].AddFunction("cos",
"Rt").Name.GetRange().Runs.First.Font.Italic = false;

// Add third delimiter.
delimiter = om.AddDelimiter();
item = delimiter.Items.Add();

// Add third matrix.
matrix = item.AddMatrix(3, 1);

foreach (var column in matrix.Columns)
    column.HorizontalAlignment = OMathHorizontalAlignment.Center;
matrix.Rows[0].Cells[0].AddRun("U");
matrix.Rows[0].Cells[0].AddDelimiter(new string[] { "0" });
matrix.Rows[1].Cells[0].AddRun("V");
matrix.Rows[1].Cells[0].AddDelimiter(new string[] { "0" });
matrix.Rows[2].Cells[0].AddRun("W");
matrix.Rows[2].Cells[0].AddDelimiter(new string[] { "0" });

// Save the Word document.
sampleDoc.Save("Equation.docx");
```

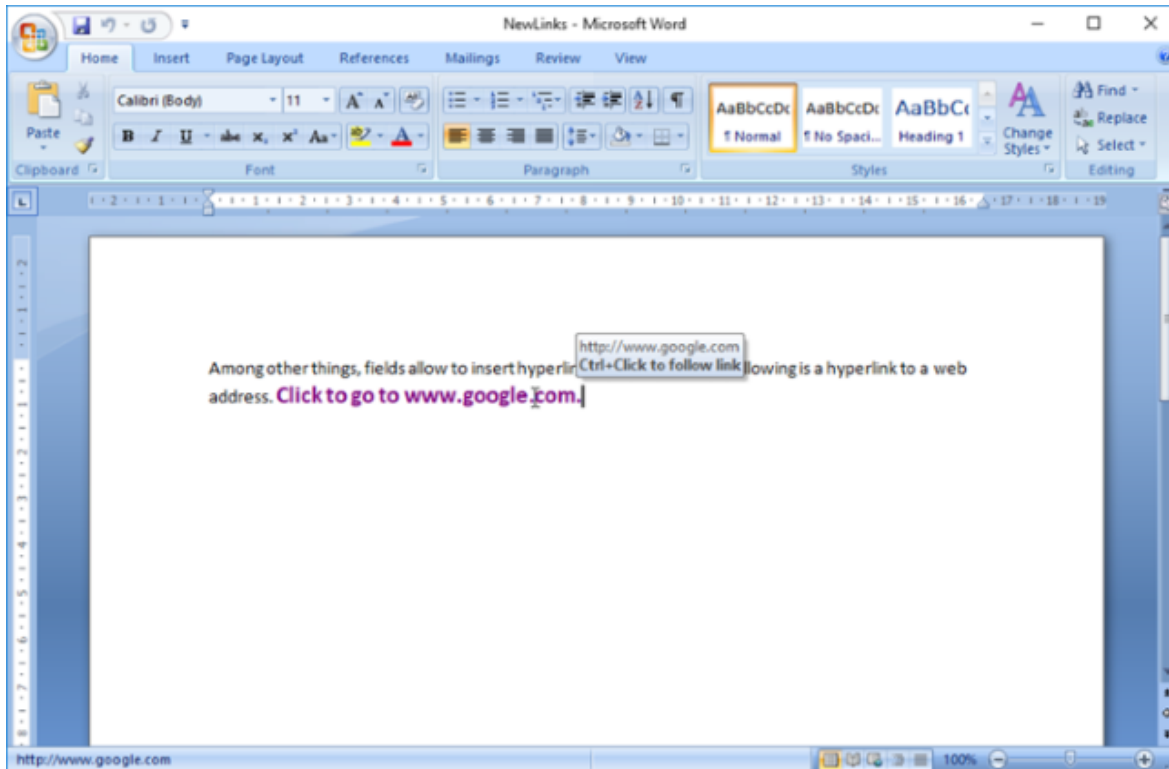
Links

Links are required to jump from one location to other location within the document or outside the document. In this section, we will discuss about the following types of links:

- [Hyperlink](#)
- [Bookmark](#)

Hyperlink

DsWord allows you to add, modify, and delete hyperlinks in a document. In DsWord, hyperlink element is represented by the `Hyperlink` class. You can add a hyperlink in a document using `Add` method of the `HyperlinkCollection` class. It can also be modified using the `Hyperlink` class properties, and deleted using `Delete` method of the `ContentObject` class.



Add Hyperlink

To add a hyperlink in a document:

1. Access a section in a document where the hyperlink is to be added.
2. Add a paragraph to the section using `Add` method of the `ParagraphCollection` class.
3. Add a hyperlink to the paragraph using `Add` method of the `HyperlinkCollection` class.

```
C#  
  
var section = doc.Body.Sections.First;  
  
//Add the first paragraph  
var p = section.GetRange().Paragraphs.Add(  
    "Among other things, fields allow to insert hyperlinks into documents." +  
    " Following is a hyperlink to a web address. ");  
  
//Add a hyperlink to it  
Hyperlink link1 = p.GetRange().Hyperlinks.Add(new Uri("http://www.google.com"),  
    "", "Click to go to www.google.com.");  
  
//Save the document  
doc.Save("AddHyperlink.docx");
```

Back to Top

Modify Hyperlink

To modify a hyperlink:

1. Access a hyperlink from the hyperlink collection using **Hyperlinks** property of the **RangeBase** class. For example, access the first hyperlink of the collection.
2. Modify the address for the specified link using **Address** property of the **Hyperlink** class.
3. Modify the text content value for the link using **Value** property of the **Text** class.

```
C#
//Load the existing word document
doc.Load("AddHyperlink.docx");

//Modify the hyperlink code
Hyperlink link1 =
    doc.Body.Sections.First.GetRange().Paragraphs.First.GetRange().Hyperlinks.First;
link1.Address = new Uri("http://developer.mescius.com");
link1.GetRange().Texts[0].Value = "Click to visit MESCIUS, Inc. website";

//Save the document
doc.Save("ModifyHyperlink.docx");
```

[Back to Top](#)

Delete Hyperlink

To delete a hyperlink:

1. Access a hyperlink from the hyperlink collection using **Hyperlinks** property of the **RangeBase** class. For example, access the first hyperlink of the collection.
2. Delete the field using the **Delete** method of the **ContentObject** class.

```
C#
//Load the existing word document
doc.Load("AddHyperlink.docx");

//Delete hyperlink to bookmark
doc.Body.Sections.First.GetRange().Paragraphs.First.GetRange().Hyperlinks.First.Delete();

//Save the document
doc.Save("DeleteHyperlink.docx");
```

[Back to Top](#)

Support for Malformed URI

To load a document containing malformed URI in DsWord, the URI needs to be rewritten in a valid manner. DsWord provides **MalformedUriRewriter** property in **GcWordDocument** class which encapsulates two delegates namely, **OnLoadInvalidUriHandler** and **OnSaveInvalidUriHandler** which are called on loading malformed URI event or on saving any URI from the document, respectively.

The **DefaultMalformedUriRewriter** class implements the **IMalformedUriRewriter** interface and provides the default rewriting strategy for handling invalid URIs.

You can also define any custom URI rewriter implementation which overrides the delegates and rewrites malformed URI as defined in the custom implementation.

The below template document contains malformed URI and is loaded in DsWord after using **MalformedUriRewriter** property to handle it. Refer to [Report Templates](#) to know more about templates.

Name: {{ds.name}}

Company: {{ds.co

mailto:{{ds.email}}

Ctrl+Click to follow link

Email: [[ds.email]]

C#

```
const string templateDocFileName = @"example_malformed_mailto.docx";

public void DemoDefaultRewriter()
{
    var doc = new GcWordDocument();
    try
    {
        Console.WriteLine("Loading file...");
        doc.Load(templateDocFileName);
    }
    catch (UriFormatException)
    {
        Console.WriteLine("File contains malformed urls and cannot be loaded without handling such urls.");
    }

    //apply malformed rewriter(default implementation).
    Console.WriteLine("Apply malformed rewriter (default implementation)...");

    doc.MalformedUriRewriter = new DefaultMalformedUriRewriter();

    Console.WriteLine("Loading file...");
    doc.Load(templateDocFileName);

    var contacts = new[]
    {
        new{ name = "x y",company = "A company", email = @"xy@a_company.com"},
        new{ name = "a b",company = "B company", email = @"ab@b.com"},
        new{ name = "c d",company = "C company", email = @"cd@C_company.com"},
    };

    doc.DataTemplate.DataSources.Add("ds", contacts);

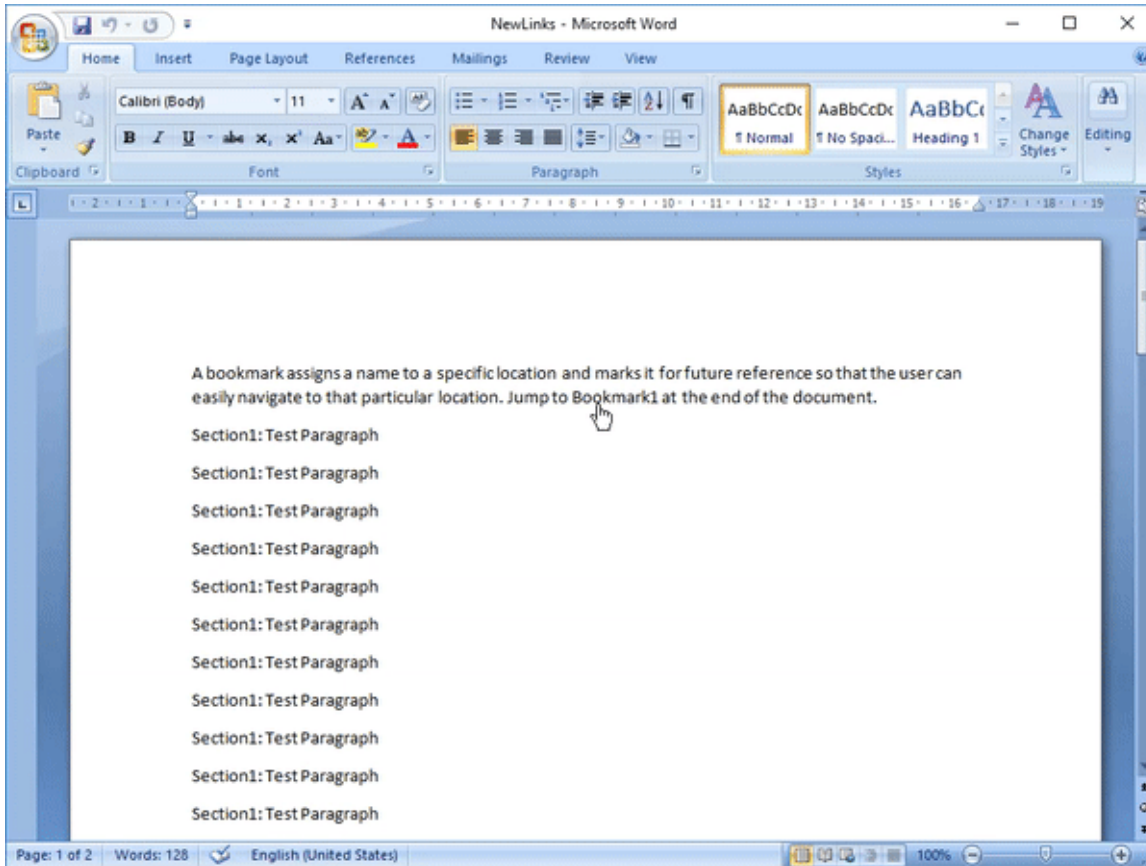
    Console.WriteLine("Filling template...");
    int i = 1;
    doc.DataTemplate.BatchProcess(() =>
    {
        var fileName = String.Format("gen{0}.docx", i++);
        Console.WriteLine(string.Format("Generated template file {0}", fileName));
        doc.Save(Path.Combine(fileName));
    });
}
```

Back to Top

For more information on how to implement hyperlinks using DsWord, see [DsWord sample browser](#).

Bookmark

A bookmark marks a specific location in a document for referencing and assigns a name to that location so that the user can easily navigate to that particular location. DsWord provides you [Bookmark](#) class which represents a bookmark range in the content. Using DsWord, you can add bookmarks in a document with [Add](#) method of the [BookmarkCollection](#) class and delete them using [Delete](#) method of the **Bookmark** class.



Add Bookmark

To add a bookmark:

1. Define a bookmark.
2. Add a paragraph to be marked by a bookmark.
3. Apply bookmark to the paragraph by adding a bookmark to the bookmark collection using **Add** method of the **BookmarkCollection** class.
4. Add a hyperlink that navigates to the bookmark location on clicking, using the [Add](#) method of the [HyperlinkCollection](#) class.

```
C#  
  
var section = doc.Body.Sections.First;  
  
// Add the first paragraph  
var p = section.GetRange().Paragraphs.Add("A bookmark marks a specific location" +  
    " in a document for referencing and assigns a name to that location so that" +  
    " the user can easily navigate to that particular location.");  
  
//Add a paragraph to the section  
for (var pr = 1; pr <= 30; pr++)
```

```
{
    section.GetRange().Paragraphs.Add("Section1: Test Paragraph");
}

//Define bookmark name
var bmk = "Bookmark1";

// Add a paragraph
var pb = section.GetRange().Paragraphs.Add($"{bmk} points here.");

//Mark the paragraph with a bookmark
pb.GetRange().Bookmarks.Add(bmk);

//Create hyperlink text to jump to the bookmark location(paragraph)
p.GetRange().Runs.Add("Jump to");
p.GetRange().Hyperlinks.Add(bmk, $"{bmk}");
p.GetRange().Runs.Add("at the end of the document.");

//Save the document
doc.Save("AddBookmark.docx");
```

[Back to Top](#)

Modify Bookmark

To modify a bookmark:

1. Access a bookmark to modify from the bookmarks collection using [Bookmarks](#) property of the [RangeBase](#) class. For example, access the first bookmark.
2. Modify the bookmark name using [Name](#) property of the **Bookmark** class.
3. Update the bookmark name in the hyperlink created for the bookmark.

```
C#
doc.Load("AddBookmark.docx");

//Modify the bookamrk name
var newBmk = "TestBookmark";
Bookmark bmark =
    doc.Body.Sections.First.GetRange().Paragraphs.Last.GetRange().Bookmarks.First;
bmark.Name = newBmk;

//Update the bookmark name in the hyperlink created for the bookmark
Hyperlink hyperlink_bookmark =
    doc.Body.Sections.First.GetRange().Paragraphs.First.GetRange().Hyperlinks.First;
hyperlink_bookmark.Anchor = newBmk;
hyperlink_bookmark.GetRange().Texts[0].Value = newBmk;

//Save the document
doc.Save("ModifyBookmark.docx");
```

[Back to Top](#)

Delete Bookmark

To delete a bookmark, access the bookmark from the bookmark collection using **Bookmarks** property of the **RangeBase** class and delete it using the **Delete** method.

C#

```
//Load the existing word document
doc.Load("AddBookmark.docx");

//Delete bookmark
Bookmark bmark =
    doc.Body.Sections.First.GetRange().Paragraphs.Last.GetRange().Bookmarks.First;
bmark.Delete();

//Delete hyperlink to bookmark
doc.Body.Sections.First.GetRange().Paragraphs.First.GetRange().Runs[1].Delete();
doc.Body.Sections.First.GetRange().Paragraphs.First.GetRange().Hyperlinks.First.Delete();
doc.Body.Sections.First.GetRange().Paragraphs.First.GetRange().Runs.Last.Delete();

//Save the document
doc.Save("DeleteBookmark.docx");
```

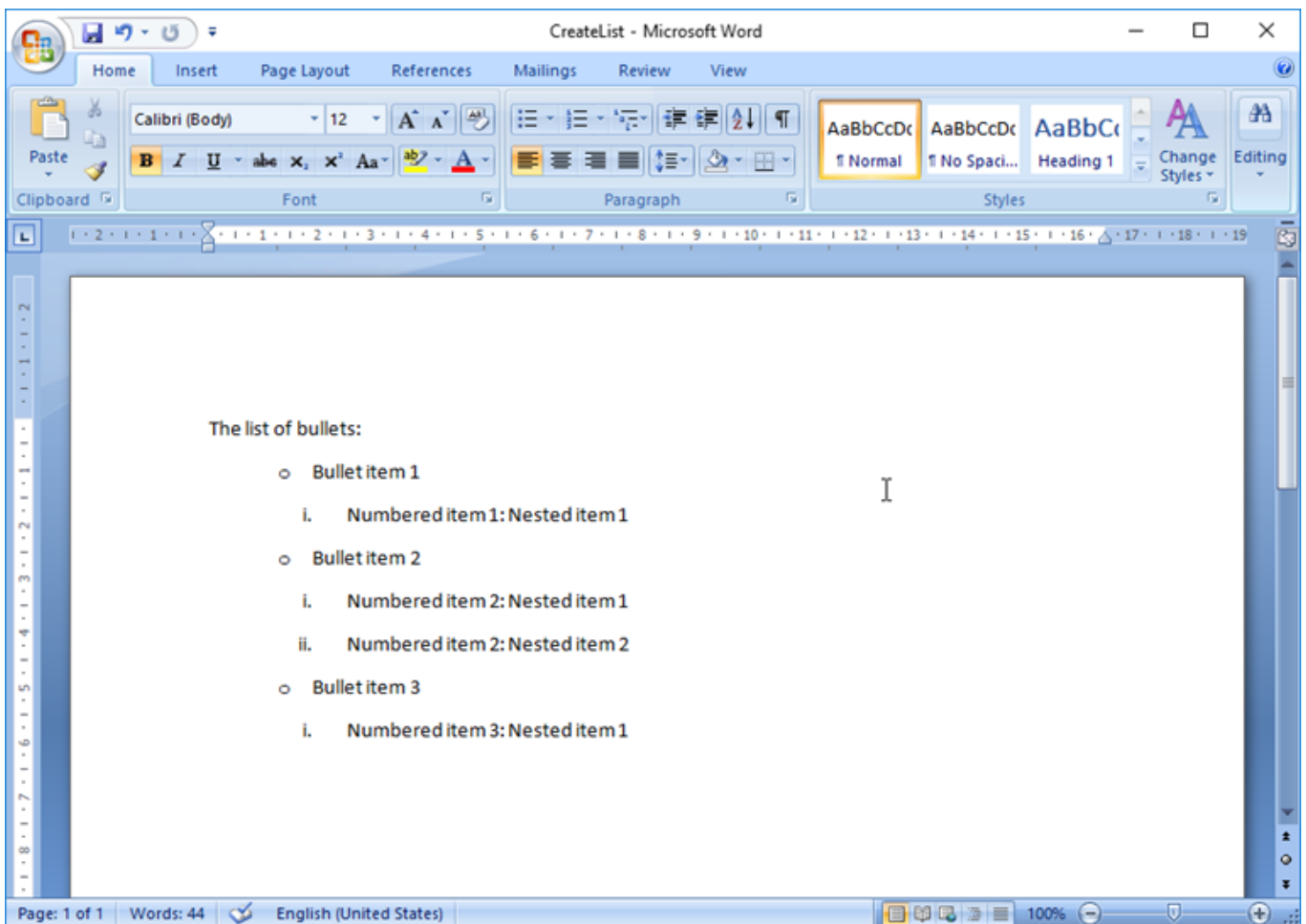
Back to Top

For more information on how to implement bookmarks using DsWord, see [DsWord sample browser](#).

Lists

Lists can be used to format and arrange text in an organized way. The appearance of lists can be customized by defining numbered, bulleted, and multilevel lists.

DsWord provides 21 built-in list templates to create lists in a document using the [ListTemplate](#) class. This class represents a list template that defines the format of list including the type of bullet or numbering style applied to the list. The [Add](#) method of the [ListTemplateCollection](#) class allows you to define and add list template to the collection of list templates in a document. The list templates are applied to a sequence of paragraphs to convert them to a list, where each paragraph serves as the list item. DsWord allows you to apply list formatting to the paragraphs using properties of the [ListFormat](#) class. It also provides [ListLevel](#) class which represents single list level for bulleted or numbered list. In addition, you can use [BuiltinListTemplateId](#) enumeration to use predefined list templates and [NumberStyle](#) enumeration to specify the number style for a list.



Create List

To create list in a Word document:

1. Add a paragraph using the [Add](#) method.
2. Access the list templates collection using [ListTemplates](#) property of the **DocumentBase** class.
3. Add a sequence of paragraphs which will serve as the list items using **Add** method of the **ListTemplateCollection** class.
4. Add paragraph part of the list using **Add** method of the paragraph collection for each item in the list.
5. Apply list template to all the paragraphs by using [ListFormat](#) property of the **Paragraph** class.

C#

```
//Define a ListTemplate for level 1 list items:
ListTemplate myListTemplate = doc.ListTemplates.Add(
    BuiltInListTemplateId.BulletDefault, "myListTemplate");

//Define list templates for level 2 list items
ListTemplate mySubList1Template = doc.ListTemplates.Add(
    BuiltInListTemplateId.NumberUppercaseRomanDot, "mySubList1Template");
ListTemplate mySubList2Template = doc.ListTemplates.Add(
    BuiltInListTemplateId.NumberUppercaseRomanDot, "mySubList2Template");
ListTemplate mySubList3Template = doc.ListTemplates.Add(
    BuiltInListTemplateId.NumberUppercaseRomanDot, "mySubList3Template");

ParagraphCollection pars = doc.Body.Sections.First.GetRange().Paragraphs;
pars.Add("The list of bullets:");

//Add list items using Paragraph and set their list template and level number:
Paragraph p1 = pars.Add("Bullet item 1");
p1.ListFormat.Template = myListTemplate;
p1.ListFormat.LevelNumber = 1;
Paragraph p1s1 = pars.Add("Numbered item 1: Nested item 1");
p1s1.ListFormat.Template = mySubList1Template;
p1s1.ListFormat.LevelNumber = 2;

Paragraph p2 = pars.Add("Bullet item 2");
p2.ListFormat.Template = myListTemplate;
p2.ListFormat.LevelNumber = 1;
Paragraph p2s1 = pars.Add("Numbered item 2: Nested item 1");
p2s1.ListFormat.Template = mySubList2Template;
p2s1.ListFormat.LevelNumber = 2;
Paragraph p2s2 = pars.Add("Numbered item 2: Nested item 2");
p2s2.ListFormat.Template = mySubList2Template;
p2s2.ListFormat.LevelNumber = 2;

Paragraph p3 = pars.Add("Bullet item 3");
p3.ListFormat.Template = myListTemplate;
p3.ListFormat.LevelNumber = 1;
Paragraph p3s1 = pars.Add("Numbered item 3: Nested item 1");
p3s1.ListFormat.Template = mySubList3Template;
p3s1.ListFormat.LevelNumber = 2;

//Save the document
doc.Save("CreateList.docx");
```

[Back to Top](#)

Modify List

To modify a list:

1. Access the list templates collection using **ListTemplates** property of the **DocumentBase** class.
2. Add a paragraph to the list using the **Add** method. This adds another item in the list.

3. Apply list template to all the paragraph using **ListFormat** property of the **Paragraph** class.
4. Set the list level for the new paragraph, if required. For example, set the level number to 2, so that it appears as a nested item in the list.

```
C#
doc.Load("List.docx");

//Access the list template collection
ListTemplate myListTemplate = doc.ListTemplates["myListTemplate"];

//Add a paragraph to the list
Paragraph p4=doc.Body.Sections.First.GetRange().Paragraphs.Add("Bullet item 4");
p4.ListFormat.Template = myListTemplate;

//Set the bullet level
p4.ListFormat.LevelNumber = 2;

//Save the document
doc.Save("ModifyList.docx");
```

[Back to Top](#)

Delete List

To delete a list, remove list formatting from the last paragraph of the document using [ClearFormatting](#) method of the [ListFormat](#) class.

```
C#
doc.Load("List.docx");

//Access the list template collection
ListTemplate myListTemplate = doc.ListTemplates["myListTemplate"];

//Remove list formatting from all the paragraphs serving as the list items
for (int p = 1; p < doc.Body.Paragraphs.Count; p++)
    doc.Body.Paragraphs[p].ListFormat.ClearFormatting();

//Save the document
doc.Save("DeleteList.docx");
```

[Back to Top](#)

Restart List

You can also restart a list by using the **RestartList** method of **Paragraph** class. The method provides an optional **startNumberingValue** parameter which defines the new numerical value to be used as the restart value in the list. If the list is alphabetical, this numerical value defines the index of the letter in the alphabet to be used as the restart value in the list. For example, if the numerical value is 8:

- A numbered list will restart from 8
- An alphabetical list will restart from 'h' (8th letter)

If **startNumberingValue** parameter is null, the list will restart from the first number or first letter, whichever is applicable.


The screenshot below shows a document (List.docx) that contains a multi-level list, and the same document after being updated using the following code:

Input	Output
A. A-1 B. B-1 C. C-1 a. Sub-a-1 b. Sub-b-1 c. Sub-c-1 d. Sub-d-1 D. D-1 E. E-1 F. F-1 a. Sub-a-2 b. Sub-b-2 c. Sub-c-2 d. Sub-d-2 G. G-1	A. A-1 B. B-1 C. C-1 a. Sub-a-1 h. Sub-b-1 i. Sub-c-1 j. Sub-d-1 D. D-1 E. E-1 F. F-1 a. Sub-a-2 b. Sub-b-2 c. Sub-c-2 d. Sub-d-2 G. G-1

C#

```
var doc = new GcWordDocument();
//Load document containing list
doc.Load("List.docx");
//Restart 5th paragraph of the list and switch its list element to 8th alphabet
letter (h)
doc.Body.Paragraphs[4].RestartList(8);

//Save the document
doc.Save("RestartList.docx");
```

 **Note:** The following behavior in DsWord differs from MS Word:

- DsWord allows you to restart last items of lists unlike MS Word.
- DsWord does not allow you to restart single-level lists as multilevel-lists unlike MS Word.

Back to Top

For more information on how to implement lists using DsWord, see [DsWord sample browser](#).

Paragraph

In DsWord, a paragraph is represented by the Paragraph class. This class allows you to work with the paragraph element in a Word document. DsWord provides ParagraphFormat class which contains all the formatting properties for a paragraph, such as alignment, indentation, shading, tab stops, etc.

- [Paragraph](#)
- [Indentation](#)
- [Line spacing](#)
- [Borders](#)
- [Background shading](#)
- [Tab stops](#)
- [Paragraph numbering](#)
- [Paragraph styling](#)

Paragraph

DsWord allows you to add, modify, and delete a paragraph from a Word document. It allows you to add a paragraph to the paragraph collection using [Add](#) method of the [ParagraphCollection](#) class, modify it using the [Paragraphs](#) property which accesses the paragraph collection, and delete it using the [Delete](#) method of the [ContentObject](#) class.

Add Paragraph

To add a paragraph to a document:

1. Access the paragraph collection using **Paragraphs** property of the [RangeBase](#) class.
2. Add a paragraph using **Add** method of the **ParagraphCollection** class.

```
C#  
  
//Add first paragraph  
doc.Body.Sections.First.GetRange().Paragraphs.Add("Hello, World!");  
  
//Add another paragraph  
doc.Body.Sections.First.GetRange().Paragraphs.Add("Document Solutions");  
  
//Save the document  
doc.Save("AddParagraph.docx");
```

[Back to Top](#)

Modify Paragraph

To modify a paragraph in a document:

1. Access the paragraph you want to modify from the paragraph collection using [Paragraphs](#) property of the **RangeBase** class. For example, access the first paragraph using [First](#) property of the [ParagraphCollection](#) class.
2. Modify the [Paragraph](#) class properties. For example, use the [Format](#) property for accessing the paragraph formatting properties such as the [Alignment](#) property to set the alignment of the paragraph.

```
C#  
  
doc.Load("AddParagraph.docx");  
//Modify settings of the first paragraph  
doc.Body.Sections.First.GetRange().Paragraphs.First.Format.Alignment =  
ParagraphAlignment.Center;  
  
//Save the document  
doc.Save("ModifyParagraph.docx");
```

[Back to Top](#)

Delete Paragraph

To delete a paragraph from a document, access a paragraph from the paragraph collection using the [Paragraphs](#) property and delete it using [Delete](#) method of the [ContentObject](#) class.

```
C#
```

```
doc.Load("AddParagraph.docx");  
//Delete the last paragraph  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Delete();  
  
//Save the document  
doc.Save("DeleteParagraph.docx");
```

Back to Top

For more information on how to work with paragraphs using DsWord, see [DsWord sample browser](#).

Indentation

DsWord allows you to specify the set of indentation properties for a paragraph using the [Indentation](#) class which can be accessed using [Indentation](#) property of the [ParagraphFormat](#) class.

Get Indents

To get the indents of a paragraph:

1. Access a paragraph from the paragraph collection using [Paragraphs](#) property of the [RangeBase](#) class.
2. Access the paragraph formatting properties using [Format](#) property of the [Paragraph](#) class.
3. Get the indentation properties applied to the paragraph using **Indentation** property of the **ParagraphFormat** class.
4. Get indentation property. For example, get the value of the first line indent of the paragraph using [FirstLineIndent](#) property of the **Indentation** class.
5. Display the indent value on the console.

```
C#  
  
//Get indents  
Console.WriteLine("Indent: " +  
  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Indentation.FirstLineIndent);
```

[Back to Top](#)

Set Indents

To set the indents of a paragraph:

1. Access a paragraph from the paragraph collection using **Paragraphs** property of the [RangeBase](#) class.
2. Access the paragraph formatting properties using **Format** property of the **Paragraph** class.
3. Get the indentation properties applied to the paragraph using **Indentation** property of the [ParagraphFormat](#) class.
4. Set indentation property. For example, set the value of the first line indent of the paragraph using **FirstLineIndent** property of the [Indentation](#) class.

```
C#  
  
//Set indents  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Indentation.FirstLineIndent =  
20;
```

[Back to Top](#)

For more information on how to implement paragraph indentation using DsWord, see [DsWord sample browser](#).

Line Spacing

DsWord allows you to specify line spacing for a paragraph using [LineSpacing](#) property of the [Spacing](#) class which can be accessed using [Spacing](#) property of the [ParagraphFormat](#) class.

Get Line Spacing

To get the line spacing of a paragraph:

1. Access a paragraph from the paragraph collection using [Paragraphs](#) property of the [RangeBase](#) class.
2. Access the paragraph formatting properties using [Format](#) property of the [Paragraph](#) class.
3. Get the spacing properties applied to the paragraph using **Spacing** property of the **ParagraphFormat** class.
4. Get the value of the line spacing for the paragraph using **LineSpacing** property of the **Spacing** class.
5. Display the line spacing value on the console.

```
C#
//Get line spacing
Console.WriteLine("Line spacing:" +
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Spacing.LineSpacing);
```

[Back to Top](#)

Set Line Spacing

To set the line spacing of a paragraph:

1. Access a paragraph from the paragraph collection using **Paragraphs** property of the **RangeBase** class.
2. Access the paragraph formatting properties using **Format** property of the **Paragraph** class.
3. Get the spacing properties applied to the paragraph using **Spacing** property of the [ParagraphFormat](#) class.
4. Set the line spacing rule using [LineSpacingRule](#) property of the [Spacing](#) class.
5. Set the value of the line spacing for the paragraph using **LineSpacing** property of the [Spacing](#) class.

```
C#
//Set line spacing
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Spacing.LineSpacingRule
=
    LineSpacingRule.Exact;
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Spacing.LineSpacing =
30;
```

[Back to Top](#)

For more information on how to implement line spacing using DsWord, see [DsWord sample browser](#).

Borders

DsWord allows you to specify the border properties for a paragraph using the [Border](#) class which can be accessed using [Borders](#) property of the [ParagraphFormat](#) class.

Get Borders

To get the borders of a paragraph:

1. Access a paragraph from the paragraph collection using [Paragraphs](#) property of the [RangeBase](#) class.
2. Access the paragraph formatting properties using [Format](#) property of the [Paragraph](#) class.
3. Get border collection of the paragraph using [Borders](#) property of the [ParagraphFormat](#) class.
4. Get the left and right borders using [Left](#) and [Right](#) properties of the [BorderCollection](#) class respectively.
5. Get the border style using [LineStyle](#) property of the [Border](#) class.
6. Get the border color using [Color](#) property of the [Border](#) class.
7. Display the left and right border style and color on the console.

```
C#  
  
//Get borders  
LineStyle styleLeft =  
  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Borders.Left.LineStyle;  
LineStyle styleRight =  
  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Borders.Right.LineStyle;  
Color colorLeft =  
  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Borders.Left.Color.RGB;  
Color colorRight =  
  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Borders.Right.Color.RGB;  
Console.WriteLine("Left Border: " + styleLeft + " " + colorLeft);  
Console.WriteLine("Right Border: " + styleRight + " " + colorRight);
```

[Back to Top](#)

Set Borders

To set the borders of a paragraph:

1. Access a paragraph from the paragraph collection using [Paragraphs](#) property of the [RangeBase](#) class.
2. Access the paragraph formatting properties using [Format](#) property of the [Paragraph](#) class.
3. Get border collection of the paragraph using [Borders](#) property of the [ParagraphFormat](#) class.
4. Get the left and right borders using [Left](#) and [Right](#) properties of the [BorderCollection](#) class respectively.
5. Set the border style using [LineStyle](#) property of the [Border](#) class.
6. Set the border color using [Color](#) property of the [Border](#) class.

```
C#  
  
//Set borders  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Borders.Left.Color.RGB =  
Color.Red;  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Borders.Left.LineStyle =  
LineStyle.ChainLink;  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Borders.Right.Color.RGB
```

```
= Color.Yellow;  
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Borders.Right.LineStyle  
=  
LineStyle.ChainLink;
```

Back to Top

For more information on how to implement paragraph borders using DsWord, see [DsWord sample browser](#).

Background Shading

DsWord allows you to specify the background shading for a paragraph using `BackgroundColor` property of the `Shading` class which can be accessed using `Shading` property of the `ParagraphFormat` class.

Get Shading

To get the shading:

1. Access a paragraph from the paragraph collection using `Paragraphs` property of the `RangeBase` class.
2. Access the paragraph formatting properties using `Format` property of the `Paragraph` class.
3. Access the shading formatting for the paragraph using `Shading` property of the `ParagraphFormat` class.
4. Get shading property. For example, get the shading texture using `Texture` property of the `Shading` class.
5. Get the background color using `BackgroundColor` property of the `Shading` class.
6. Display the shading pattern and color on the console.

```
C#
//Get shading
TexturePattern texture =
    doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Shading.Texture;
Color shading=

doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Shading.BackgroundColor.RGB;
Console.WriteLine("Shading Pattern: " + texture + "\n Shading Color: " + shading);
```

[Back to Top](#)

Set Shading

To set the shading:

1. Access a paragraph from the paragraph collection using `Paragraphs` property of the `RangeBase` class.
2. Access the paragraph formatting properties using `Format` property of the `Paragraph` class.
3. Access the shading formatting for the paragraph using `Shading` property of the `ParagraphFormat` class.
4. Set a shading property. For example, set the shading texture using `Texture` property of the `Shading` class.
5. Set the background color using `BackgroundColor` property of the `Shading` class.

```
C#
//Set shading
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Shading.Texture =
    TexturePattern.Percent15;
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.Shading.BackgroundColor.RGB
=
    Color.LightPink;
```

[Back to Top](#)

For more information on how to implement background shading using DsWord, see [DsWord sample browser](#).

Tab Stops

Tab stops are used in Word documents to enable text alignment by pressing the Tab key from the keyboard. There are generally five types of tab stops, namely, left, right, center, decimal, and bar tab.

DsWord allows you to specify tab stop properties for a paragraph using the `TabStop` class which can be accessed using `TabStops` property of the `ParagraphFormat` class. It supports all the five types of tab stops which can be set using `Alignment` property of the `TabStop` class. This property takes value from the `TabStopAlignment` enumeration.

Get Tab Stops

To get the tab stops:

1. Access a paragraph from the paragraph collection using `Paragraphs` property of the `RangeBase` class.
2. Access the paragraph formatting properties using `Format` property of the `Paragraph` class.
3. Get the tab stop properties applied to the paragraph using `TabStops` property of the `ParagraphFormat` class.
4. Get tab stop property. For example, get the position of the tab stop in points using `Position` property of the `TabStop` class.
5. Display the indent value on the console.

```
C#
//Get tab stops
Console.WriteLine("Tab Stop: "+
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.TabStops[0].Position);
```

[Back to Top](#)

Set Tab Stops

To set the tab stops:

1. Access a paragraph from the paragraph collection using `Paragraphs` property of the `RangeBase` class.
2. Access the paragraph formatting properties using `Format` property of the `Paragraph` class.
3. Get the tab stop properties applied to the paragraph using `TabStops` property of the `ParagraphFormat` class.
4. Add or replace the tab stop in the collection using `Add` method of the `TabStopCollection` class.

```
C#
//Set tab stops
doc.Body.Sections.First.GetRange().Paragraphs.Last.Format.TabStops.Add(20,
TabStopAlignment.Bar);
```

[Back to Top](#)

For more information on how to implement tab stops using DsWord, see [DsWord sample browser](#).

Paragraph Numbering

DsWord allows you to specify paragraph numbering using the [ListTemplate](#) class which can be accessed using [ListTemplates](#) property of the **DocumentBase** class.

Get Paragraph Numbering

To get the paragraph numbering:

1. Access a paragraph from the paragraph collection using [Paragraphs](#) property of the **RangeBase** class.
2. Get the list formatting specified for the paragraph using [ListFormat](#) property of the **Paragraph** class.
3. Access list template of the paragraph using [Template](#) property of the [ListFormat](#) class.
4. Get the name of the list template using [Name](#) property of the **ListTemplate** class.
5. Display the paragraph numbering on the console.

```
C#  
  
//Get paragraph numbering  
ParagraphCollection pars = doc.Body.Sections.First.GetRange().Paragraphs;  
Console.WriteLine("Numbering template for the first paragraph: " +  
    pars[0].ListFormat.Template.Name);
```

[Back to Top](#)

Set Paragraph Numbering

To set the paragraph numbering:

1. Access the collection of list templates using **ListTemplates** property of the GcWordDocument class.
2. Add a new list template to the collection of list templates using [Add](#) method of the [ListTemplateCollection](#) class.
3. Access a paragraph from the paragraph collection using **Paragraphs** property of the RangeBase class.
4. Get the list formatting specified for the paragraph using **ListFormat** property of the Paragraph class.
5. Set list template of the paragraph using **Template** property of the **ListFormat** class.

```
C#  
  
//Set paragraph numbering  
ParagraphCollection pars = doc.Body.Sections.First.GetRange().Paragraphs;  
// A ListTemplate is used to assign ListFormat/numbering to paragraphs  
ListTemplate myListTemplate = doc.ListTemplates.Add(  
    BuiltInListTemplateId.NumberArabicParenthesis, "myListTemplate");  
//Set the ListFormat for the paragraphs  
Paragraph p1 = pars[0];  
p1.ListFormat.Template = myListTemplate;  
Paragraph p2 = pars[1];  
p2.ListFormat.Template = myListTemplate;
```

[Back to Top](#)

For more information on how to implement paragraph numbering using DsWord, see [DsWord sample browser](#).

Paragraph Styling

DsWord allows you to define a style using [Style](#) class which can be accessed using [Styles](#) property of the **Run** class. It allows you to apply the defined style to a paragraph using [Style](#) property of the [Paragraph](#) class.

Set Paragraph Style

To set the paragraph style:

1. Define a new style using the **Style** class of type Paragraph using the [StyleType](#) enumeration and add it to the style collection using [Add](#) method of the [StyleCollection](#) class.
2. Get the character formatting of the paragraph using [Font](#) property of the Style class.
3. Set the character formatting using the [FontBase](#) class properties. For example, set the name of the font using the [Name](#) property and format the font as bold and italics using the [Bold](#) and [Italic](#) properties respectively.
4. Apply the defined style on the paragraph using [Style](#) property of the **Paragraph** class.

C#

```
//Add style to the first paragraph
Style sPara = doc.Styles.Add("ParaStyle", StyleType.Paragraph);
sPara.Font.Name = "Times New Roman";
sPara.Font.Bold = true;
sPara.Font.Italic = true;
doc.Body.Sections.First.GetRange().Paragraphs.Last.Style = sPara;
```

Back to Top

For more information on how to implement paragraph styling using DsWord, see [DsWord sample browser](#).

Range Objects

A Range object represents an arbitrary contiguous range of a document body. It is used to identify specific portions of a document. DsWord provides the [Range](#) class which represents a contiguous area of content objects in a document. This class is inherited from the [RangeBase](#) class which is the base class for range and body that allows manipulation of content in a document. The [RangeBase](#) class provides [GetPersistentRange](#) method which allows you to access the Range objects.

Access Range of Objects in a Document

To access a range of a content object in a document, use [GetPersistentRange](#) method of the RangeBase class. For example, access the range starting from the first paragraph to the end of the second paragraph in a document.

C#

```
//Load the existing word document
doc.Load("TestRange.docx");

//Get Range which starts from the first paragraph and ends at the second paragraph
Range paraRange = doc.Body.GetPersistentRange(doc.Body.Paragraphs.First,
doc.Body.Paragraphs[1]);
```

[Back to Top](#)

Access all Content within the Range Objects

To access all content within the range objects:

1. Get the range of a content object using [GetPersistentRange](#) method of the RangeBase class. For example, get the paragraph range starting from the first paragraph to the end of the second paragraph.
2. Display all the content objects within the accessed range on the console.

C#

```
//Load the existing word document
doc.Load("TestRange.docx");

//Get Range which starts from the first paragraph and ends at the second
paragraph
Range paraRange = doc.Body.GetPersistentRange(doc.Body.Paragraphs.First,
doc.Body.Paragraphs[1]);

//Get all the content objects in the range
Console.WriteLine("List of all the content objects contained in the range \n");
for (int i = 0; i < paraRange.Count; i++)
{
    Console.WriteLine(paraRange[i] + "\n");
}
```

[Back to Top](#)

Insert Objects Before or After Range Objects

To insert objects before or after range objects:

1. Get the range of a content object using **GetPersistentRange** method of the RangeBase class. For example, get the paragraph range starting from the first paragraph to the end of the second paragraph.
2. Insert an object before or after the range. For example, create and insert a table after the paragraph using **Insert** method of the **TableCollection** class.

```
C#
//Load the existing word document
doc.Load("TestRange.docx");

//Get Range which starts from the first paragraph and ends at the second
paragraph
Range paraRange = doc.Body.GetPersistentRange(doc.Body.Paragraphs.First,
doc.Body.Paragraphs[1]);

//Insert bookamrk before the range
paraRange.Bookmarks.Insert("Bookmark1", RangeLocation.Before);

//Insert table after the range
string[][] tableContent = new string[][] {
    new string[]{"0", "1", "2", "3"} ,
    new string[]{"4", "5", "6", "7"} ,
    new string[]{"8", "9", "10", "11"}
};
paraRange.Tables.Insert(tableContent, InsertLocation.After);

//Save the document with the changes
doc.Save("NewTestRange.docx");
```

Back to Top

Remove shapes and drawing objects from a document

To remove shapes and drawing objects from a document:

1. Access the unknown object collection from the document body using **Unknowns** property of the **RangeBase** class.
2. Access the shapes and delete them from the document using **Delete** method of the **ContentObject** class.

```
C#
//Load the existing word document
doc.Load("TestRange.docx");

//Get range which starts from the first paragraph and ends at the second
paragraph
Range pictureRange = doc.Body.GetPersistentRange(doc.Body.Pictures.First,
doc.Body.Paragraphs[1]);

//Delete the last picture in the range
pictureRange.Pictures.First.Delete();

//Access the unknown object collection
UnknownContentCollection unknowns = doc.Body.Unknowns;
```

```
Console.WriteLine("\n " + unknowns.Count.ToString());
for (int i = unknowns.Count - 1; i >= 0; i--)
{
    if (unknowns[i].OuterXml.ToString().Contains("Oval 1") ||
        unknowns[i].OuterXml.ToString().Contains("Isosceles Triangle 2"))

        //Delete the two shapes that exist in the document
        unknowns[i].Delete();
}

//Save the document with the changes
doc.Save("RemoveRange.docx");
```

Back to Top

Enumerate Content Objects in a Range

In DsWord, there are two types of range enumerators which differ on the basis of range of content objects:

- The inner collection enumerator includes only those content objects which start and/or end inside the range. These can be enumerated by using [GetInnerCollection](#) method of **RangeBase** class.
- The default range enumerator includes content objects lying in the inner collection range as well as those which start before the range and end after the range.

To enumerate content objects with default and inner collection enumerator:

C#

```
GcWordDocument doc = new GcWordDocument();
Paragraph para = doc.Body.Paragraphs.Add();
Run run = para.GetRange().Runs.Add("xxx");
Shape shape = run.GetRange().Shapes.Add();
TextFrame frame = shape.AddTextFrame();
Range frameRange = frame.GetRange();
frameRange.Paragraphs.Add("yyy");
//default enumerator
int count = frameRange.Runs.Count; // includes parent run "xxx" and "yyy" frame run
//inner collection enumerator
int innerCount = frameRange.GetInnerCollection<Run>().Count; // includes only "yyy"
frame run
```

To enumerate content objects in a parent-child relationship, you can use [GetChildren](#) method of **ContentObject** class. It returns the content objects directly related to a parent.

C#

```
var doc = new GcWordDocument();
var para = doc.Body.Paragraphs.Add("x");
var shape = para.GetRange().Runs.First().GetRange().Shapes.Add();
shape.AddTextFrame("textframe");
int count = para.GetRange().Runs.Count; // includes "x" and Shape run
int childCount = para.GetChildren<Run>().Count; // includes Shape run
```

Limitation

The **GetChildren<T>** method enumerates only ContentObjects and not ContentRanges or its derived classes such as Bookmark, Comment, EditableRange, ComplexField.

For more information on how to work with range objects using DsWord, see [DsWord sample browser](#).

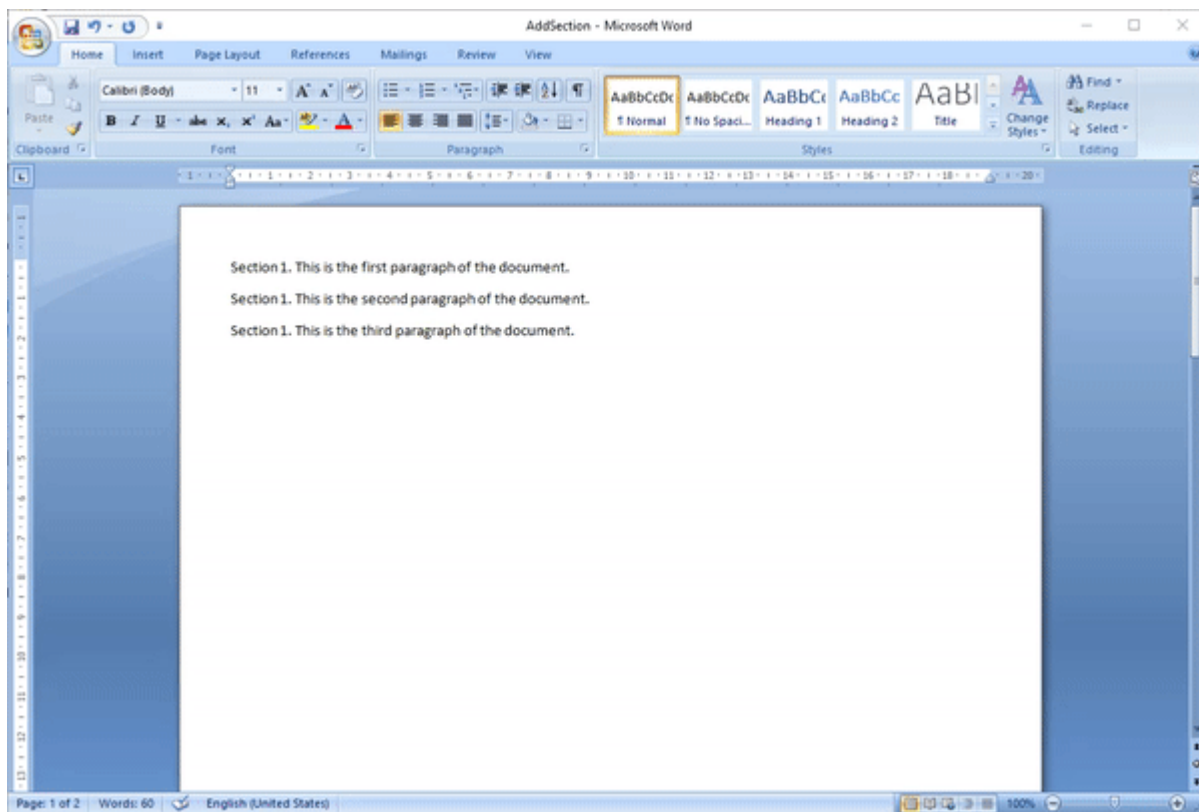
Sections

Sections in a word document allows you to insert, modify, remove sections and get and set section properties. A document can be divided into sections by inserting a section which creates a unique page layout for pages in that particular section. For instance, to create a portion of document with landscape orientation while leaving the rest as default portrait mode, insert a section before and after that landscape portion.

In DsWord, sections of the document are represented by the [Section](#) class which allows you to access section element in the body of a document. The Section objects are immediate children of the Document's body and can be accessed via the [Sections](#) property.

You can add sections to a document using [AddSectionBreak](#) method of the [Paragraph](#) class. This method accepts the section break type as a parameter, which can be passed using the [SectionStart](#) enumeration. The default value of this parameter is set to "NewPage", so invoking this method without a parameter inserts a section break of type new page in a document. However, the type of section break inserted can be altered by passing in appropriate value from the SectionStart enum. This enumeration has the following values:

- **Continuous**: Starts a new section on the same page.
- **NewPage**: Starts a new section from new page.
- **NewColumn**: Starts a new section from new column.
- **EvenPage**: Starts a new section from the next even page.
- **OddPage**: Starts a new section from the next odd page.



Insert Section

To insert a section in a document:

1. Create a new Word document using an instance of the [GcWordDocument](#) class.
2. Access the first section of the document.
3. Change the default page margins using [Margin](#) property of the **PageSetup** class.

4. Add paragraphs to the section using the **Add** method of **ParagraphCollection** class.
5. Add a new section in the document using **AddSectionBreak** method of the **Paragraph** class.
6. Set orientation of the new section using **Orientation** property and add paragraphs to it.

```
C#  
  
//Create a document  
GcWordDocument doc = new GcWordDocument();  
//Access the first section available by default  
Section sec1 = doc.Body.Sections.First;  
  
// Change default margins  
sec1.PageSetup.Margin.Left = sec1.PageSetup.Margin.Right =  
sec1.PageSetup.Margin.Top = sec1.PageSetup.Margin.Bottom = 36;  
  
//Add paragraphs to the first section  
ParagraphCollection pars1 = sec1.GetRange().Paragraphs;  
pars1.Add("Section 1. This is the first paragraph of the document.");  
pars1.Add("Section 1. This is the second paragraph of the document.");  
pars1.Add("Section 1. This is the third paragraph of the document.");  
  
//Add another section by inserting a section break  
Section sec2 = pars1.Last.AddSectionBreak();  
  
// Change page orientation for the second section  
sec2.PageSetup.Size.Orientation = PageOrientation.Landscape;  
  
//Add paragraphs in the second section  
ParagraphCollection pars2 = sec2.GetRange().Paragraphs;  
  
pars2.Add("Section 2. This is the first paragraph of second section.");  
pars2.Add("Section 2. This is the second paragraph of second section.");  
pars2.Add("Section 2. This is the third paragraph of second section.");  
  
//Save the document  
doc.Save("AddSection.docx");
```

[Back to Top](#)

Modify Section

To modify a section in a Word document:

1. Access the paragraph collection in a section.
2. Add a new paragraph to the section.
3. Set the paper size using **PageSetup** property of the **Section** class.

```
C#  
  
doc.Load("AddSection.docx");  
  
//Access the first section  
Section sec1 = doc.Body.Sections.First;  
  
//Access the paragraph collection in the first section
```



```
ParagraphCollection pars1 = sec1.GetRange().Paragraphs;

//Add a new paragraph to the first section
pars1.Add("Section 1. Adding fourth paragraph to the first section.");

//Set the paper size
sec1.PageSetup.Size.PaperSize = PaperSize.PaperA4;

//Save the document
doc.Save("ModifySection.docx");
```

[Back to Top](#)

Remove Section

To remove a section or section break from a document, you can use one of the methods given below:

Method	Description
MergeWithPrevious()	Removes the section break between two sections and the section content becomes the content of the previous section.
MergeWithNext()	Removes the section break between two sections and the section content becomes the content of the next section.
Delete()	Removes the section and its whole content.

1. Access the section in the document which needs to be removed.
2. Remove the section break by merging the section with the previous section using **MergeWithPrevious** method of the **Section** class.

```
C#
doc.Load("InsertSectionBreak.docx");

//Remove the section break
Section sec = doc.Body.Sections.Last;

//removing section break by merging a section with the previous one
sec.MergeWithPrevious();

doc.Save("RemoveSectionBreak.docx");
```

[Back to Top](#)

Get Section Properties

To get the properties of a section, you need to access a particular section. For example, access the last section of the document to get the direction of the text flow using the [TextFlowDirection](#) enumeration.

```
C#
SetSectionProperties();
doc.Load("SetSectionProperties.docx");
```

```
//Access the last section of the document
Section sec1 = doc.Body.Sections.Last;

//Get the direction of text flow
TextFlowDirection TypeA = (TextFlowDirection)Enum.Parse(typeof(TextFlowDirection),
    sec1.PageSetup.TextFlowDirection.ToString() );

//Write the fetched direction of text flow on the console
Console.WriteLine("TextFlowDirection: " + TypeA);
```

[Back to Top](#)

Set Section Properties

To set the section properties using the Section object:

1. Access a section to set its properties. For example, access the last section.
2. Set the section properties. For example, set the direction of text flow in the section and the paper size for last section of the document.

```
C#
doc.Load("AddSection.docx");

//Access the last section of the document
Section sec = doc.Body.Sections.Last;

//Set the direction of text flow
sec.PageSetup.TextFlowDirection = TextFlowDirection.TopToBottomRightToLeft;
//Set the size of the paper for the last section
sec.PageSetup.Size.PaperSize = PaperSize.PaperA4;

doc.Save("SetSectionProperties.docx");
```

[Back to Top](#)

For more information about how to implement sections using DsWord, see [DsWord sample browser](#).

Table

Tables in a document help in displaying the content in organized and structured manner. In DsWord, a table is a block content object and is represented by the [Table](#) class which lets you access the structure, content and formatting of the table.

In this section, you learn how to work with the following elements of the table:

- [Table](#)
- [Cell](#)
- [Row](#)

Table

DsWord allows you to add, modify, and delete a table from a Word document. A table can be created in a document using [Add](#) or [Insert](#) method of the [TableCollection](#) class. It also lets you format the table using the [TableFormat](#) class properties and apply styles to the table using [Style](#) class properties.

Adding a table:

Row 1, col 1	Row 1, col 2	Row 1, col 3	Row 1, col 4
Row 2, col 1	Row 2, col 2	Row 2, col 3	Row 2, col 4
Row 3, col 1	Row 3, col 2	Row 3, col 3	Row 3, col 4
Row 4, col 1	Row 4, col 2	Row 4, col 3	Row 4, col 4
Row 5, col 1	Row 5, col 2	Row 5, col 3	Row 5, col 4
Row 6, col 1	Row 6, col 2	Row 6, col 3	Row 6, col 4

Create Table

To create a table in Word document using DsWord:

1. Create a new Word document using an instance of the **GcWordDocument** class.
2. Get range of the first section in body of the document and add an empty table to the first section using the **Add** method.
3. Add rows and cells to the table using the **Add** method.
4. Create a new table style using the [Style](#) class and apply it on the table.
5. Save the document using the [Save](#) method.

C#

```
GcWordDocument doc = new GcWordDocument();
//Access the first section of the document
Section section = doc.Body.Sections.First;
//Add a paragraph in the section
Paragraph header = section.GetRange().Paragraphs.Add(" Adding a table:");

// Add an empty table with:
int rows = 6;
int cols = 4;
Table t = section.GetRange().Tables.Add(0, 0);
```

```
// Add some rows and cells to it:
List<string> cells = new List<string>(cols);
for (int col = 0; col < cols; ++col)
    cells.Add(string.Empty);
for (int row = 0; row < rows; ++row)
{
    for (int col = 0; col < cols; ++col)
        cells[col] = $"Row {row + 1}, col {col + 1}";
    t.Rows.Add(cells.ToArray());
}

// Create a new table style:
Style ts1 = doc.Styles.Add("Table Style 1", StyleType.Table);
foreach (var border in ts1.Table.Borders)
{
    border.LineStyle = LineStyle.BasicBlackDots;
    border.LineWidth = 0.5f;
    border.Color.RGB = Color.Purple;
}

//Apply the table style on the table
t.Style = ts1;

//Save the document
doc.Save("CreateTable.docx");
```

[Back to Top](#)

Modify Table

To modify the table formatting in a Word document:

1. Load the document created in **Create Table** section, using the [Load](#) method.
2. Access formatting properties of the table through [Format](#) property of the [Table](#) class.
3. Modify the table. For example, modify the alignment of the table using the [Alignment](#) property and set the [AllowAutoFit](#) property to allow the table cells to automatically resize according to their content.

```
C#
//Load an existing document
doc.Load("CreateTable.docx");

//Access the table
Table t = doc.Body.Sections.First.GetRange().Tables[0];

//Modify the table
t.Format.AllowAutoFit = true;
t.Format.Alignment = TableAlignment.Center;

//Save the document
doc.Save("ModifiedDoc.docx");
```

[Back to Top](#)

Delete Table

To delete a table from a Word document:

1. Access the table from the table collection using [Tables](#) property of the **RangeBase** class.
2. Delete the table using the [Delete](#) method.

```
C#  
  
//Load an existing document  
doc.Load("CreateTable.docx");  
  
//Access the table  
Table t = doc.Body.Sections.First.GetRange().Tables[0];  
  
//Delete the table  
t.Delete();  
  
//Save the document  
doc.Save("TableDeleted.docx");
```

[Back to Top](#)

AutoFit Table

To allow automatic resizing of cells in a table according to their content, use **AllowAutoFit** property of the [TableFormatBase](#) class.

```
C#  
  
//Load an existing document  
doc.Load("CreateTable.docx");  
  
//Access the table and allow automatic resizing of the cells  
Table t = doc.Body.Sections.First.GetRange().Tables[0];  
t.Format.AllowAutoFit = true;  
  
//Save the document  
doc.Save("AutoFitEnabled.docx");
```

[Back to Top](#)

Split Table

To split a table in a Word document:

1. Access a table and then access its row. For example, access the second row of the table.
2. Split the table from the specified row of an existing table.
3. Insert a paragraph between the two tables. This enables you to view two separate tables.

```
C#  
  
doc.Load("CreateTable.docx");  
  
//Access the table and it's second row  
Table t = doc.Body.Sections.First.GetRange().Tables[0];
```

```
Row row = t.Rows[1];

//This generates a new table from the specified row of an existing table.
Table splitTable1 = t.Split(row, InsertLocation.After);

//Insert a paragraph (without content) between the two tables
splitTable1.GetRange().Paragraphs.Insert(InsertLocation.Before);

//Save the document
doc.Save("SplitTable.docx");
```

[Back to Top](#)

Set Table Styles

To set table styles in a Word document:

1. Access the table from the table collection using **Tables** property of the **RangeBase** class.
2. Add a new table style to the style collection using **Add** method of **StyleCollection** class.
3. Apply the style on the table using **Style** property of **Table** class.
4. Set formatting properties of the table, such as Description, Alignment, and Title and save the document.

```
C#

//Load the document
doc.Load("CreateTable.docx");

//Access the table
Table t = doc.Body.Sections.First.GetRange().Tables[0];

// Create a new table style:
Style ts1 = doc.Styles.Add("Table Style 2", StyleType.Table);
ts1.Table.ColumnStripe=3;

// Assign the style to the table:
t.Style = ts1;

//Set table formatting
t.Format.Alignment = TableAlignment.Center;
t.Format.Title = "Test Table";
t.Format.Description = "This is the table description";

//Save the document
doc.Save("SetTableInfo.docx");
```

[Back to Top](#)

Get Table Styles

To get table styles and formatting:

1. Access the table from the table collection using **Tables** property of the **RangeBase** class.
2. Fetch the existing table styles using the **Style** property and the table formatting using the **Format** property.
3. Display the fetched details on the console.

```
C#
//Load the document
doc.Load("SetTableInfo.docx");

//Access the table
Table t = doc.Body.Sections.First.GetRange().Tables[0];

//Get table styles
Style st=t.Style;
uint cstripe = st.Table.ColumnStripe;
Console.WriteLine("Table stripe: " + cstripe.ToString());

//Get table formatting
TableFormat tformat = t.Format;
string title = tformat.Title;

//Write the fetched title of the table on the console
Console.WriteLine("Table Title: " + title);
```

[Back to Top](#)

Set Indents

To set the indentation of a table in Word document:

1. Add a new table style to the style collection using the **Add** method.
2. Set the indentation of the table, in points, using the **Indent** property
3. Apply the style on the table using the **Style** property.

```
C#
//Load the table
doc.Load("CreateTable.docx");

//Access the table and allow automatic resizing of the cells
Table t = doc.Body.Sections.First.GetRange().Tables[0];
t.Format.AllowAutoFit = true;

// Create a new table style to set the Indentation
Style ts1 = doc.Styles.Add("Table Style 2", StyleType.Table);

//Set the indentation
ts1.Table.Indent = 3;

//Apply the style
t.Style = ts1;

//Save the document
doc.Save("SetIndent.docx");
```

[Back to Top](#)

Get Indents

To get the indentation of a table from a Word document:

1. Access the table from the table collection using **Tables** property of the **RangeBase** class.
2. Fetch indentation of the table using **Indent** property.
3. Display the fetched table indentation on the console.

```
C#  
  
//Load an existing document  
doc.Load("SetIndent.docx");  
  
//Access the table  
Table t = doc.Body.Sections.First.GetRange().Tables[0];  
  
//Get the style applied on the table  
Style s = t.Style;  
  
//Get the indentation  
float indenting=s.Table.Indent;  
  
//Write the fetched indentation(in points) on the console  
Console.WriteLine("Get Indenting: " + indenting);
```

Back to Top

For more information about implementation of tables using DsWord, see [DsWord sample browser](#).

Cell

In DsWord, a cell element is represented by the `Cell` class which allows you to access each cell of a table which can be added through `Add` method of the `CellCollection` class.

Each element of a table, such as row, column, and cell, can have different styles and formatting. DsWord allows you to set the table cell formatting using `Format` property of the `Cell` class along with the `CellFormat` class properties. It also lets you merge cells, both vertically and horizontally, in a table. The `VerticalMerge` property allows a cell to merge vertically with the other cells in a table. Similarly, the `HorizontalMerge` property allows a cell to merge horizontally with the other cells in a table.

Set Cell Formatting

To set cell formatting in a table:

1. Access the table from the table collection using `Tables` property of the `RangeBase` class.
2. Set cell formatting. For example, set flow direction of the text and padding of third cell in the first row.

```
C#
//Load the document and access the table
doc.Load("CreateTable.docx");
Table t = doc.Body.Sections.First.GetRange().Tables[0];

//Set cell formatting
t.Rows[0].Cells[2].Format.TextFlowDirection =
TextFlowDirection.TopToBottomRightToLeft;
t.Rows[0].Cells[2].Format.Padding.Top = t.Rows[0].Cells[2].Format.Padding.Bottom
= 2;

//Save the document
doc.Save("SetCellFormatting.docx");
```

[Back to Top](#)

Get Cell Formatting

To get formatting of a cell from a table in an existing document:

1. Access the table from the table collection using `Tables` property of the `RangeBase` class.
2. Fetch the existing cell formatting using the `Format` property of the `Cell` class. For example, fetch the padding of third cell in the first row.

```
C#
//Load the document and access the table
doc.Load("SetCellFormatting.docx");
Table t = doc.Body.Sections.First.GetRange().Tables[0];

//Get cell padding of the third cell in the first row
CellFormat cformat = t.Rows[0].Cells[2].Format;
float paddingTop = cformat.Padding.Top;

//Write the fetched padding on the console
Console.WriteLine("Top Padding: " + paddingTop);
```

[Back to Top](#)

Merge Cells

To merge cells horizontally in a table:

1. Access the table from the table collection using **Tables** property of the **RangeBase** class.
2. Set the second cell of the second row to visually span two following cells, to merge the cells horizontally.
3. Remove the two following cells.

```
C#  
  
//Load the document and access the table  
doc.Load("CreateTable.docx");  
Table t = doc.Body.Sections.First.GetRange().Tables[0];  
  
// Set up cell (1,1) to visually span two following cells, hence merge the cells  
t.Rows[1].Cells[1].Format.GridSpan = 3;  
  
// Remove the 2 following cells (unless we want to create a jagged table)  
t.Rows[1].Cells[3].Delete();  
t.Rows[1].Cells[2].Delete();  
  
//Save the document  
doc.Save("MergedCells.docx");
```

[Back to Top](#)

Insert Graphic Elements

To insert a graphic element, such as an image, in a table cell:

1. Access a table in the document.
2. Load picture data into a byte array.
3. Access the cell where you want to add a picture and add the picture into the cell using **Add** method of the **PictureCollection** class. For example, access the last cell.
4. Set the picture size and set the **AllowAutoFit** property to true to allow automatic resizing of the cell.

```
C#  
  
doc.Load("CreateTable.docx");  
  
//Access the table  
Table t = doc.Body.Sections.First.GetRange().Tables[0];  
  
// Load picture data  
var picBytes = File.ReadAllBytes(Path.Combine("Resources", "Images",  
"road.jpg"));  
  
// Add picture, specifying its mime type:  
var pic = t.Rows.Last.Cells.Last.GetRange().Runs.First.GetRange().Pictures.Add(  
    picBytes, "image/jpeg");  
  
// Picture size
```

```
pic.Size.Width.Value = 200;
pic.Size.Height.Value = 100;

//Allow automatic resizing of the cell
t.Format.AllowAutoFit = true;

//Save the document
doc.Save("InsertGraphicElement.docx");
```

Back to Top

For more information about implementation of cells using DsWord, see [DsWord sample browser](#).

Row

DsWord provides the [Row](#) class representing a table row element. A row can be added to a table through [Add](#) method of the [RowCollection](#) class. DsWord provides you the access to the formatting properties of table row using [Format](#) property of the **Row** class along with the [RowFormat](#) class properties. In addition to formatting a row, DsWord allows you to get the row index, which can be done using [Index](#) property of the **Row** class.

Add Row

To add a new row to a table:

1. Access the table from the table collection using [Tables](#) property of the **RangeBase** class.
2. Add a row to the table using **Add** method of the **RowCollection** class.

```
C#  
  
//Load the document and access the table  
doc.Load("CreateTable.docx");  
Table t=doc.Body.Sections.First.GetRange().Tables[0];  
  
//Add a new row  
string[] newrow = new string[4] { "NR1", "NR2", "NR3", "NR4" };  
t.Rows.Add(newrow);  
  
//Save the document  
doc.Save("RowAdded.docx");
```

[Back to Top](#)

Delete Row

To delete a row from a table:

1. Access the table from the table collection using **Tables** property of the **RangeBase** class.
2. Delete a row of the table using the [Delete](#) method.

```
C#  
  
//Load the document and access the table  
doc.Load("CreateTable.docx");  
Table t = doc.Body.Sections.First.GetRange().Tables[0];  
  
//Delete the fourth row from the table  
t.Rows[3].Delete();  
  
//Save the document  
doc.Save("RowDeleted.docx");
```

[Back to Top](#)

Get Row Index

To fetch the row index from a table:

1. Access the table from the table collection using **Tables** property of the **RangeBase** class.
2. Get the row index of third row in the table using the **Index** property.
3. Display the row index on the console.

C#

```
//Load the document and access the table
doc.Load("CreateTable.docx");
Table t = doc.Body.Sections.First.GetRange().Tables[0];

//Get the row index
int i=t.Rows[2].Index;

//write the row index on the console
Console.WriteLine("Row Index for selected row is:" + i);
```

Back to Top

Format Rows

To format the rows in a table:

1. Access the table from the table collection using **Tables** property of the **RangeBase** class.
2. Set alignment for the table row using **Alignment** property of the **RowFormat** class.
3. Set spacing between the cells using **Spacing** property of the **RowFormat** class.

Back to Top

C#

```
//Load the document and access the table
doc.Load("CreateTable.docx");
Table t = doc.Body.Sections.First.GetRange().Tables[0];

//Set the alignment of a row and spacing between the cells
t.Rows[0].Format.Alignment = TableAlignment.Right;
t.Rows[0].Format.Spacing = 4F;

//Save the document
doc.Save("FormattedRow.docx");
```

Back to Top

For more information about implementation of rows using DsWord, see [DsWord sample browser](#).

Text

DsWord provides [Text](#) class to handle text element in the body of a document. It allows you to add text to a word document using [Add](#) method of the [TextCollection](#) class. By default, DsWord adds text to a document in a single column which covers the body of the document from one margin to other. However, you can change this formatting and create multiple columns using the [TextColumn](#) class. This class represents a text column and all the columns of text in a section are represented through the [TextColumnCollection](#) class.

Additionally, DsWord supports rendering text in multiple languages, including right-to-left, far eastern languages and vertical text with advanced paragraph formatting and multi-language font support. For right-to-left languages, you need to additionally set [Bidi](#) property of the [ParagraphFormat](#) class.

For more information on vertical text, see [Vertical text](#).

Add Text

To add text in a Word document:

1. Access a section where you want to add text using the [GetRange](#) method. For example, access the first run of a section.
2. Access the text collection using [Texts](#) property of the [RangeBase](#) class.
3. Insert text into the text collection using [Insert](#) method of the [TextCollection](#) class.

```
C#
doc.Load("SampleDoc.docx");

//Add Text
doc.Body.Sections.First.GetRange().Runs.First.GetRange().Texts.Insert(
    " Hello World! This document is created using DsWord. ",
    InsertLocation.End);

//Insert font formatting for the run
Style s = doc.Styles.Add("NewStyle", StyleType.Character);
s.Font.Name = "Times New Roman";
s.Font.Underline = Underline.Thick;
doc.Body.Sections.First.GetRange().Runs.First.Style = s;

//Add new run and text
doc.Body.Sections.First.GetRange().Paragraphs.First.GetRange().Runs.Insert("New
line " +
    "of the second run", InsertLocation.End);

//Save the document
doc.Save("AddText.docx");
```

[Back to Top](#)

Modify Text

To modify text in a Word document:

1. Access a run and the run text to modify text using the [Runs](#) and [Texts](#) property of the [RangeBase](#) class. For example, access the first run and the first run text.

- Split the run text using [Split](#) method of the **Text** class and split the run using [Split](#) method of the **Run** class respectively.
- Apply styling to the first and the second run. For example, italicize and bold the run text.

```
C#
doc.Load("AddText.docx");

//Access first run and change its font size
Run firstRun = doc.Body.Sections.First.GetRange().Runs.First;
firstRun.Font.Size = 16;

//Get first run text
Text firstRun_Text = firstRun.GetRange().Texts[0];

//Split first run text
Text splitText = firstRun_Text.Split(14);

//Split Run
Run splitRun = firstRun.Split(splitText, InsertLocation.Before);

//Apply styling to first run
firstRun_Text.ParentRun.Font.Bold = true;

//Apply styling to second/new run
splitRun.Font.Italic = true;

//Save the document
doc.Save("ModifyText.docx");
```

[Back to Top](#)

Delete Text

To delete text in a Word document:

- Access the section where you want to delete text using the **GetRange** method. For example, access the first run of a section.
- Access the text from the text collection using Texts property of the RangeBase class.
- Delete the text using [Delete](#) method of the [ContentObject](#) class.

```
C#
doc.Load("AddText.docx");

//Delete Text
doc.Body.Sections.First.GetRange().Runs.First.GetRange().Texts.First.Delete();

doc.Save("DeleteText.docx");
```

[Back to Top](#)

Align Text

To align text in a Word document:

1. Access a paragraph from the paragraph collection using the [Paragraphs](#) property.
2. Access a part of a paragraph from the run collection using the [Runs](#) property.
3. Set the alignment of text in the paragraph using [Alignment](#) property of the [ParagraphFormat](#) class. For example, center align the text.

```
C#
doc.Load("SampleDoc.docx");

//Create a new style
Style s = doc.Styles.Add("NewStyle", StyleType.Paragraph);
s.Font.Name = "Times New Roman";
s.Font.Underline = Underline.Thick;

//Center align text
s.ParagraphFormat.Alignment = ParagraphAlignment.Center;

//Apply style to the paragraph
Paragraph p = doc.Body.Sections.First.GetRange().Paragraphs.Add("Hello World!",
s);

//Add text to the new paragraph
p.GetRange().Runs.First.GetRange().Texts.Insert(" The text is center aligned.",
InsertLocation.End);

//Add new run and text to the first paragraph
doc.Body.Sections.First.GetRange().Paragraphs.First.GetRange().Runs.Insert(" It"
+
" includes an example of text alignment.", InsertLocation.End);

doc.Save("AlignText.docx");
```

[Back to Top](#)

Get Document Text

To get whole text of the document, use the [Body.Text](#) property.

```
C#
doc.Load("TestDoc.docx");

//Fetch text at document level
Console.WriteLine(doc.Body.Text);
```

[Back to Top](#)

Set Subscript and Superscript Text

DsWord provides [FontBase.VerticalPosition](#) property of the **SubSuperScript** class for applying subscript and superscript styles on particular characters in the text to be rendered in the Word document. This property accepts values from the [VerticalTextPosition](#) enumeration.

Subscript: C₆H₁₂O₆

Superscript: January 1st

To add subscript or superscript text:

1. Initialize the **GcWordDocument** class to create a Word document.
2. Define a style using the **Style** class, with the **StyleType** set to character. This ensures that the mentioned style options apply to the individual characters in the paragraph or text run to which the style is applied.
3. Set the **VerticalPosition** property of the defined character style to Subscript or Superscript, to allow the text to be repositioned as subscript or superscript.
4. Loop through the characters of the paragraph to be rendered, and identify the characters which should be rendered as superscript or subscript. Apply the style with subscript/superscript settings to the character by adding a new run to the paragraph being rendered.

C#

```
GcWordDocument doc = new GcWordDocument();

//Create subscript and superscript styles
Style subscriptStyle = doc.Styles.Add("Subscript Style", StyleType.Character);
subscriptStyle.Font.VerticalPosition = VerticalTextPosition.Subscript;
Style superscriptStyle = doc.Styles.Add("Superscript Style",
StyleType.Character);
superscriptStyle.Font.VerticalPosition = VerticalTextPosition.Superscript;

//Add a paragraph
var para1 = doc.Body.Sections.First.GetRange().Paragraphs.Add("Subscript: ");
string chemicalFormula = "C6H12O6";

//Add text to paragraph1
foreach(char c in chemicalFormula)
{
    //If the character is a digit render it with the subscript style
    if (Char.IsDigit(c))
        para1.GetRange().Runs.Add(c.ToString(), subscriptStyle);
    else
        para1.GetRange().Runs.Add(c.ToString());
}

//Add another paragraph
var para2= doc.Body.Sections.First.GetRange().Paragraphs.Add("Superscript: ");
string date="January 1st";
var dateFragments = Regex.Split(date, "(st)");

//Add text to the paragraph2
foreach (var frag in dateFragments)
{
    //Render the text "st" with superscript style
    if (frag=="st")
        para2.GetRange().Runs.Add(frag, superscriptStyle);
}
```

```
        else
            para2.GetRange().Runs.Add(frag);
    }

    //Saves the document
    doc.Save("SubSuperScript.docx");
```

For more information on how to render superscript and subscript text using DsWord, see [DsWord sample browser](#).

Fit Text Width

DsWord lets you fit a text into the width specified by **FontBase.FitTextWidth** property of the **FitTextWidth** class which resizes every character of the text equally to achieve the purpose.

To fit text into the specified width:

1. Initialize the **GcWordDocument** class to create a Word document.
2. Define a style using the **Style** class, with the **StyleType** set to character. This ensures that the mentioned style options apply to the individual characters in the paragraph or text run to which the style is applied.
3. Set the **FitTextWidth** property of the defined style to a desired value, to define the width in which the rendered text should fit.

C#

```
//Create new PDF document
GcWordDocument doc = new GcWordDocument();
Paragraph para =
    doc.Body.Sections.First.GetRange().Paragraphs.Add(
        "FitTextWidth examples: \r\n");
para.Style.Font.Size = 14;

//Define styles with FitTextWidth and FitTextId settings
Style run1Style = doc.Styles.Add("Style1", StyleType.Character);
run1Style.Font.Bold = true;
run1Style.Font.Size = 16;
run1Style.Font.FitTextWidth = 400;
run1Style.Font.FitTextId = 1;

Style run2Style = doc.Styles.Add("Style2", StyleType.Character);
run2Style.Font.Italic = true;
run2Style.Font.Size = 18;
run2Style.Font.FitTextWidth = 400;
run2Style.Font.FitTextId = 1;

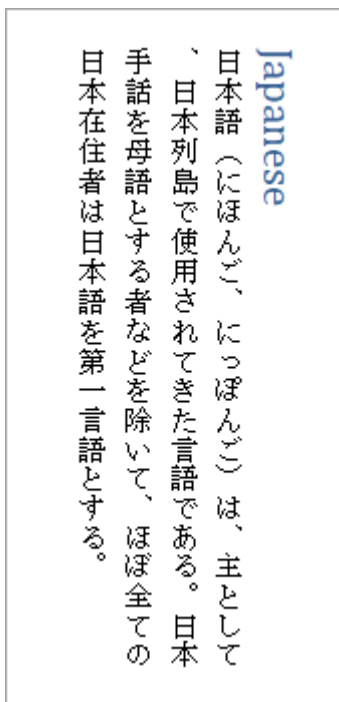
//Apply the defined styles to the text
para.GetRange().Runs.Add("This is run1 of paragraph1.", run1Style);
para.GetRange().Runs.Add(" This is run2 of paragraph1 " +
    "with a different formatting.", run2Style);

//Save the document
doc.Save("FitTextWidth.docx");
```

[Back to Top](#)

Vertical Text

DsWord supports rendering vertical text through `TextFlowDirection` property of the `PageSetup` class which accepts value from the `TextFlowDirection` enumeration. To set the vertical text alignment and reading order of the content in right to left direction, set this property to **TopToBottomRightToLeft**. This property draws the text in top-to-bottom, right-to-left layout. Additionally, DsWord supports rendering short upright Latin text or numbers in a block of vertical text referred to as 縦中横 (Tate Chu Yoko) in Japanese. You can set such blocks of horizontal text in vertical text through `HorizontalInVertical` property of the `EastAsianLayout` class.



C#

```
doc.Load("Sample.docx");

//Add heading
doc.Body.Sections.First.GetRange().Paragraphs.Add("Japanese",
    doc.Styles[BuiltInStyleId.Heading1]);

//Add a paragraph with Japanese text
string text = "日本語（にほんご、にっぽんご）は、主として、" +
    "日本列島で使用されてきた言語である。日本手話を母語とする者などを除いて、" +
    "ほぼ全ての日本在住者は日本語を第一言語とする。";

doc.Body.Sections.First.GetRange().Paragraphs.Add(text);

//Use TopToBottomRightToLeft page layout to draw the japanese text
doc.Body.Sections.First.PageSetup.TextFlowDirection =
    TextFlowDirection.TopToBottomRightToLeft;

//Save the document
doc.Save("VerticalText.docx");
```

[Back to Top](#)

Find and Replace Text

DsWord allows you to find and replace text in a document using the [Find](#) and [Replace](#) methods of [RangeBaseFindReplaceExtensions](#) class. The [RangeBaseFindReplaceExtensions](#) class extends [RangeBase](#) class and hence, its methods can be used on any range.

The search pattern for the **Find** and **Replace** methods can be any static text, [regex](#) or special token. The supported special tokens are:

- Paragraph end - &p
- Manual break object - &l
- Page break - &m
- Section break - &b

Find Text

The **Find** method finds all the occurrences of a user-defined search pattern in a document. It also provides various find options using which you can search backwards, ignore case, define culture, use regular expressions or get formatting options

To find text in a document:

1. Specify the search pattern which needs to be found in a document.
2. Load the document using **Load** method of **GcWordDocument** class.
3. Find the specified search pattern using the **Find** method of **RangeBaseFindReplaceExtensions** class.
4. Get the count of total occurrences of the search pattern by using the **Count** function.

```
C#
// find pattern contains static text and special token
const string findPattern = "javascript framework&p";

// Load the document
var doc = new GcWordDocument();
doc.Load("JsFrameworkExcerpt.docx");

// Find all occurrences of the search string, ignoring the case
var finds = doc.Body.Find(findPattern, new FindOptions(doc) { IgnoreCase = true
});

// Print the number of found occurrences at the top of the document
doc.Body.Paragraphs.Insert($"Found {finds.Count()} occurrences of
'{findPattern}' in the document.\n\n", InsertLocation.Start);

// Done
doc.Save("FindText.docx");
```

Back to Top

Replace Text

The [Replace](#) method finds and replaces all the occurrences of a user-defined search pattern in a document with the text to be replaced. The [ReplaceAction](#) enum specifies what action should be taken on finding a match, like replacing, skipping the current match or stopping the search.

To find and replace text in a document using regex and special tokens:

1. Create a new Word document by instantiating the **GcWordDocument** class.
2. Add paragraphs using **Add** method of the **ParagraphCollection** class.
3. Find and replace the specified search pattern (regex and special token) with a static string using **Replace** method of **RangeBaseFindReplaceExtensions** class.


```
C#
var doc = new GcWordDocument();

// Add paragraph
doc.Body.Paragraphs.Add("Document Solutions is a cross-platform solution for
document management1");
doc.Body.Paragraphs.Add("which aims to provide a universal document, editor and
viewer solution for all popular document formats.");

//Replace using regex and special token
RangeBaseFindReplaceExtensions.Replace(doc.Body, "[0-9]&p", ". It");

doc.Save("RemoveDigits.docx");
```

Back to Top

 **Note:** A search pattern can also be removed from a document by passing the replaceText string as empty.

If replacing text leads to any empty runs in a document, you can choose to delete those empty runs by using [RemoveEmptyRuns](#) property of **FindReplaceOptions** class.

To replace text with empty text and remove empty runs in a document:

1. Create a new Word document by instantiating the **GcWordDocument** class.
2. Add a paragraph using **Add** method of the **ParagraphCollection** class.
3. Add runs to the paragraph using **Add** method of **RunCollection** class
4. Invoke the **Replace** method of the **RangeBaseFindReplaceExtensions** class to replace second and third runs with empty text and set **RemoveEmptyRuns** property to true.

```
C#
var doc = new GcWordDocument();
var para = doc.Body.Paragraphs.Add();
para.GetRange().Runs.Add("first run");
para.GetRange().Runs.Add(" second run");
para.GetRange().Runs.Add(" third run");
para.GetRange().Runs.Add(" fourth run");

Console.WriteLine(String.Format(@"Current paragraph text is ""{0}"" and runs
count = {1}", para.GetRange().Text, para.GetRange().Runs.Count));
Console.WriteLine("Removed text from second and third runs");
//Replace second and third runs with empty text and remove empty runs
para.GetRange().Replace(" second run third run", "", new FindReplaceOptions(doc)
{ RemoveEmptyRuns = true });
Console.WriteLine(String.Format(@"Current paragraph text is ""{0}"" and runs
count = {1}", para.GetRange().Text, para.GetRange().Runs.Count));
return doc;
```

Replace Text using Regex and Special Tokens

In case, each match is to be replaced by a different value (like numerals to textual representation), the [FindReplaceOptions](#) class provides callback functions for the same.

To find and replace digits to their textual representation in a document using callback method:

1. Create a new Word document by instantiating the **GcWordDocument** class.
2. Add a paragraph using **Add** method of the **ParagraphCollection** class.
3. Invoke the **Replace** method of the **RangeBaseFindReplaceExtensions** class to perform the replace operation.
4. Pass the [ReplacingCallback](#) function with a callback method name as one of the parameters to it. This method is used to determine the replacement string.

```
C#  
  
/// Replace all digits with their textual representation  
/// <returns>Document without digits</returns>  
public void ReplaceDigitsWithText()  
{  
    var doc = new GcWordDocument();  
    var para = doc.Body.Paragraphs.Add("The game board was yellow with numbers;  
2 digits, a space, and 3 digits. All the digits 0 to 9 were placed same width  
apart");  
  
    doc.Body.Replace("[0-9]", "", new FindReplaceOptions(doc) {  
        ReplacingCallback = new ReplaceDigitsWithTextCb().Replacing });  
    doc.Save("NumeralsToTextualDigits.docx");  
}  
  
private class ReplaceDigitsWithTextCb  
{  
    Dictionary<string, string> _numbers = new Dictionary<string, string>()  
        {{ "1", "one"}, {"2", "two"}, {"3", "three"}, {"4", "four"}, {"5", "five"},  
{"6", "six"}, {"7", "seven" },  
        { "8", "eight"}, {"9", "nine"}, {"0", "zero"} };  
  
    //Callback method called on every found match. There we can examine found  
match  
    //and change replacement string  
    public ReplaceAction Replacing(ReplacingArgs args)  
    {  
        args.Replacement =  
_numbers.Keys.Contains(args.RegexDetails.Capture.Value) ?  
_numbers[args.RegexDetails.Capture.Value] : "_";  
        return ReplaceAction.Replace;  
    }  
}
```

Back to Top

Replace Formatting

The find and replace operations can also be performed based on the formatting rules like any font, color or style etc.

To find and replace green text in a document:

1. Create a new Word document by instantiating the **GcWordDocument** class.
2. Add a paragraph using **Add** method of the **ParagraphCollection** class.
3. Add runs to the paragraph using **Add** method of **RunCollection** class.

4. Apply red and green color to the runs separately by using the **Color** property of **FontBase** class.
5. Create an instance of **FindReplaceOptions** class and configure its **FormattingOptions** property to search for text in the document which is green in color.
6. Replace the green text with an empty string by using the **Replace** method of **RangeBaseFindReplaceExtensions** class.

```
C#
//create document
var doc = new GcWordDocument();
//create paragraph
var para = doc.Body.Paragraphs.Add();
//add green text to paragraph
var run_green = para.GetRange().Runs.Add("Document Solutions for Word library is
a part of Document Solutions that " +
    "aims to be a complete solution to program and work with Word documents.");
//add red text to paragraph
var run_red = para.GetRange().Runs.Add("It offers a rich and comprehensive
object model " +
    "which is simple to use and is based on Microsoft Office API, Word
Javascript API and OpenXML SDK.");

//color red text as red
run_red.Font.Color.RGB = Color.Red;
//color green text as green
run_green.Font.Color.RGB = Color.Green;

var fo = new FindReplaceOptions(doc);
//find green text
fo.FormattingOptions.Font.Color.RGB = Color.Green;

//replace any found green text to empty string
doc.Body.Replace("", "", fo);

//save result document
doc.Save("RemovingTextinGreenColor.docx");
```

Back to Top

For more information on how to work with text objects using DsWord, see [DsWord sample browser](#).

Themes

Themes, in a Word document, helps in making the global formatting changes, giving your document a consistent and professional look. Hence, themes are generally used to reinforce a brand across every document that is generated by a business.

DsWord allows you to customize the themes applied to a Word document through the **Theme** class which can be accessed using **Theme** property of the **GcWordDocument** class. The **Theme** class saves the font and colors that would be used in the document using the **FontScheme** and **ColorScheme** properties.

The theme font can be set using the **FontScheme** property of the Theme class. This property is of type **FontScheme** class which provides the **MajorFont** and **MinorFont** properties to specify the font for heading and body respectively.

And, to set the theme colors you would need to fetch the theme color scheme using the **GetThemeColor** method of the **Settings** class which accepts the **ThemeColorId** as a parameter. This method returns an instance of the **ThemeColor** class, which provides the **RGB** property to set the value of a color in the color scheme of the theme.

To change the theme color and font in a Word document:

1. Access the color scheme of the document using **GetThemeColor** method of the **Settings** class.
2. Set the value of a color in the color scheme of the theme using the RGB property of the ThemeColor class. For example, set dark orange color for Accent1 theme color and dark blue color for the hyperlink theme color.
3. Access the font scheme for the document using **FontScheme** property of the Theme class.
4. Set the theme font for heading and body of the document using the **Name** property of the **ThemeFont** class.

C#

```
GcWordDocument doc = new GcWordDocument();
var section = doc.Body.Sections.First;

//Add Heading
section.GetRange().Paragraphs.Add("Working with Themes",
    doc.Styles[BuiltInStyleId.Heading1]);

//Add the first paragraph
var p = section.GetRange().Paragraphs.Add(
    "This sample shows how you can access and manipulate" +
    " the document theme properties." +
    " For more information, refer to ");

//Add a hyperlink
Hyperlink link1 = p.GetRange().Hyperlinks.Add(
    new Uri("https://www.google.co.in/"),
    "", "Google");

//Customizing the color scheme(theme colors) of the document.
//Setting the Accent1 theme color
doc.Settings.GetThemeColor[ThemeColorId.Accent1].RGB = Color.DarkOrange;

//Setting the Hyperlink theme color
doc.Settings.GetThemeColor[ThemeColorId.Hyperlink].RGB = Color.DarkBlue;

//Setting the FollowedHyperlink theme color
doc.Settings.GetThemeColor[ThemeColorId.FollowedHyperlink].RGB = Color.DarkGray;
```

```
//Customizing the font scheme(theme fonts) of the document
//Setting Major(Heading) theme font for Latin characters
doc.Theme.FontScheme.MajorFont.Latin.Name = "Calibri";

//Setting Minor(Body) theme font for Latin characters
doc.Theme.FontScheme.MinorFont.Latin.Name = "Century Gothic";

//Saves the document
doc.Save("Themes.docx");
```

For more information on how to modify theme colors using DsWord, see [DsWord sample browser](#).

Helper Methods for Adding Content

DsWord provides various Add methods in the classes to add different content elements directly, thus making the code shorter, clearer, and the content elements easily accessible. The following table lists the various Add methods added to the classes:

Class	Method	Description
Body	AddTable	Adds a Table to the end of the body.
	AddParagraph	Adds a Paragraph to the end of the body.
	AddContentControl	Adds a ContentControl to the end of the body.
	CanAdd	Gets whether a ContentObject type can be added to the end of the body.
	CanAddContentControl	Gets whether a ContentControl with a specified type can be added to the end of the body.
Section	AddTable	Adds a Table to the end of the section.
	AddParagraph	Adds a Paragraph to the end of the section.
	AddContentControl	Adds a ContentControl to the end of the section.
	CanAdd	Gets whether a ContentObject type can be added to the end of the section.
	CanAddContentControl	Gets whether a ContentControl with a specified type can be added to the end of the section.
Table	AddContentControl	Adds a ContentControl to the end of the table.
Row	AddContentControl	Adds a ContentControl to the end of the row.
Cell	AddTable	Adds a Table to the end of the cell.
	AddParagraph	Adds a Paragraph to the end of the cell.
	AddContentControl	Adds a ContentControl to the end of the cell.
Paragraph	AddSimpleField	Adds a SimpleField to the end of the paragraph.
	AddHyperlink	Adds a Hyperlink to the end of the paragraph.
	AddBidirectionalOverride	Adds a BidirectionalOverride to the end of the paragraph.
	AddOMathParagraph	Adds an OMathParagraph to the end of the paragraph.
	AddOMath	Adds an OMath to the end of the paragraph.
	AddRun	Adds a Run to the end of the paragraph.
	AddFootnote	Adds a Footnote to the end of the paragraph.
	AddEndnote	Adds an Endnote to the end of the paragraph.
	AddComplexField	Adds a ComplexField to the end of the paragraph.
	AddContentControl	Adds a ContentControl to the end of the paragraph.
	AddSectionBreak	Breaks the parent section right after this paragraph.

SimpleField	AddSimpleField	Adds a SimpleField to the end of the simple field.
	AddHyperlink	Adds a Hyperlink to the end of the simple field.
	AddBidirectionalOverride	Adds a BidirectionalOverride to the end of the simple field.
	AddOMathParagraph	Adds an OMathParagraph to the end of the simple field.
	AddOMath	Adds an OMath to the end of the simple field.
	AddRun	Adds a Run to the end of the simple field.
	AddFootnote	Adds a Footnote to the end of the simple field.
	AddEndnote	Adds an Endnote to the end of the simple field.
	AddComplexField	Adds a ComplexField to the end of the simple field.
	AddContentControl	Adds a ContentControl to the end of the simple field.
Hyperlink	AddSimpleField	Adds a SimpleField to the end of the hyperlink.
	AddHyperlink	Adds a Hyperlink to the end of the hyperlink.
	AddBidirectionalOverride	Adds a BidirectionalOverride to the end of the hyperlink.
	AddOMathParagraph	Adds an OMathParagraph to the end of the hyperlink.
	AddOMath	Adds an OMath to the end of the hyperlink.
	AddRun	Adds a Run to the end of the hyperlink.
	AddFootnote	Adds a Footnote to the end of the hyperlink.
	AddEndnote	Adds an Endnote to the end of the hyperlink.
	AddComplexField	Adds a ComplexField to the end of the hyperlink.
	AddContentControl	Adds a ContentControl to the end of the hyperlink.
BidirectionalOverride	AddSimpleField	Adds a SimpleField to the end of the bidirectional override.
	AddHyperlink	Adds a Hyperlink to the end of the bidirectional override.
	AddBidirectionalOverride	Adds a BidirectionalOverride to the end of the bidirectional override.
	AddOMathParagraph	Adds an OMathParagraph to the end of the bidirectional override.
	AddOMath	Adds an OMath to the end of the bidirectional override.
	AddRun	Adds a Run to the end of the bidirectional override.
	AddFootnote	Adds a Footnote to the end of the bidirectional override.
	AddEndnote	Adds an Endnote to the end of the bidirectional override.
	AddComplexField	Adds a ComplexField to the end of the bidirectional override.
	AddContentControl	Adds a ContentControl to the end of the bidirectional override.
OMathParagraph	AddOMath	Adds an OMath to the end of the Office Math paragraph.
	AddRun	Adds a Run to the end of the Office Math paragraph.
	AddFootnote	Adds a Footnote to the end of the Office Math paragraph.
	AddEndnote	Adds an Endnote to the end of the Office Math paragraph.

	AddComplexField	Adds a ComplexField to the end of the Office Math paragraph.
OMath	AddSimpleField	Adds a SimpleField to the end of the Office Math zone.
	AddHyperlink	Adds a Hyperlink to the end of the Office Math zone.
	AddRun	Adds a Run to the end of the Office Math zone.
	AddFootnote	Adds a Footnote to the end of the Office Math zone.
	AddEndnote	Adds an Endnote to the end of the Office Math zone.
	AddComplexField	Adds a ComplexField to the end of the Office Math zone.
	AddOMathStruct	Adds a new OMathStruct to the end of the Office Math zone.
	AddContentControl	Adds a ContentControl to the end of the Office Math zone.
	AddAccent	Adds a new OMathAccent to the end of the Office Math zone.
	AddBar	Adds a new OMathBar to the end of the Office Math zone.
	AddBoundingBox	Adds a new OMathBoundingBox to the end of the Office Math zone.
	AddBox	Adds a new OMathBox to the end of the Office Math zone.
	AddDelimiter	Adds a new OMathDelimiter to the end of the Office Math zone.
	AddEquationArray	Adds a new OMathEquationArray to the end of the Office Math zone.
	AddFraction	Adds a new OMathFraction to the end of the Office Math zone.
	AddFunction	Adds a new OMathFunction to the end of the Office Math zone.
	AddGroupCharacter	Adds a new OMathGroupCharacter to the end of the Office Math zone.
	AddLimitLower	Adds a new OMathLimitLower to the end of the Office Math zone.
	AddLimitUpper	Adds a new OMathLimitUpper to the end of the Office Math zone.
	AddMatrix	Adds a new OMathMatrix to the end of the Office Math zone.
	AddNary	Adds a new OMathNary to the end of the Office Math zone.
	AddOMathStruct	Adds a new OMathStruct to the end of the Office Math zone.
	AddPhantom	Adds a new OMathPhantom to the end of the Office Math zone.
AddPreSubSuperscript	Adds a new OMathPreSubSuperscript to the end of the Office Math zone.	
AddRadical	Adds a new OMathRadical to the end of the Office Math zone.	
AddSubscript	Adds a new OMathSubscript to the end of the Office Math zone.	
AddSubSuperscript	Adds a new OMathSubSuperscript to the end of the Office Math zone.	
AddSuperscript	Adds a new OMathSuperscript to the end of the Office Math zone.	
OMathElement	AddSimpleField	Adds a SimpleField to the end of the Office Math element.
	AddHyperlink	Adds a Hyperlink to the end of the Office Math element.
	AddRun	Adds a Run to the end of the Office Math element.

	AddFootnote	Adds a Footnote to the end of the Office Math element.
	AddEndnote	Adds an Endnote to the end of the Office Math element.
	AddComplexField	Adds a ComplexField to the end of the Office Math element.
	AddOMathStruct	Adds a new OMathStruct to the end of the Office Math element.
	AddContentControl	Adds a ContentControl to the end of the Office Math element.
	AddAccent	Adds a new OMathAccent to the end of the Office Math element.
	AddBar	Adds a new OMathBar to the end of the Office Math element.
	AddBoundingBox	Adds a new OMathBoundingBox to the end of the Office Math element.
	AddBox	Adds a new OMathBox to the end of the Office Math element.
	AddDelimiter	Adds a new OMathDelimiter to the end of the Office Math element.
	AddEquationArray	Adds a new OMathEquationArray to the end of the Office Math element.
	AddFraction	Adds a new OMathFraction to the end of the Office Math element.
	AddFunction	Adds a new OMathFunction to the end of the Office Math element.
	AddGroupCharacter	Adds a new OMathGroupCharacter to the end of the Office Math element.
	AddLimitLower	Adds a new OMathLimitLower to the end of the Office Math element.
	AddLimitUpper	Adds a new OMathLimitUpper to the end of the Office Math element.
	AddMatrix	Adds a new OMathMatrix to the end of the Office Math element.
	AddNary	Adds a new OMathNary to the end of the Office Math element.
	AddOMathStruct	Adds a new OMathStruct to the end of the Office Math element.
	AddPhantom	Adds a new OMathPhantom to the end of the Office Math element.
	AddPreSubSuperscript	Adds a new OMathPreSubSuperscript to the end of the Office Math element.
	AddRadical	Adds a new OMathRadical to the end of the Office Math element.
	AddSubscript	Adds a new OMathSubscript to the end of the Office Math element.
	AddSubSuperscript	Adds a new OMathSubSuperscript to the end of the Office Math element.
	AddSuperscript	Adds a new OMathSuperscript to the end of the Office Math element.
Run	AddGroupShape	Adds a GroupShape to the end of the run.
	AddCanvasShape	Adds a CanvasShape to the end of the run.

	AddShape	Adds a new GeometryType.Rectangle 100 x 100 points Shape to the end of the run.
	AddPicture	Adds a new Picture to the end of the run.
	AddInkShape	Adds an InkShape to the end of the run.
	AddText	Adds a Text to the end of the run.
	AddTab	Adds a Tab to the end of the run.
	AddBreak	Adds a Break to the end of the run.
	AddSymbol	Adds a Symbol to the end of the run.
GroupShape	AddGroupShape	Adds a GroupShape to the end of the group shape.
	AddShape	Adds a new GeometryType.Rectangle 100 x 100 points Shape to the end of the group shape.
	AddPicture	Adds a new Picture to the end of the group shape.
	AddInkShape	Adds an InkShape to the end of the group shape.
CanvasShape	AddGroupShape	Adds a GroupShape to the end of the canvas shape.
	AddShape	Adds a new GeometryType.Rectangle 100 x 100 points Shape to the end of the canvas shape.
	AddPicture	Adds a new Picture to the end of the canvas shape.
	AddInkShape	Adds an InkShape to the end of the canvas shape.
TextFrame	AddTable	Adds a Table to the end of the text frame.
	AddParagraph	Adds a Paragraph to the end of the text frame.
	AddContentControl	Adds a ContentControl to the end of the text frame.
ControlContent	AddTable	Adds a Table to the end of the control content.
	AddParagraph	Adds a Paragraph to the end of the control content.
	AddSimpleField	Adds a SimpleField to the end of the control content.
	AddHyperlink	Adds a Hyperlink to the end of the control content.
	AddBidirectionalOverride	Adds a BidirectionalOverride to the end of the control content.
	AddOMathParagraph	Adds an OMathParagraph to the end of the control content.
	AddOMath	Adds an OMath to the end of the control content.
	AddRun	Adds a Run to the end of the control content.
	AddFootnote	Adds a Footnote to the end of the control content.
	AddEndnote	Adds an Endnote to the end of the control content.
	AddComplexField	Adds a ComplexField to the end of the control content.
	AddContentControl	Adds a ContentControl to the end of the control content.
ContentObject	CanAdd	Gets whether a ContentObject type can be added to this content object.
	CanAddContentControl	Gets whether a ContentControl type can be added to this content

object.

Refer to the following example code to add run and hyperlink directly to the paragraph:

```
C#
// Initialize GcWordDocument.
GcWordDocument doc = new GcWordDocument();

// Add paragraph with default formatted text.
var p = doc.Body.AddParagraph("AddRun and AddHyperlink methods will add the run and
hyperlink directly to the paragraph. ");

// Add another text and a hyperlink into the paragraph formatted with "Heading1"
style.
/* Earlier, the code would have looked like this:
    p.GetRange().Runs.Add("Follow this link to open Google. ",
doc.Styles[BuiltInStyleId.Heading1]);
    Hyperlink link1 = p.GetRange().Hyperlinks.Add(new Uri("http://www.google.com"), "",
"Click here.");
    Now the code is shorter and more clear about what content can be added to the
object. */
p.AddRun("Follow this link to open Google. ", doc.Styles[BuiltInStyleId.Heading1]);
p.AddHyperlink(new Uri("http://www.google.com"), "", "Click here.");

// Save the Word document.
doc.Save("HelperMethods.docx", DocumentType.Document);
```

Refer to the following example code to check if a paragraph can be added after content control using CanAdd method:

```
C#
// Initialize GcWordDocument.
var doc = new GcWordDocument();

// Add a paragraph.
var p = doc.Body.Paragraphs.Add();


// Add content control.
var cc = p.AddContentControl(ContentControlType.RichText).Content;

// Check if a paragraph can be added or not and add a paragraph; otherwise, add a
run.
if (cc.CanAdd(typeof(Paragraph))) // Returns false since the content control was
created on inline level.
    cc.AddParagraph("rich text");
else
    cc.AddRun("rich text");

// Save the document.
doc.Save("CanAdd.docx");
```


Report Templates

Every organization needs to maintain important documents such as financials, legal contracts and agreements, resources and logistics, recruitments and various others. Such documents contain similar type of static content with dynamic values like dates, costs, names, numbers etc. Creating and modifying these documents every single time consumes a considerable amount of time and effort. Also, the chances of manual error cannot be eliminated altogether. Therefore, the need for Data or Report Templates.

 **Note:** Please note that 'Report Templates' and 'Data Templates' are two names for the same feature and can be used interchangeably.

The template layout can be created in Word which can contain static content as well as placeholder fields for dynamic values. When the template is processed by using DsWord API, it is bound to a data source which populates the placeholder fields with actual data and generates the final Word document with advanced layouts.

Lets take a simple example where a placeholder field is defined in mustache braces {{ }} in a template layout:

Hello {{ds.name}} <After Template processing> Hello John

Here, ds is the name of data source and John is the value in 'name' data field.

Some of the powerful features provided by DsWord Report Templates are as follows:

- **Flexible:** Highly flexible template syntax and API to bind Word documents to data from data source. It supports advanced formatting and layout options.
- **Document Automation:** Automates the process of generating structured documents with increased accuracy.
- **Efficient:** Provide extended reusability, meaning that the templates can be used with minor modifications, or as it is time and again, saving both time and effort.
- **Multi-platform:** Supported on Windows, Linux and macOS.

The DsWord template layout is a pre-defined and formatted Word document. It is used as an input to create the resultant Word document with required data from data source. In the following sections, you will find DsWord Report Templates being used in diverse use-cases.

The below examples use short notation for template tags. To know more about short notations, refer [Template Tags](#).

- **Use Case 1 - Rental Agreement**

In this use case, we have created a template layout in Word for House Rental Agreement which contains placeholder fields for all the dynamic values like, the landlord's, tenant's and rental information. The value of placeholder fields is populated by the data source upon template processing. Notice that the currency and date formatters are also used for relevant fields.

You can also download the [Template layout](#) used in below example.

House Rental Agreement

Landlord Information		Tenant Information	
Name {{landlordName}}		Name {{tenantName}}	
Address {{landlordAddrLine1}}, {{landlordAddrLine2}} {{landlordAddrCity}}, {{landlordAddrState}}, {{landlordAddrPostal}} United States		Email {{tenantEmail}}	
Phone Number {{landlordPhone}}		Phone Number {{tenantPhone}}	
		Number of Occupants {{tenantOccupants}}	
Rental Information			
House Rental Address {{rentalAddrLine1}}, {{rentalAddrLine2}}, {{rentalAddrCity}}, {{rentalAddrState}}, {{rentalAddrPostal}}, United States	Rent Amount per Month	{{rentalAmntPerMonth}:format(C)}	
Start Date of Agreement {{rentalStart}:format(D)}	Security Deposit	{{rentalSecurityDep}:format(C)}	
End Date of Agreement {{rentalEnd}:format(D)}	Late Charges	{{rentalLateCharge}:format(C)}	
Date of First Payment Due {{rentalFirstPayment}:format(D)}	Payment Method	{{paymentMethod}}	
Pay Period: {{rentalPayPeriod}}	Collected by	{{collectedBy}}	

The Rental Agreement generated after template processing from the above Template is given below:

House Rental Agreement

Landlord Information		Tenant Information	
Name	Jillene Holtaway	Name	Nadine Rousell
Address	725 Barby Junction, 85 Grayhawk Center San Antonio, Texas, 78220 United States	Email	aibrahim@example.com
Phone Number	(192) 472-5503	Phone Number	(123) 456-7890
		Number of Occupants	2
Rental Information			
House Rental Address	321 Ken Junction, 12 Blueowl Plaza, San Antonio, Texas, 33322, United States	Rent Amount per Month	\$3,200.00
Start Date of Agreement	Thursday, December 31, 2020	Security Deposit	\$5,000.00
End Date of Agreement	Saturday, December 31, 2022	Late Charges	\$767.00
Date of First Payment Due	Friday, January 1, 2021	Payment Method	Option 2
Pay Period:	1 month	Collected by	Nadine Roussel

- Use Case 2 - Consulting Agreement**

In this template, a Consulting Agreement has been set up between the Consultant and Client. The generic content of the agreement is kept static whereas the dynamic content like consultant and client details, dates, scope of work is incorporated as placeholder fields.

You can also download the [Template layout](#) used in below example.

Consulting Agreement

THIS AGREEMENT (the "Agreement"), is entered into on this date `{{dateEntered}}` by and between:

`{{consFirmName}}` a registered company located at `{{consFirmAddrLine1}}`, `{{consFirmAddrLine2}}`, `{{consFirmAddrCity}}`, `{{consFirmAddrState}}`, `{{consFirmAddrPostal}}` (hereby known as the "Consultant"), and;

`{{clientName}}`, whose address is at `{{clientAddrLine1}}`, `{{clientAddrLine2}}`, `{{clientAddrCity}}`, `{{clientAddrState}}`, `{{clientAddrPostal}}` (hereby known as the "Client");

WHEREAS, the Client desires to hire the services of the Company to render services to `{{scopeOfWork}}`;

NOW, THEREFORE, for and in consideration of the mutual covenants made by the parties hereto, the Parties to this agreement agree as follows:

The Services

The Consultant agrees that it shall render services to the Client on matters pertaining to `{{scopeOfWork}}` (the "Services").

Compensation and Payment

The Client shall provide payment to the Consultant at the amount of `{{paymentAmount}}` paid on every `{{paymentTerm}}`, beginning the next month of the commencement of this Agreement.

`{{retainerAmount}}:hbi-equals(0):format(Retainer)}`

A retainer of `{{retainerAmount}}:hbi-equals(0):format(C)` (the "Retainer") is payable by the Client upon execution of this Agreement.

The Consulting Agreement generated after template processing from the above Template is given below:

Consulting Agreement

THIS AGREEMENT (the "Agreement"), is entered into on this date 11/22/20 10:16:00 PM by and between:

Aliquam erat volutpat a registered company located at **1 Oak Drive, Tennyson Plaza, Frederick, Maryland, 21705** (hereby known as the "Consultant"), and;

Overtop Software, whose address is at **135 Borland Alley, Pie close, Gotharm, New Jersey, 35123** (hereby known as the "Client");

WHEREAS, the Client desires to hire the services of the Company to render services to **Vivamus metus arcu, adipiscing molestie, hendrerit at, vulputate vitae, nisl. Aenean lectus**;

NOW, THEREFORE, for and in consideration of the mutual covenants made by the parties hereto, the Parties to this agreement agree as follows:

The Services

The Consultant agrees that it shall render services to the Client on matters pertaining to **Vivamus metus arcu, adipiscing molestie, hendrerit at, vulputate vitae, nisl. Aenean lectus** (the "Services").

Compensation and Payment

The Client shall provide payment to the Consultant at the amount of **5** paid on every **2 weeks**, beginning the next month of the commencement of this Agreement.

Retainer

A retainer of \$500.00 (the "Retainer") is payable by the Client upon execution of this Agreement.

- **Use Case 3 - Lease Agreement**

This use case depicts a template for Lease Agreement between Lessee and Lessor as two parties. The placeholder fields are defined for unique information about the parties which is replaced by actual data from data source after template processing.

You can also download the [Template layout](#) used in below example.

Lease Agreement

This Lease Agreement between **{{Lessor}}** (Lessor) and **{{Lessee}}** (Lessee) was executed on **{{ExecDate}:tolongdatestring()}**.

For informational purposes, here are the contact details of both parties:

Lessee Information

{{Lessee}} **{{LesseePhone}}**
{{LesseeEmail}}

Lessor Information

{{Lessor}}
{{LessorEmail}}

{{LessorAddrLine1}}, **{{LessorAddrCity}}**, **{{LessorAddrState}}**, United States
{{LessorPhone}}

PREMISE

The Condominium is located at **{{PremAddrLine1}}**, **{{PremAddrCity}}**, **{{PremAddrState}}**, **{{PremAddrPostal}}**.

The Lessee will use the condominium for residential living purposes only.

The Lessee and Lessor agree that in the time of this agreement, the condominium is in good condition.

TERM

This Agreement states that the lease term will start on **{{TermStart}}** and will end on **{{TermEnd}}**.

For renewals, a new agreement or contract is required and needs to be signed.

PAYMENTS

The monthly payment costs **\${{MonthlyFee}}**.

Payments will be made via **{{PaymentMethod}}**.

The person who will receive the payment is **{{Lessor}}** (Lessor).

For late payments which is 5 days after the due date, there'll be an additional charge of **\${{LateFee}}**.

The Lease Agreement generated after template processing from the above Template is given below:

Lease Agreement

This Lease Agreement between Jillene Holtaway (Lessor) and Jane Donahue (Lessee) was executed on Sunday, November 22, 2020.

For informational purposes, here are the contact details of both parties:

Lessee Information

Jane Donahue (123)098-7654
janed@example.com

Lessor Information

Jillene Holtaway
jilleneh@example.com

123 Main St., Gotham, New Jersey, United States
(123)456-7890

PREMISE

The Condominium is located at 39628 Golf Pass, Saint Louis, Missouri, 63169.

The Lessee will use the condominium for residential living purposes only.

The Lessee and Lessor agree that in the time of this agreement, the condominium is in good condition.

TERM

This Agreement states that the lease term will start on 12/13/20 10:16:00 PM and will end on 12/13/22 10:16:00 PM.

For renewals, a new agreement or contract is required and needs to be signed.

PAYMENTS

The monthly payment costs \$4000.

Payments will be made via Credit Card.

The person who will receive the payment is Jillene Holtaway (Lessor).

For late payments which is 5 days after the due date, there'll be an additional charge of \$8000.

- **Use Case 4 - Address Labels**

In this example, we have created a template layout for address label in Word which contains placeholder fields for dynamic values like, name, address, state and phone number. These address labels need to be created for various people of an organization. Hence, the address label template is batch processed so that a separate document is created for each data source item.

You can also download the [Template layout](#) used in below example.

Name: {{name}}

Address: {{address}}

State: {{state}}

Phone No: {{contact}}

The Address labels generated after batch processing of template are given below:

Name: John Smith

Address: 9843 Elementum St.

State: Missouri

Phone No: (963) 356-9268

Name: Jasper Carney

Address: 9631 Semper Ave

State: California

Phone No: (906) 217-1470

Name: Mark Jordan

Address: 1195 Lobortis Rd.

State: Washington

Phone No: (763) 409-5446

Template Configuration

The template layout can be configured to include various formatting features like bulleted lists, tables, hyperlinks, conditional formatting etc. This provides a well defined structure and design to the final Word documents.

The data bound fields follow the specified syntax and are defined in mustache braces {{ }} in the template layout. These fields can also include formatters like date, currency, count, uppercase etc. which are applied in the final Word document when populated with data from datasource.

A template layout can be configured to include the following:

- [Template Tags](#)
- [List](#)
- [Table](#)
- [Links](#)
- [Loop and Nesting](#)
- [Conditionally Hide Blocks](#)
- [Formatters](#)

Template Tags

Template tags are defined in the template layout which define placeholders, range blocks, formatting options etc. and generate desired Word documents after template processing.

Value Tag

The value tags specify placeholders which are replaced by actual data from the data source after template processing. These can be defined by using the below syntax:

```
{{ds.value_path[:formatter1(args1):formatter2(args2)..formatterN(argsN)]}}
```

- ds - User-defined name of data source
- value_path - Full dot-delimited path from data source to the property name
- formatterX - (Optional) One or more formatters that process the value, Refer [Formatters](#) for more details

Example 1: `{{ds.cities.name}:toupper()}}`

Here ds is the name of datasource and cities.name is full value path in data source. The 'toupper' formatter converts all strings to their upper case representation. After processing, the template tag expands automatically and produces a list of every city in every object.

Result:

```
NEW YORK  
LOS ANGELES  
CHICAGO  
HOUSTON  
SAN FRANCISCO
```

Example 2: You can also use multiple formatters in a single tag by chaining them together. In such cases, the result of one formatter serves as an input to the next formatter. The below value tag hides all records where UnitPrice is less than 30 and formats the resultant values as currency.

```
{{ds.UnitPrice}:hide-block-if-less(30):format(C)}
```

Short Notation for Value Tag

As a single data source can be used in a template, you can also specify the value tag in a short format by omitting the name of datasource. For example:

- `{{cities.name}:toupper()}}`
- `{{UnitPrice}:hbi-less(30):format(C)}`

Range Tag

Range tags are used when you want to define a strict area for repeating a section of data object. The content in this section is copied for every enumerated object. If omitted, the template automatically calculates the start and end of a repeating block and may produce unwanted results.


Open and close tags can be used to resolve the start and end of a repeating section of data.


If 'ds' is the name of a data set:

- `{{#ds}}` or `{{#ds.enumerable_path}}` - Start of a range block
- `{{/ds}}` or `{{/ds.enumerable_path}}` - End of a range block

For example, a fully qualified template tag to list all cities in uppercase from the data source will look like:

```
{{#ds}}{{#ds.cities}}{{ds.cities.name}:toupper()}}{{/ds.cities}}{{/ds}}
```

 **Note:** A Root template is a part of document which contains template tags and is covered by `{{#ds}}...{/ds}}` tags, where ds is name of used datasource. A single Root template is supported per Word document.

 **Note:** Sometimes, you may need to exclude some template tags from processing to make them appear as they are in the document (for example, to show a tag and the result of processing it side by side). DsWord allows you to do it by adding a backslash (\) immediately before the opening curly braces of the tag. To insert a backslash and still process the tag, use two backslashes. The backslashes have no special meaning to the template engine except when immediately preceding a template tag.

Note that when escaping several related template tags, you must escape them all; otherwise, the template engine will encounter an invalid template construct and throw an exception. For example, to escape the construct "`{{#ds}}{{ds.seas.name}:toupper()}/ds}}`", use "`\"{{#ds}}\"{{ds.seas.name}:toupper()}/ds}}`".

State Functions

The **IsFirst**, **IsLast**, and **Index** template state functions enable you to identify the first and last iteration, as well as the index of the current iteration in a collection.

IsFirst returns true when the index of the current iteration is zero. IsLast returns true when the index of the current iteration is Count-1 in the collection items. Index returns the index of the current iteration in a collection.

Example: Record `{{calc Index(ds)+1}}` of `{{calc Count(ds)}}`: `{{ds.name}}``{{if IsFirst(ds)}} (first){/if}``{{if IsLast(ds)}} (last){/if}``{{endif}}`

Here, "ds" is the name of the data source, and "ds.name" is the name field in the data source. "calc Index(ds)+1" gets the index of the current iteration in the collection and adds one to it. "calc Count(ds)" gets the total number of iterations in the collection. "if IsFirst(ds)" checks if the current index is the first index of the collection, and "if IsLast(ds)" checks if the current index is the last index of the collection.

Result:

Record 1 of 5: Australasian Mediterranean Sea (first)

Record 2 of 5: Philippine Sea

Record 3 of 5: Coral Sea

Record 4 of 5: South China Sea

Record 5 of 5: Sargasso Sea (last)

For more information on how to implement state functions, see [Collection State](#).

List

You can use various types of lists in the template layout by specifying the correct syntax as explained below:

Bullet List

To generate output data in bullet list structure, add a bullet to the template tag by using 'Bullets' button in the template.




For example, consider the following sample data:

JSON

```
[
  {
    "product": "Canon",
    "id": "0050"
  },
  {
    "product": "Bose",
    "id": "9809"
  },
  {
    "product": "Redmi",
    "id": "5643"
  }
]
```

By using above data as data source, the template tag on the left will generate the bulleted list data, as displayed on the right:

- `{{product}} {{id}}` 
 - Canon 0050
 - Bose 9809
 - Redmi 5643

Number List

To generate output data in numbered list structure, add a number to the template tag by using 'Numbering' button in the template.




For example, consider the following sample data:

JSON

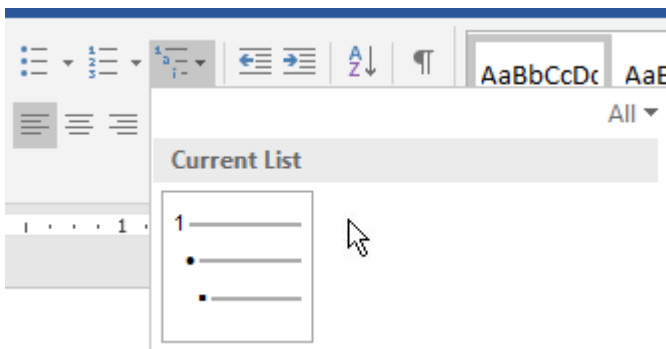
```
[
  {
    "product": "Canon",
    "id": "0050"
  },
  {
    "product": "Bose",
    "id": "9809"
  },
  {
    "product": "Redmi",
    "id": "5643"
  }
]
```

By using above data as data source, the template tag on the left will generate the numbered list data, as displayed on the right:

1. `{{product}} {{id}}`  1. Canon 0050
2. Bose 9809
3. Redmi 5643

MultiLevel List

To generate output data in multilevel list structure, add multiple levels to the template tag by using 'Multilevel List' button in the template.



For example, consider the following sample data of different companies, their revenue and cities which generate the maximum revenue:

JSON

```
[
  {
    "company": "Amazon",
    "percentOfRevenue": "28.2%",
    "cities": [
      { "name": "Delhi" },
      { "name": "Bangalore" },
      { "name": "Mumbai" }
    ]
  }
]
```

```

]
},
{
  "company": "Flipkart",
  "percentOfRevenue": "11.5%",
  "cities": [
    { "name": "Bangalore" },
    { "name": "Delhi" },
    { "name": "Hyderabad" }
  ]
},
{
  "company": "Snapdeal",
  "percentOfRevenue": "4.29%",
  "cities": [
    { "name": "Delhi" },
    { "name": "Hyderabad" },
    { "name": "Kolkatta" }
  ]
}
]

```

By using above data as data source, the template tag on the left will generate the multilevel list data, as displayed on the right:

1. {{company}} – {{percentOfRevenue}}
 • {{cities.name}}



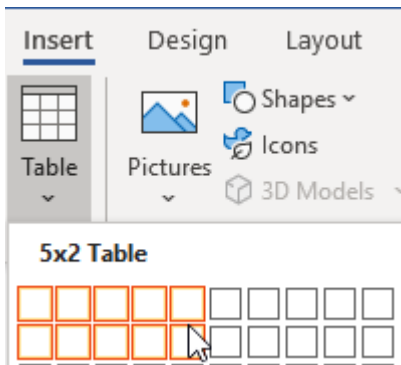
1. Amazon – 28.2%
 - Delhi
 - Bangalore
 - Mumbai
2. Flipkart – 11.5%
 - Delhi
 - Bangalore
 - Hyderabad
3. Snapdeal – 4.29%
 - Delhi
 - Hyderabad
 - Kolkata

Table

Tables help in organizing a large amount of data in a well structured manner. You can generate data in regular as well as dynamic tables in Word documents by using Report Templates in DsWord.

Add Table

To generate output data in a tabular form, insert a Table in the template layout by using 'Insert Table' option in the template.



Consider the following sample data of a school, its contact details and a few of its students:

JSON

```
{
  "school": {
    "name": "Montessori",
    "email": "contact@montessori.com"
  },
  "students": [
    {
      "name": "Richard",
      "grade": "8",
      "eyecolor": "Blue",
      "weight": "40",
      "phone": "(206) 854-9798"
    },
    {
      "name": "Jared",
      "grade": "8",
      "eyecolor": "Financial Department",
      "weight": "45",
      "phone": "(206) 598-1259"
    },
    {
      "name": "Natalie",
      "grade": "7",
      "eyecolor": "Marketing Department",
      "weight": "38",
      "phone": "(206) 789-1598"
    }
  ]
}
```

```

    "name": "Angela",
    "grade": "9",
    "eyecolor": "Financial Department",
    "weight": "42",
    "phone": "(206) 784-1258"
  }
]
}

```

By using above data as data source, the below template layout will generate the corresponding output:

Students at {{school.name}}

Email: {{school.email}}

Name	Grade	Eye Color	Weight	Phone
{{students.name}}	{{students.grade}}	{{students.eyecolor}}	{{students.weight}}	{{students.phone}}



Students at Montessori

Email: contact@montessori.com

Name	Grade	Eye Color	Weight	Phone
Richard	8	Blue	40	(206) 854-9798
Jared	8	Brown	45	(206) 598-1259
Natalie	7	Black	38	(206) 789-1598
Angela	9	Hazel	42	(206) 784-1258

Add Table with Repeating Rows

To generate output data in a table with repeating rows, insert a Table in the template layout by using 'Insert Table' in the template and add the number of rows you want to repeat for the same object. Consider the following sample data of a few electronics with their revenue and area of sale:

JSON

```

[
  {
    "name": "Canon 0050",
    "area": "North America",
    "Revenue": "$92800"
  },
  {
    "name": "Bose 9809",
    "area": "New York",

```

```

    ""Revenue"": ""$78900""
  },
  {
    ""name"": ""Redmi 5643"",
    ""area"": ""Chicago"",
    ""Revenue"": ""$57800""
  },
]

```

The template tags for the properties of same object are defined in two rows. Hence, the generated output repeats both rows for each object. The below template layout will generate the corresponding output as displayed:

Name	Area
{{name}}	{{area}}
{{revenue}}	

↓

Name	Area
Canon 0050	North America
\$92800	
Bose 9809	New York
\$78900	
Redmi 5643	Chicago
\$57800	

Add Dynamic Table

To create dynamic tables which render the table data automatically, you need to define a two-dimensional array which can be added in a single template tag. Consider the following sample data which specifies the time to take medicines for different medical reasons.

JSON

```

{
  ""myArray"": [
    [
      ""Fever"",
      ""Wake up"",
      ""Fever only""
    ],
    [
      ""Cough"",
      ""After meal, Before meal,"",
      ""Nausea only""
    ],
    [
      ""Cold"",
      ""Before sleep"",
      ""Required""
    ]
  ]
}

```



```

    ],
  ]
}

```

The array specified in template tag replicates the data dynamically in table rows and columns. The below template layout will generate the corresponding output:

{{myArray}}



Fever	Wake up	Fever only
Cough	After meal, Before meal	Nausea only
Cold	Before sleep	When required

Add Table Columns Dynamically

Similar to dynamic tables, you need to define a two-dimensional array to add table columns dynamically. Consider the following sample data of a school and its students. Notice that the metadata property is a two-dimensional array.

JSON

```

{
  "company": "Montessori",
  "contacts": {
    "website": "http://montessori.com",
    "support": "contacts@montessori.com",
    "sales": "sales@montessori.com"
  },
  "students": [
    {
      "name": "Richard",
      "metadata": [
        [
          "Monitor",
          "Grade 10",
          "(206) 854-9798"
        ]
      ]
    },
    {
      "name": "Jared",
      "metadata": [
        [
          "Head Boy",
          "Grade 12",
          "(206) 598-1259"
        ]
      ]
    }
  ]
}

```

```
    ]
  },
  {
    "name": "Natalie",
    "metadata": [
      [
        "Prefect",
        "Grade 10",
        "(206) 789-1598"
      ]
    ]
  },
  {
    "name": "Angela",
    "metadata": [
      [
        "Head Girl",
        "Grade 12",
        "(206) 784-1258"
      ]
    ]
  }
]
```

The array specified in template tag creates a new column for each item in the array. The below template layout will generate the corresponding output:

{{school}}

Website: {{contacts.website}}

Technical Support: {{contacts.support}}

Sales department: {{contacts.sales}}

Students

{{students.name}}	{{students.metadata}}
-------------------	-----------------------



Montessori

Website: <http://montessori.com>

Technical Support: contacts@montessori.com

Sales department: sales@montessori.com

Students

Richard	Monitor	Grade 10	(206) 854-9798
Jared	Head Boy	Grade 12	(206) 598-1259
Natalie	Prefect	Grade 10	(206) 789-1598
Angela	Head Girl	Grade 12	(206) 784-1258

Limitation

Template tags inside collection tags must use only tags from parent-child chain. Access to sibling elements is not allowed in iterative elements (lists/tables etc)

Links

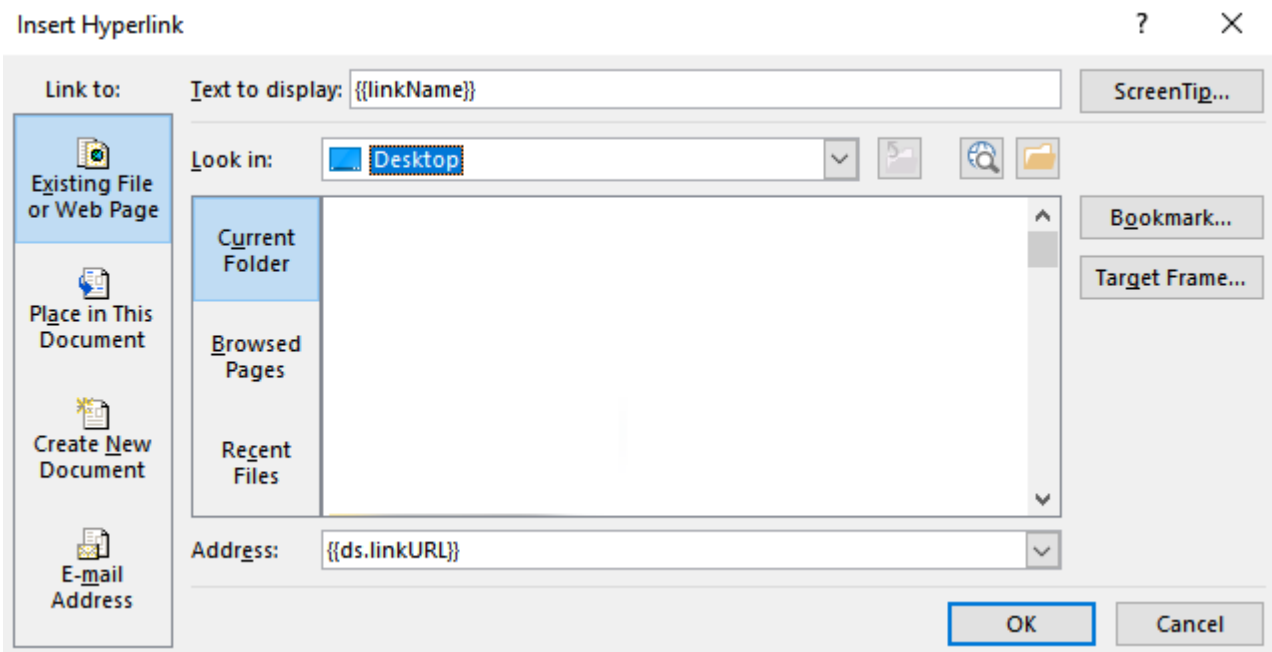
While using Report Templates in DsWord, you can also add hyperlinks to Word documents which can navigate to any specified web URL.

For example, consider a template layout for Document Solutions products which are added in a table.

Document Solutions

Name	Description	Link
{{ds.name}}	{{ds.description}}	{{ds.linkName}}

A Hyperlink is added to {{linkName}} field by clicking on 'Link' in 'Insert' tab and adding a template tag {{ds.linkURL}} (here ds is the name of data source), into the Address field as shown below:



Consider the following sample data:

```
JSON
[
  {
    "name": "Document Solutions for Word",
    "description": "DsWord is a part of Document Solutions that aims to be a complete solution to program and work with Word documents",
    "linkName": "Learn More",
    "linkURL": "https://developer.mescius.com/documents-api-word"
  },
  {
    "name": "Document Solutions for Imaging",
    "description": "DsImaging offers imaging API for image processing without using any external image editor.",
    "linkName": "Learn More",
    "linkURL": "https://developer.mescius.com/documents-api-imaging"
  }
]
```

```

},
{
  "name": "Document Solutions for Pdf",
  "description": "Document Solutions for PDF handles majority of the PDF related
needs as it conforms to a large part of Adobe PDF specification",
  "linkName": "Learn More",
  "linkURL": "https://developer.mescius.com/documents-api-word"
},
{
  "name": "Document Solutions for Excel, .NET",
  "description": "Document Solutions for Excel, .NET Edition is a new small-
footprint, high-performance spreadsheet component that can be used in your server or
desktop applications.",
  "linkName": "Learn More",
  "linkURL": "https://developer.mescius.com/documents-api-excel"
},
{
  "name": "Document Solutions for Excel, Java",
  "description": "DsExcel Java is a high-performance spreadsheet component that
comes packaged with all the necessary features to help users handle complex
spreadsheet challenges in an efficient way.",
  "linkName": "Learn More",
  "linkURL": "https://developer.mescius.com/documents-api-excel-java"
}
]

```

The below output is generated upon template processing which contains hyperlinks to corresponding web pages:

Document Solutions

Name	Description	Link
Document Solutions for Word	DsWord is a part of Document Solutions that aims to be a complete solution to program and work with Word documents	Learn More
Document Solutions for Imaging	DsImaging offers imaging API for image processing without using any external image editor.	Learn More
Document Solutions for Pdf	Document Solutions for PDF handles majority of the PDF related needs as it conforms to a large part of Adobe PDF specification	Learn More
Document Solutions for Excel, .NET	Document Solutions for Excel, .NET Edition is a new small-footprint, high-performance spreadsheet component that can be used in your server or desktop applications.	Learn More
Document Solutions for Excel, Java	DsExcel Java is a high-performance spreadsheet component that comes packaged with all the necessary features to help users handle complex spreadsheet challenges in an efficient way.	Learn More

Note: DsWord supports using template documents containing malformed URI by rewriting them in a valid manner using **MalformedUriRewriter** property. To know more, refer [Hyperlink](#).

Limitation

You can add `{{ds.linkURL}}` as a hyperlink address on a template tag (as explained above) but it is not allowed to specify any hyperlink inside a template tag.

Loops and Nesting

Report Templates allow you to create repeating objects by specifying one repeating object inside the other. This is helpful while implementing complex use case scenarios.

For example, the below sample data represents a complex object with a few nested objects and collections for a student's grade card.

JSON

```
[
  {
    "name": "Richard",
    "school": [
      {
        "name": "Montessori",
        "curriculum": [
          {
            "category": "Dance",
            "achievement": [
              {
                "description": "Performed in inter-school
events"
              }
            ]
          }
        ],
        "student": [
          {
            "subject": "English",
            "grade": "8",
            "remarks": "Good pick-up but there is scope of
improvement in grammar"
          }
        ]
      }
    ]
  }
]
```

The template layout contains a table containing placeholder fields for the subject, grade and its remarks. It also defines nested fields for Co-Curricular activities and their respective achievements.

{{school.name}}

Grade Card:

{{name}}

Subject	Grade
{{school.student.subject}}	{{school.student.grade}}
{{ school.student.remarks}}	

Co-Curricular Activities:

- {{school.curriculum.category}}
 - {{school.curriculum.achievement.description}}

The output of above template layout is shown below. The final Word document will contain repeated data for different students, if provided in the data source.

Montessori

Grade Card:

Richard

Subject	Grade
English	8
Good pick-up but there is scope of improvement in grammar	
Maths	9.5
Great speed and accuracy. Tries to solve complex problems	

Co-Curricular Activities:

- Dance
 - Performed in inter-school events
- Sports
 - Average performance in Football

Conditionally Hide Blocks

Conditionally hide blocks allow you to use conditional formatting by hiding the specified blocks of a document. You can use following conditions to check and then hide or replace the blocks if the specified criteria does not satisfy.

- hide-block-if
- if-endif
- if-else-endif

Hide Specific Block

The hide-block-if condition lets you hide a particular block if it does not meet the mentioned criteria. The possible block types in a document can be:

- Table row
- Bullet List item
- Chapter
- Paragraph

Hide Table Rows

The following example considers sample data of students who belong to a particular society. The conditionally hide block formatter applies conditional formatting to hide table rows if a student belongs to "Dance" society.

JSON

```
{
  "students": [
    {
      "name": "Richard",
      "society": "Dramatics"
    },
    {
      "name": "Natalie",
      "society": "Music"
    },
    {
      "name": "Angela",
      "society": "Dance"
    }
  ]
}
```

The template layout with hide-if formatter and the result generated after template processing is shown below:

Students

Name	Society
{{students.name}}	{{students.society}:hide-block-if-equals(dance)}



Students

Name	Society
Richard	Dramatics
Natalie	Music

Hide Bullet List Items

Similarly considering the above example, you can also hide a block containing bullet list item by applying a hide-if formatter as shown below:

Students

- {{students.name}} {{students.society}:hide-block-if-equals(dance)}



Students

- Richard Dramatics
- Natalie Music

Hide Null Block

This example applies conditional formatting to hide blocks of data containing null values.

JSON

```
{
  "school": "Montessori",
  "site": "http://montessori.com",
  "contacts": []
}
```

The template layout with hide-if formatter and Word document generated after template processing is shown below. The Phone field is not displayed in the output as it contains no value.

```
{{school}}
```

```
Site: {{site}}
```

```
Phone: {{contacts}} {{contacts:hide-block-if-empty()}}
```



Montessori

Site: <http://montessori.com>

Hide Multiple Blocks

The if-endif function lets you hide blocks that do not meet the specified condition. The condition can be created using template value operands combined with arithmetic, string or logic operations. An example of if-endif condition is shown below where sea names are generated only if they start with "S".

Data source

```
// The data source (ocean and sea data from Wikipedia):
var oceans = new[]
{
    new { name = "Pacific", areaOfWorldOcean = 0.466, volumeOfWorldOcean = 0.501, seas
= new[]
        { new {name = "Australasian Mediterranean Sea" }, new { name = "Philippine
Sea" }, new { name = "Coral Sea" }, new { name = "South China Sea" } }
    },
    new { name = "Atlantic", areaOfWorldOcean = 0.235, volumeOfWorldOcean = 0.233,
seas = new[]
        { new {name = "Sargasso Sea" }, new { name = "Caribbean Sea" }, new { name =
"Mediterranean Sea" }, new { name = "Gulf of Guinea" } }
    },
    new { name = "Indian", areaOfWorldOcean = 0.195, volumeOfWorldOcean = 0.198, seas
= new[]
        { new {name = "Arabian Sea" }, new { name = "Bay of Bengal" }, new { name =
"Andaman Sea" }, new { name = "Laccadive Sea" } }
    },
};
```

if-endif

```
// Filter output using if/endif - print only names starting with an 's':
var p = doc.Body.Paragraphs.Add("{{if StartsWith(UCCase(ds.seas.name), \"S\")}}
{{ds.seas.name}}{{endif}}", doc.Styles[BuiltInStyleId.ListParagraph]);
```

Hide and Replace with Alternative Block

The if-else-endif function lets you choose between two options based on a condition. Rendering a checked or


unchecked checkbox depending on a condition is a typical example of if-else condition.

The below example uses the data source used in the section above and shows how to render a block when sea name starts with 'S', while alternate string is rendered when it does not.

if-else

```
// Filter output using if/else - print names starting with an 's', else print a string:
var p = doc.Body.Paragraphs.Add("{{if StartsWith(UCase(ds.seas.name), \"S\")}}
{{ds.seas.name}}{{else}}(\"Sea name doesn't start with S\"){{endif}}",
doc.Styles[BuiltInStyleId.ListParagraph]);
```

To view if-else and if-endif functions in action, see [StartsWith](#) demo sample.

 **Note:** Template values inside expression are used without usual {{ and }} braces.

Limitations

- if functions cannot be used without at least one datasource added to DataTemplate.
- if-endif and if-else blocks must start and end in the same cell

Calculation

Calc Expression

Calculation functions calculate inner expression and write result as output. Currently, a function subset of our [Calc engine](#) has been adapted to use with the template engine and hence the feature supports various binary, unary and arithmetic operators, aggregate functions, constants, and text functions. The examples below show how to use aggregate and arithmetic operations to create expressions in Calc function.

Refer to the following example code to calculate the average using multiple arguments:

```
C#  
  
// Initialize GcWordDocument.  
GcWordDocument doc = new GcWordDocument();  
var ds = new int[] { 1, 2, -2, 16, 7, 5, 4 };  
  
// Bind data source.  
doc.DataTemplate.DataSources.Add("ds", ds);  
  
// Calculate average using multiple arguments.  
doc.Body.Paragraphs.Add("{ calc Average(ds.value+Sum(ds.value)/2) }");  
doc.DataTemplate.Process();  
doc.Save("complex-expression-inside-average.docx");
```

Refer to the following example code to calculate the sum using a constant:

```
C#  
  
// Initialize GcWordDocument.  
var doc = new GcWordDocument();  
var ds = new int[] { 1, 2, -2, 16, 7, 5, 4 };  
  
// Bind data source.  
doc.DataTemplate.DataSources.Add("ds", ds);  
  
// Calculate sum using a constant.  
doc.Body.Paragraphs.Add("{ calc Sum(2 + ds.value) }");  
doc.DataTemplate.Process();  
doc.Save("sum-with-constant.docx");
```

Refer to the following example code to calculate the average using two collections inside the aggregate function:

```
C#  
  
// Initialize GcWordDocument.  
var doc = new GcWordDocument();  
var ds = new int[] { 1, 2, -2, 16, 7, 5, 4 };  
var ds2 = new int[] { 5, 3, };  
  
// Bind data source.  
doc.DataTemplate.DataSources.Add("ds", ds);  
doc.DataTemplate.DataSources.Add("ds2", ds2);
```

```
// Calculate average using two collections inside the aggregate function.
doc.Body.Paragraphs.Add("{ calc Average(ds.value+ds2.value)}");
doc.DataTemplate.Process();
doc.Save("sum-with-two-colls.docx");
```

To view Calc functions in action, see [Calculation demo](#) sample.

Format Function

DSWord provides **Format** function that can calculate inner expressions and output the result in different formats and cultures. The Format function is used with the Calc Expression.

Refer to the following example code to display the data in date formatting:

```
C#
// Add dates in date format and different locales.
var paras = doc.Body.Paragraphs;
var p = paras.Add("Date formatting examples:", style);
p.ListFormat.Template = list;
p = paras.Add("Date with default formatting: {{calc Format(dsDate.value)}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;
p = paras.Add("Date formatted as ('D', 'en-US'): {{calc Format(dsDate.value, \"D\", \"en-US\")}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;
p = paras.Add("Date formatted as ('D', 'fr-FR'): {{calc Format(dsDate.value, \"D\", \"fr-FR\")}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;
p = paras.Add("Date formatted as ('D', 'ja-JP'): {{calc Format(dsDate.value, \"D\", \"ja-JP\")}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;
```

- Date formatting examples:
 - Date with default formatting: 4/12/2023
 - Date formatted as ('D', 'en-US'): Wednesday, April 12, 2023
 - Date formatted as ('D', 'fr-FR'): mercredi 12 avril 2023
 - Date formatted as ('D', 'ja-JP'): 2023年4月12日水曜日

Refer to the following example code to display the data in number formatting:

```
C#
// Add number in different fromats and cultures.
p = paras.Add("Number formatting examples:", style);
p.ListFormat.Template = list;
p = paras.Add("Number with default formatting: {{calc Format(dsNum.value)}}",
```

```

style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;
p = paras.Add("Number formatted as ('C', 'en-US'): {{calc Format(dsNum.value, \"C\", \"en-US\")}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;
p = paras.Add("Number formatted as ('C', 'fr-FR'): {{calc Format(dsNum.value, \"C\", \"fr-FR\")}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;
p = paras.Add("Number formatted as ('C3', 'ja-JP'): {{calc Format(dsNum.value, \"C3\", \"ja-JP\")}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;

```

- Number formatting examples:
 - Number with default formatting: 123.456
 - Number formatted as ('C', 'en-US'): \$123.46
 - Number formatted as ('C', 'fr-FR'): 123,46 €
 - Number formatted as ('C3', 'ja-JP'): ¥ 123.456

Refer to the following example code to display the data in expression formatting:

```

C#
// Add expressions in different formats.
p = paras.Add("Expression formatting examples:", style);
p.ListFormat.Template = list;
p = paras.Add("Numeric expression: {{calc Format(dsNum.value / 2)}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;
p = paras.Add("Boolean expression: {{calc Format(IsFirst(dsNum.value))}}", style);
p.ListFormat.Template = list;
p.ListFormat.LevelNumber = 1;

```

- Expression formatting examples:
 - Numeric expression: 61.728
 - Boolean expression: True

Limitations

- Calc functions cannot be used without at least one datasource added to DataTemplate.
- Function names are reserved and cannot be used inside data sources. This limitation is case in-sensitive and limits only simplified names. So if you have datasource field **ds.count**, you can use it as {{calc ds.count*5}}. Calc Engine fails to process if you use it as {{calc count*5}}.

- When multiple collections of different lengths are used at the same level of expression, the engine will determine the minimal length of the used collections and use it for both collections. In the code example above, both collections ds and ds2 are used "at the same level" in the expression. So, only two elements from both collections will be used, as 2 is $\text{Min}(\text{ds.Length}, \text{ds2.Length})$.

**Note:**

- Template values inside expression are used without usual {{ and }} braces.
- In many cases, result type is double, even if original type was integer.

Formatters

DsWord supports various kinds of formatters in Report Templates which allow you to format the Word document in a structured manner. You can apply any input formatting to generate the desired output. For example, hiding some piece of information based on a condition, displaying strings in upper or lower case, inserting an image, displaying date time in the specified format etc.

Range Formatters

Range formatters can be used to hide a block of data if an enumerable collection meets the formatter requirements. These are mentioned below along with their short versions:

- hide-block-if-equals() OR hbi-equals()
- hide-block-if-contains() OR hbi-contains()
- hide-block-if-empty() OR hbi-empty()

The collections of standard C# types can be only formatted by using hbi-contains/hbi-equals range formatters.

Example: `{{ds.countries}:hbi-equals("England")}`

Block Hiding Formatters

These formatters can be used to hide a block of data if the value in a block meets the condition. These formatters are mentioned below along with their short versions:

- hide-block-if-equals() OR hbi-equals()
- hide-block-if-greater() OR hbi-greater()
- hide-block-if-less() OR hbi-less()
- hide-block-if-contains() OR hbi-contains()
- hide-block-if-starts() OR hbi-starts()
- hide-block-if-ends() OR hbi-ends()
- hide-block-if-empty() OR hbi-empty()
- hide-block-if-not-equals() OR hbi-not-equals()
- hide-block-if-not-contains() OR hbi-not-contains()
- hide-block-if-not-starts() OR hbi-not-starts()
- hide-block-if-not-ends() OR hbi-not-ends()

Example: `{{ds.year}:hide-block-if-equals("2020")}`

Value Formatters

Value formatters can be used to format the value of template tags. You can use these with or without parameters:

- format() - Default. If applied on a date, will format as short date string "d", numeric types will be formatted as "g"
- format(format) - Formats the value using the specified format string, e.g. "format(N2)"
- substring(index) - Returns the specified substring of the value
- substring(index,length) - Returns the specified substring of the value
- join(separator) - Joins array values using the specified separator
- bool(yes,no,maybe) - Converts a boolean to a 'yes'/'no'/'maybe' values ('maybe' is optional and is used if the value is null)
- empty(default) - If the value is null, empty or empty array, returns 'default'

For example:

Sample Data	Template Tag	Output
-------------	--------------	--------

{ Name= "Richard Clark"}	{{Name}:substring(5)}	Clark
--------------------------	-----------------------	-------

String Formatters

String formatters can be used to format string values:

- length - Gets the number of characters in the current string.
- tolower - Returns a copy of this string converted to lowercase.
- toupper - Returns a copy of this string converted to uppercase.
- trim - Removes all leading and trailing white-space characters from this string.
- gethashcode - Gets the hash code of this string.

For example:

Sample Data	Template Tag	Output
{ cities = new[] {name = "China"};}	{{cities.name}:toupper()}	CHINA

Date and time formatters

The below mentioned formatters are applicable to DateTime and DateTimeOffset:

- date - Date component with zeroed time
- day - Day of the month
- dayofweek - Day of the week
- dayofyear - Day of the year
- hour - Hour component of the DateTime
- millisecond - Millisecond component of this DateTime
- minute - Minute component of this DateTime
- month - Month component of this DateTime
- second - Second component of this DateTime
- ticks - Number of ticks representing this DateTime
- timeofday - Time of day
- year - Year component of this DateTime
- add - Adds TimeSpan to this DateTime
- subtract - Subtracts TimeSpan from this DateTime
- tofiletime - Converts this DateTime to a Windows file time

The below mentioned formatters are applicable to DateTime:

- gethashcode - Hash code of this DateTime.
- tolongdatestring - Long string date representation of this DateTime.
- tolongtimestring - Long string time representation of this DateTime.
- toodate - OLE Automation date representation of this DateTime.
- toshortdatestring - Short string date representation of this DateTime.
- toshorttimestring - Short string time representation of this DateTime.
- tofiletimeutc - Converts the value of this DateTime to a Windows file time.

For example:

Sample Data	Template Tag	Output
{ Timestamp= "2020-04-21T16:15:45-08:00"}	{{TimeStamp}:year()}	2020

Image Formatters


Image formatters convert a value to an image and inserts it into the document. The value must be of one of the following types:

- A byte array containing image data
- A System.IO.Stream object that can be used to read image data
- A System.Drawing.Image object
- A string that can be converted to an absolute file URI, or containing Base64-encoded image data

Formatters:

- `image(width, height)` - Inserts the image with the specified width and height
- `image()` - If the image is inside a shape, it is stretched to fill the shape. Otherwise it is inserted with its original width and height
- `image(keepratio|fitheight|fitwidth|fitsize|fitsizelim)` - Fits the image into the host shape using a specified fitting strategy
- `keepratio` - Stretches the image to fit the host shape, preserving the image aspect ratio
- `fitheight` - The height of the host shape is adjusted so that the image fills the shape, preserving the image aspect ratio
- `fitwidth` - The width of the host shape is adjusted so that the image fills the shape, preserving the image aspect ratio
- `fitsize` - Fits the size of the host shape to the size of the image
- `fitsizelim` - Like `fitsize` but does not increase the size of the shape

Example: `{{ds.value}:image(100,100)}`

 **Note:** Image formatters use System.Drawing.Image class. This works out of the box on Windows, but requires installation of additional shared libraries on Linux and macOS:


- Linux:
`sudo apt-get update`
`sudo apt-get install libc6-dev libgdiplus -y`
- macOS
`brew install mono-libgdiplus`

Type Conversion Formatters

As the name suggests, the type conversion formatters let you convert the input types into numeric type data. These formatters are helpful in conversion of strings that hold numeric data in the JSON data source.

Formatters:

- `todouble(optional_string)` - Converts the input type to a double value.
- `tobool(optional_string)` - Converts the input type to a bool value.

 **Note:** If conversion fails, then the `optional_string` parameter is converted. In case the optional string also cannot be converted, then the optional string itself is returned as output.

Limitations:

Culture-specific bool conversion works only with user-defined types because conversions are implemented using Convert API. For instance, "true" or "True" strings are converted to **true** regardless of CultureInfo setting. That is,

- when numeric type is used as data source, 0 is converted to False and any other number except 0 is converted to True.
- when string type is used as data source, only "true", "True", "false", and "False" strings are converted to bool regardless of culture. Any other string, including "0" and "1" are not converted. This rule is applicable to **optional_string** parameter as well.

For example:


Sample Data	Template Tag	Output
{ "-2", "32", "16", "5,45", "12,39" }	{{ds.value}:todouble():hbi-less(7)}	32 16 12,39
{ d = "samplestring" }	{{ds.d}:tobool(false)}	False

Miscellaneous Formatters:

Report Templates also support following formatters:

- length - Gets the total number of elements in all dimensions of this array. Example: `{{ds.countries}:length()}}`
- count - Gets the number of elements in this collection. Example: `{{ds.countries}:count()}}`

Example: `{{ds.countries}:count()}}`

 **Note:** Chars like '{',';',':' are allowed inside formatter parameters but only when paired with '\' so `format({x:y})` should be written as `format(\{x\;y\})`. This is due to the regex nature of engine.

Note:

- DsWord uses "not" instead of "!" for negative checks. The use of "!" will throw an exception when processing the template.
For example: `{{if not isLast(ds.companyKeywordList)}}`
- Formatters cannot be applied to tags inside a conditional template's calc and if blocks.
For example: `ds.tag:toupper()` cannot be used as `{{if ds.tag:toupper() = "XXX"}}` and `{{calc ds.tag:toupper() ...}}`.
- Functions cannot be used outside conditional template definitions.
For example: `{{#ds}}{{ToUpper(ds.tag)}}{{/ds}}`.
- The short notation will create all necessary range tags automatically. If the user wants to inspect such auto-generated tags, user should use `DataTemplate.DebugExpandTemplate` API. It is useful when the user wants to know why and how the template structure was auto-generated in an unexpected way.

Restart List Formatter

DsWord provides 'restart' formatter in Report Templates which can be used for numbered lists to make them restart on each iteration within a template range. The template range here refers to a range where a template tag starts with a '#' and ends with a '/'.

The below example explains the usage of 'restart' formatter when two numbered lists are used. As can be observed from the output, the numbered list is not restarted where the formatter itself is used, rather it is restarted on the inner numbered list.

Template Tags	Output (Template processing using a datasource)
<pre>1. {{#companies:restart()}}{{companies.id}} a. {{companies.items.id}}{/companies}</pre>	<pre>1. Company A a. Milk b. Eggs 2. Company B a. Steel b. Aluminium 3. Company C a. Wool b. Cotton</pre>

The below example explains the usage of 'restart' formatter with 'all' parameter (optional) and two numbered lists. As can be observed from the output, the numbered list is restarted at both levels, with each iteration.

Template Tags	Output (Template processing using a datasource)
<pre>1. {{#companies:restart(all)}}{{companies.id}} a. {{companies.items.id}}{/companies}</pre>	<pre>1. Company A a. Milk b. Eggs 1. Company B a. Steel b. Aluminium 1. Company C a. Wool b. Cotton</pre>

The below example shows how to create a numbered list of oceans with a numbered list of seas under its listing. Here, the numbered list of seas is restarted on each iteration by using the 'restart' formatter. The below image shows the template tags used inside the template. You can also download the [Template layout](#) here.

- ```
1. {{#ds:restart()}}{{ds.name}} Ocean:
 a. {{#ds.seas}}{{ds.seas.name}}{/ds.seas}{/ds}
```

The below code example loads the above template and processes it using a JSON data source:

```
C#

var doc = new GcWordDocument();
//Load template
doc.Load("RestartFormatter[Template].docx");

//Add datasource
var oceans = File.OpenRead("oceans.json");
doc.DataTemplate.DataSources.Add("ds", oceans);

//Process template
doc.DataTemplate.Process();
doc.Save("RestartListFormatter.docx");
```

The output of above code example will look like below:

1. Pacific Ocean:
  - a. Australasian Mediterranean Sea
  - b. Philippine Sea
  - c. Coral Sea
  - d. South China Sea
2. Atlantic Ocean:
  - a. Sargasso Sea
  - b. Caribbean Sea
  - c. Mediterranean Sea
  - d. Gulf of Guinea
3. Indian Ocean:
  - a. Arabian Sea
  - b. Bay of Bengal
  - c. Andaman Sea
  - d. Laccadive Sea
4. Southern Ocean:
  - a. Weddell Sea
  - b. Somov Sea
  - c. Riiser-Larsen Sea
  - d. Lazarev Sea
5. Arctic Ocean:
  - a. Barents Sea
  - b. Greenland Sea
  - c. East Siberian Sea
  - d. Kara Sea

## Sequence and Follow Formatter

Sequence and Follow formatter can be used to co-iterate the data from different datasources, meaning that the data from first datasource should contain only one iteration of data from the another datasource. To understand this better, lets assume we have two datasources:

```
numbers = new[] { new { id = 1 }, new { id = 2 }, new { id = 3 }, new { id = 4 } }
letters = new[] { new { id = "Apple" }, new { id = "Banana" }, new { id = "Orange" }, new { id = "Mango" } }
```


After using a template layout and processing it, the output will look like:

| Without Sequence and Follow Formatter | With Sequence and Follow Formatter |
|---------------------------------------|------------------------------------|
| 1 Apple                               | 1 Apple                            |
| 1 Banana                              | 2 Banana                           |
| 1 Orange                              | 3 Orange                           |
| 1 Mango                               | 4 Mango                            |
| 2 Apple                               |                                    |
| 2 Banana                              |                                    |
| ..... so on                           |                                    |

The sequence and follow formatter can be used within a template range where a template tag starts with a '#' and ends with a '/'. The syntax of the formatters is:

- seq(unique\_id)/sequence(unique\_id)
- follow(unique\_id)

Here, the unique\_id is an alias name which is bound to the template range containing the sequence formatter. The alias name should be unique across the template layout.

 **Note:** The sequence formatter can be written as 'sequence' or 'seq'.

The usage of sequence and follow formatters should be done together to observe their intended behavior. The sequence formatter alone does not affect anything in the template range. However, when used with the follow formatter, it produces strictly one copy and data index of data in the follower collection. A usage example of both formatters in a template range:

```
{{#numbers}:seq(num){{numbers.id}}{{#letters}:follow(num){{letters.id}}{/letters}}{/numbers}}
```

The following example shows how to use sequence and follow formatters to iterate only once over the data in two different datasources. The below image shows the template tags used inside the template. You can also download the [Template layout](#) here.

- `{{#nums}:seq(seq1){{nums.num}}|{{#ds}:follow(seq1){{ds.name}}{/ds}}{/nums}}`

The below code example loads the above template and processes it to save the final Word document:

```
C#
var doc = new GcWordDocument();
doc.Load("SeqFollowerFormatter.docx");
```


```
var sequence = @"[
 { ""num"": ""First"" },
 { ""num"": ""Second"" },
 { ""num"": ""Third"" },
 { ""num"": ""Fourth"" },
 { ""num"": ""Fifth"" },
 { ""num"": ""Sixth"" },
 { ""num"": ""Seventh"" },
 { ""num"": ""Eights"" },
 { ""num"": ""Ninth"" },
 { ""num"": ""Tenth"" }
]";
// Add data source
doc.DataTemplate.DataSources.Add("nums", sequence);
// Add second JSON data source
var oceans = File.OpenRead("oceans.json");
doc.DataTemplate.DataSources.Add("ds", oceans);

doc.DataTemplate.Process();
doc.Save("SeqFollowTemplate.docx");
```

The output of above code example will look like below:

- First - Pacific
- Second - Atlantic
- Third - Indian
- Fourth - Southern
- Fifth - Arctic

You can download the 'oceans.json' data source used in the above example by downloading the 'Sequence and Follow formatters' demo from [here](#).

 **Note:** If the number of items in sequence and follow ranges are different, the template engine will stop when it reaches the end of the shorter of the two ranges. In other words, the number of repetitions will be the minimum of the two ranges' lengths.

## Block Behavior Formatters

Block behavior formatters allow you to modify the default behavior when rendering a data range (a collection of data items). DsWord provides two block behavior formatters:

- **paragraph-block-behavior()**, **pbb()**: When applied to a start range tag inside a table cell, it causes the range blocks to repeat as separate paragraphs inside that single cell rather than repeating each block as a table row. This formatter cannot be used anywhere else. The end range tag must appear inside the same cell.
- **run-block-behavior()**, **rbb()**: When applied to a start range tag, it causes the range blocks to repeat as runs within the same paragraph rather than separate paragraphs.

### Paragraph block behavior (paragraph-block-behavior, pbb) formatter

Paragraph block behavior formatter is helpful in rendering a range of data or list of items, as a block inside a single table cell. It renders the range of data or items as a vertical list in the single cell of a table. This formatter is applicable only in the case of table cells. The following table demonstrates the difference in generated output without the pbb and with it:

|                                | Without pbb                                                                                                                                                                                                                                                                                                                                                                                      | With pbb                                                  |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|-------------------|----------------|-----------|-----------------|--------------|---------------|-------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|----------------|-----------|-----------------|--------------|---------------|-------------------|----------------|
| <b>Template Tags</b>           | <code>{{#ds.seas}}{{ds.seas.name}}{/ds.seas}}</code>                                                                                                                                                                                                                                                                                                                                             | <code>{{#ds.seas:pbb()}}{ds.seas.name}}{/ds.seas}}</code> |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| <b>Output</b>                  | <p>Pacific</p> <table border="1"> <tr><td>Australasian Sea</td></tr> <tr><td>Mediterranean Sea</td></tr> <tr><td>Philippine Sea</td></tr> <tr><td>Coral Sea</td></tr> <tr><td>South China Sea</td></tr> </table> <p>Atlantic</p> <table border="1"> <tr><td>Sargasso Sea</td></tr> <tr><td>Caribbean Sea</td></tr> <tr><td>Mediterranean Sea</td></tr> <tr><td>Gulf of Guinea</td></tr> </table> | Australasian Sea                                          | Mediterranean Sea | Philippine Sea | Coral Sea | South China Sea | Sargasso Sea | Caribbean Sea | Mediterranean Sea | Gulf of Guinea | <p>Pacific</p> <table border="1"> <tr><td>Australasian Mediterranean Sea</td></tr> <tr><td>Philippine Sea</td></tr> <tr><td>Coral Sea</td></tr> <tr><td>South China Sea</td></tr> </table> <p>Atlantic</p> <table border="1"> <tr><td>Sargasso Sea</td></tr> <tr><td>Caribbean Sea</td></tr> <tr><td>Mediterranean Sea</td></tr> <tr><td>Gulf of Guinea</td></tr> </table> | Australasian Mediterranean Sea | Philippine Sea | Coral Sea | South China Sea | Sargasso Sea | Caribbean Sea | Mediterranean Sea | Gulf of Guinea |
| Australasian Sea               |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Mediterranean Sea              |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Philippine Sea                 |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Coral Sea                      |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| South China Sea                |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Sargasso Sea                   |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Caribbean Sea                  |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Mediterranean Sea              |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Gulf of Guinea                 |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Australasian Mediterranean Sea |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Philippine Sea                 |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Coral Sea                      |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| South China Sea                |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Sargasso Sea                   |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Caribbean Sea                  |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Mediterranean Sea              |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |
| Gulf of Guinea                 |                                                                                                                                                                                                                                                                                                                                                                                                  |                                                           |                   |                |           |                 |              |               |                   |                |                                                                                                                                                                                                                                                                                                                                                                            |                                |                |           |                 |              |               |                   |                |

```
C#
// Add a list for oceans, and a nested table for seas:
doc.Body.Paragraphs.Add("{{#ds}}{{ds.name}}", doc.Styles[BuiltInStyleId.ListParagraph]);

var table = doc.Body.Tables.Add(1, 1);
table.Style = doc.Styles[BuiltInStyleId.GridTable1LightAccent1];
var cell_11 = table[0, 0];

//reuse first paragraph of the cell and define seas range with pbb() formatter.
//This formatter force range to be expanded in single cell, instead of generating row per each item.
cell_11.GetRange().Paragraphs.First.GetRange().Runs.Add("{{#ds.seas:pbb()}}{ds.seas.name}}{/ds.seas}");

//close parent range #ds.
doc.Body.Paragraphs.Add("{{/ds}}");
```

### Limitations

- A range defined with a paragraph block behavior formatter must start and end in the same table cell.
- If pbb range contains inner ranges, those are converted to pbb ranges too, unless they are already defined as **run-block-behavior(rbb)** ranges. This rule is not applicable to ranges in inner table. That is, if inner table is defined inside a pbb range, its ranges do not automatically inherit pbb behavior.
- In case of sibling, that is, a template range defined at same level in their parent range, following rules are applied:
  - Siblings of any type are valid if it does not include cell where pbb template is defined. If previous sibling is formatted as rbb or pbb in same cell, then pbb range is allowed to be the next sibling. Otherwise, template is considered malformed and exception **InvalidTemplateStructureException** is thrown.
  - If previous sibling is defined inside inner table of a cell, any type of siblings are allowed.
  - Any type of next siblings are allowed.
  - If next sibling in cell is rbb or default, we check the location where it is defined and template is considered malformed if this sibling is defined as end of pbb sibling in the same paragraph.

### Run block behavior (run-block-behavior, rbb) formatter

Run block behavior formatter causes the range blocks (data items in a range) to repeat as runs inside the same paragraph rather than separate paragraphs. The following table demonstrates the difference in generated output without the rbb and with it:



|                      | Without rbb                                                                                                                                                                                                                                                           | With rbb                                                                                                                                                                                                        |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Template Tags</b> | <code>{{#ds.seas}}{{ds.seas.name}} ; {{/ds.seas}}</code>                                                                                                                                                                                                              | <code>{{#ds.seas:rbb()}}{ds.seas.name}} ; {{/ds.seas}}</code>                                                                                                                                                   |
| <b>Output</b>        | <p>Pacific↵</p> <p>Australasian Mediterranean Sea ; ↵</p> <p>Philippine Sea ; ↵</p> <p>Coral Sea ; ↵</p> <p>South China Sea ; ↵</p> <p>↵</p> <p>Atlantic↵</p> <p>Sargasso Sea ; ↵</p> <p>Caribbean Sea ; ↵</p> <p>Mediterranean Sea ; ↵</p> <p>Gulf of Guinea ; ↵</p> | <p>Pacific↵</p> <p>Australasian Mediterranean Sea ; Philippine Sea ; Coral Sea ; South China Sea ; ↵</p> <p>↵</p> <p>Atlantic↵</p> <p>Sargasso Sea ; Caribbean Sea ; Mediterranean Sea ; Gulf of Guinea ; ↵</p> |

```
C#
// Add a list for oceans, and a nested table for seas:
doc.Body.Paragraphs.Add("{{#ds}}{{ds.name}}", doc.Styles[BuiltInStyleId.ListParagraph]);

//Add new paragraph to define seas range with rbb() formatter.
//This formatter force range to be expanded in single paragraph, instead of generating new paragraph
//per each item.
doc.Body.Paragraphs.Add("{{#ds.seas:rbb()}}{ds.seas.name}} ; {{/ds.seas}}");

//close parent range #ds.
doc.Body.Paragraphs.Add("{{/ds}}");
```

### Limitations

- Range, defined with rbb formatter, can be defined only in single paragraph.
- If rbb range contain inner ranges, they are converted to rbb ranges too, unless they already are rbb ranges.
- pbb range can not be defined inside an rbb range.

## Data Source Binding

Once the template layout is created, it needs to be bound to a data source to populate actual data in the Word document upon template processing. You can add a data source by using [DataSources](#) property of [DataTemplate](#) class.

DsWord supports the following data sources in Report Templates:

### DataTable

A single table which has collection of rows and columns from any type of database.

#### Template syntax

[Alias of data source].[Column name]

For example:

{{ds.ID}}

{{ds.Name}}

#### Bind DataTable datasource

C#

```
var datasource = new System.Data.DataTable();
datasource.Columns.Add(new DataColumn("ID", typeof(Int32)));
datasource.Columns.Add(new DataColumn("Name", typeof(string)));
datasource.Columns.Add(new DataColumn("Score", typeof(Int32)));
datasource.Columns.Add(new DataColumn("Team", typeof(string)));
//Add data source
document.DataTemplate.DataSources.Add("ds", datasource);
```

### DataSet

A collection of one or more DataTables from any type of database.

#### Template syntax

[Alias of data source].[Table name].[Column name]

For example:

{{ds.Table1.ID}}

{{ds.Table2.Team}}

#### Bind DataSet datasource

C#

```
var dTable1 = new System.Data.DataTable();
var dTable2 = new System.Data.DataTable();
...//Init data
var datasource = new System.Data.DataSet();
datasource.Tables.Add(team1);
datasource.Tables.Add(team2);
//Add data source
document.DataTemplate.DataSources.Add("ds", datasource);
```

## Custom Object

A user-defined object from user code or serialized object of JSON String, File or XML etc. DsWord Template supports any data source that can be serialized as a custom object.

### Template syntax

[Alias of data source].[Field path]

or

[Alias of data source].[Property path]

For example:

```
{{ds.Records.Area}}
```

```
{{{ds.Records.Product}}}
```

### Bind Custom Object datasource

C#

```
var datasource = new SalesData
{
 Records = new List<SalesRecord>()
};
var record1 = new SalesRecord
{
 Area = "NorthChina",
 Salesman = "Hellen",
 Product = "Apple",
 ProductType = "Fruit",
 Sales = 120
};
datasource.Records.Add(record1);
var record2 = new SalesRecord
{
 Area = "NorthChina",
 Salesman = "Hellen",
 Product = "Banana",
 ProductType = "Fruit",
 Sales = 143
};
datasource.Records.Add(record2);
...//Init data
//Add data source
document.DataTemplate.DataSources.Add("ds", datasource);
```

## JSON

JSON objects (or JSON text) can be used as data source in DsWord Report Templates.

### Template syntax

[Alias of data source].[Field path]

For example:

```
{{ds.ID}}
```

{{ds.Name}}

### Bind JSON datasource

The template engine allows you to add the following object types as JSON data sources:

- Newtonsoft.Json.Linq.JToken
- Newtonsoft.Json.JsonReader
- System.Text.Json.JsonElement
- System.Text.Json.JsonDocument
- System.ReadOnlyMemory<Byte> (parsed as System.Text.Json.JsonDocument)
- System.ReadOnlyMemory<Char> (parsed as System.Text.Json.JsonDocument)
- System.ReadOnlySequence<Byte> (parsed as System.Text.Json.JsonDocument)
- System.IO.StreamReader (parsed as Newtonsoft.Json.Linq.JToken)
- System.IO.Stream (parsed as Newtonsoft.Json.Linq.JToken)
- System.String (It can be a JSON formatted string or a file name that contains JSON formatted string. It will be parsed as Newtonsoft.Json.Linq.JToken. If the string does not contain valid JSON, it will be interpreted as the path name of a json file).

The below example code shows all the ways in which a JSON datasource can be bound to a template to populate data in the final Word documents.

C#

```
// json file as data source
public void JsonFileDataSource()
{
 var doc = new GcWordDocument();
 doc.Load(@"c:\template.docx");
 doc.DataTemplate.DataSources.Add("ds", @"c:\data.json");
 doc.DataTemplate.Process();
 doc.Save(@"c:\json.docx");
}

// json string as data source
public void JsonStringDataSource()
{
 var doc = new GcWordDocument();
 doc.Load(@"c:\template.docx");
 doc.DataTemplate.DataSources.Add("ds", @"{"name": "David"}");
 doc.DataTemplate.Process();
 doc.Save(@"c:\json.docx");
}

// json stream as data source
public void JsonStreamDataSource()
{
 var doc = new GcWordDocument();
 doc.Load(@"c:\template.docx");
 using (Stream stream = File.OpenRead(@"c:\data.json"))
 {
 doc.DataTemplate.DataSources.Add("ds", stream);
 }
 doc.DataTemplate.Process();
 doc.Save(@"c:\json.docx");
}
```

```

}

// json stream reader as data source
public void JsonStreamReaderDataSource()
{
 var doc = new GcWordDocument();
 doc.Load(@"c:\template.docx");
 using (StreamReader reader = File.OpenText(@"c:\data.json"))
 {
 doc.DataTemplate.DataSources.Add("ds", reader);
 }
 doc.DataTemplate.Process();
 doc.Save(@"c:\json.docx");
}

// json document as data source
public void JsonDocumentDataSource()
{
 var doc = new GcWordDocument();
 doc.Load(@"c:\template.docx");
 System.Text.Json.JsonDocument jdoc = System.Text.Json.JsonDocument.Parse(
{"name": "David"});
 doc.DataTemplate.DataSources.Add("ds", jdoc);
 doc.DataTemplate.Process();
 doc.Save(@"c:\json.docx");
}

// json token as data source
public void JsonTokenDataSource()
{
 var doc = new GcWordDocument();
 doc.Load(@"c:\template.docx");
 Newtonsoft.Json.Linq.JToken jtoken = Newtonsoft.Json.Linq.JToken.Parse(
{"name": "David"});
 doc.DataTemplate.DataSources.Add("ds", jtoken);
 doc.DataTemplate.Process();
 doc.Save(@"c:\json.docx");
}

```

The template and output for JSON string, document and token code sample above will look like below:

```
{{#ds}}{{ds.name}:toupper()}{/ds}}
```

↓  
DAVID

## List and Array

A list is a collection of heterogeneous data and elements while an array is a homogeneous collection of elements ordered in a sequence. In arrays and lists of unnamed entities of primitive types (`Type.IsPrimitive = true`) and strings, value of current element is represented by a virtual property named **value**.

### Template syntax

[[Alias of data source].value] OR [#Alias of data source][[Alias of data source].value][[/Alias of data source]]

For example:

{{ds.value}} OR {{#ds}}{{ds.value}}{/ds}

### Bind List and Array datasource

C#

```
var doc = new GcWordDocument();
// Add a simple List<String> as data source:
doc.DataTemplate.DataSources.Add("dsStrs", new List<string>()
{
 "Pacific",
 "Atlantic",
 "Indian",
 "Southern",
 "Arctic",
});

// Add a simple array as data source:
doc.DataTemplate.DataSources.Add("dsInts", new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 });

// Add a list template so that the data is formatted as a list:
var myListTemplate = doc.ListTemplates.Add(BuiltInListTemplateId.BulletDefault,
"myListTemplate");

// Strings (oceans) List:
doc.Body.Paragraphs.Add("List of Strings (oceans):",
doc.Styles[BuiltInStyleId.Heading1]);
var p = doc.Body.Paragraphs.Add("#{dsStrs}{{dsStrs.value}:toupper()}{/dsStrs}",
doc.Styles[BuiltInStyleId.ListParagraph]);
p.ListFormat.Template = myListTemplate;
```

### Variable

A variable is a number, or a quantity that changes with time and can take different values in different situations.

### Template syntax

[Alias of data source]

For example:

{{ds}}

### Bind Variable datasource


C#

```
var doc = new GcWordDocument();

// Add a variable as data sources:
string str = "Data Record";
doc.DataTemplate.DataSources.Add("dsConsts", str);
```

```
// Add a list template so that the data is formatted as a list:
var myListTemplate = doc.ListTemplates.Add(BuiltInListTemplateId.BulletDefault,
"myListTemplate");

// Variable:
doc.Body.Paragraphs.Add("Variable Data:", doc.Styles[BuiltInStyleId.Heading1]);
var p = doc.Body.Paragraphs.Add("#{#dsConsts}{{dsConsts.value}}{/dsConsts}",
doc.Styles[BuiltInStyleId.ListParagraph]);
p.ListFormat.Template = myListTemplate;
```

 **Note:**

- If template contains a field that does not exist in the bound data source, 'KeyNotFoundException' is thrown by default. However, this behavior can be controlled by setting **MissingFieldsHandling** property of the `DataTemplateOptions` class. Default value of this property is **Strict** which results into the aforementioned exception. To ignore the missing fields, you can set the property to **Relaxed**.
- When `MissingFieldsHandling` property is set to `Relaxed`, the missing range templates are interpreted as empty ranges. To hide these empty ranges which are exposed as a single element, you can use the **hide-block-if-empty()** (**hbi-empty()**) formatter. For more information about formatters, see [Formatters](#).

## Template Validation

When processing a template, there may be instances of errors. It may be difficult to identify all errors at once, and it may take a long time to fix one error at a time. DsWord provides [TemplateError](#) class and [Validate](#) method in [DataTemplate](#) class that enable a user to validate the templates in a document, defining which templates are malformed or wrongly used in the structure. When the user sets `Validate` to `True`, it marks erroneous templates in the document with the error information. When the user sets `Validate` to `False`, it only returns a list of errors.

Refer to the following example code to validate the template:

```
C#

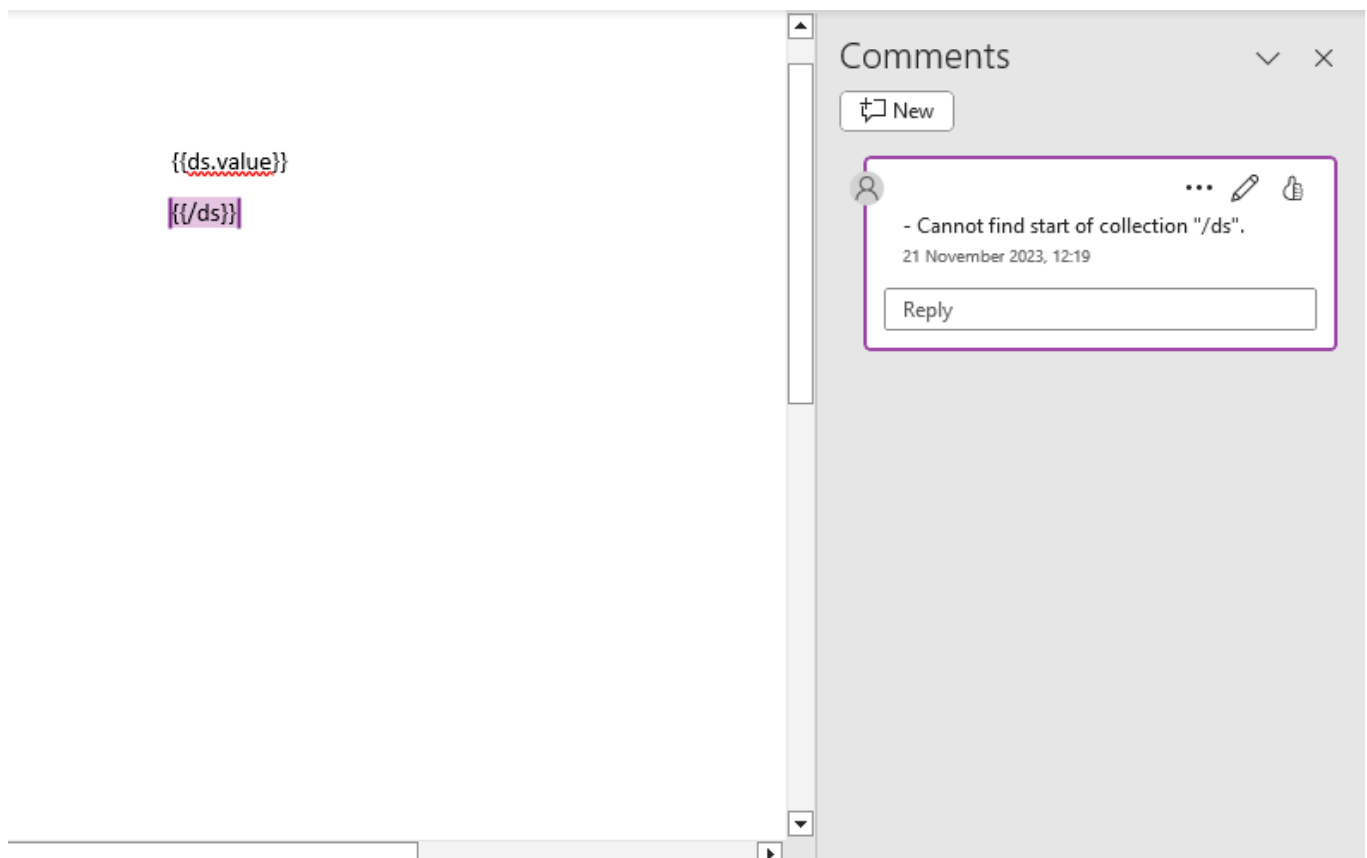
// Initialize GcWordDocument.
var doc = new GcWordDocument();

// Add template data source.
doc.DataTemplate.DataSources.Add("ds", new int[] { 1, 2, 3, 4, 5 });

// Add template tags.
doc.Body.Paragraphs.Add("{{ds.value}}");
doc.Body.Paragraphs.Add("{{/ds}}");

// Validate the template.
var errors = doc.DataTemplate.Validate(true).ToList();

// Save the Word document.
doc.Save("Validate.docx");
```





## Limitations

Validate method only identifies wrong template grammar and absent datasource path errors.

## Walkthrough

Follow the walkthrough, which is a step-by-step tutorial, to understand how to generate Word documents using Report templates.

- [Generate Document with Advanced Layout](#)
- [Generate Document using JSON Data Source](#)
- [Generate Document using Multiple Data Sources](#)

## Generate Document with Advanced Layout

The [Process](#) or [BatchProcess](#) method of [DataTemplate](#) class can be used to process templates in DsWord and generate Word documents with advanced layouts.

The **Process** method processes the template layout by replacing the template tags with actual data from the datasource. Whereas, the **BatchProcess** method processes the template layout by iterating over the root items of a data source in a loop to generate separate documents for each data source item. For example, generating multiple offer letters for different candidates or airline tickets for different travellers by using the same template layout.

### Create a Word Document using Process method

The **Process** method can be used to process the template layout and save the data to a single Word document. It replaces the template tags in a document with actual data from data source and generates a final Word Document with desired data.

This following example uses Process method to generate a Word document where a House Rental Agreement containing the information of landlord, tenant and rental charges needs to be created. The static content in the agreement is generic and can be used as it is, time and again. The dynamic fields are specified as placeholders and can be updated every time a new agreement is to be signed. The template layout is created in Word and is populated with actual data after binding with data source which, in this case, is an XML file.

The below steps describe how to generate a House Rental Agreement using a template in Word. You can also download the [Template layout](#) here.

1. Create a template layout in a Word document and define the following fields:
  - o **Static Fields:** Define the static fields in template layout, that is, the fields whose values will remain constant in the final document. For example header fields and information labels like 'Landlord Information', 'Rental Information', 'Phone Number', 'Email', 'Rent Amount per Month' etc.
  - o **Bound Fields:** Specify the datasource bound fields in mustache braces `{{ }}` like `{{landlordName}}`, `{{collectedBy}}`, `{{tenantEmail}}` etc.

## House Rental Agreement

| Landlord Information                                                                                                                                           |                              | Tenant Information                                |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|---------------------------------------------------|--|
| <b>Name</b><br>{{landlordName}}                                                                                                                                |                              | <b>Name</b><br>{{tenantName}}                     |  |
| <b>Address</b><br>{{landlordAddrLine1}}, {{landlordAddrLine2}}<br>{{landlordAddrCity}}, {{landlordAddrState}},<br>{{landlordAddrPostal}}<br>United States      |                              | <b>Email</b><br>{{tenantEmail}}                   |  |
| <b>Phone Number</b><br>{{landlordPhone}}                                                                                                                       |                              | <b>Phone Number</b><br>{{tenantPhone}}            |  |
|                                                                                                                                                                |                              | <b>Number of Occupants</b><br>{{tenantOccupants}} |  |
| Rental Information                                                                                                                                             |                              |                                                   |  |
| <b>House Rental Address</b><br>{{rentalAddrLine1}}, {{rentalAddrLine2}},<br>{{rentalAddrCity}}, {{rentalAddrState}},<br>{{rentalAddrPostal}},<br>United States | <b>Rent Amount per Month</b> | {{rentalAmntPerMonth}:format(C)}                  |  |
| <b>Start Date of Agreement</b><br>{{rentalStart}:format(D)}                                                                                                    | <b>Security Deposit</b>      | {{rentalSecurityDep}:format(C)}                   |  |
| <b>End Date of Agreement</b><br>{{rentalEnd}:format(D)}                                                                                                        | <b>Late Charges</b>          | {{rentalLateCharge}:format(C)}                    |  |
| <b>Date of First Payment Due</b><br>{{rentalFirstPayment}:format(D)}                                                                                           | <b>Payment Method</b>        | {{paymentMethod}}                                 |  |
| <b>Pay Period:</b> {{rentalPayPeriod}}                                                                                                                         | <b>Collected by</b>          | {{collectedBy}}                                   |  |

2. Load the template created in above step, in DsWord.

```
C#
GcWordDocument doc = new GcWordDocument();
//Load template layout
doc.Load("RentalAgreement[Template].docx");
```

3. Initialize a DataSet as data source and read data from an XML file. After adding the XML as dataset, add one of the tables as datasource for the template.

```
C#
//Initialize DataSet
```

```
var ds = new System.Data.DataSet();
//Read data from xml
ds.ReadXml(Path.Combine("DsWordTplDataSet.xml"));
//Add datasource
doc.DataTemplate.DataSources.Add("ds", ds.Tables["HouseRentalAgreement"]);
```

The data source used in this sample (DsWordTplDataSet.xml) can also be downloaded from Resources/data folder of [this project](#).

4. Process the template by using **Process** method.

```
C#

//Process the template
doc.DataTemplate.Process();
```

5. Save the final Word document.

```
C#

//Save the final document
doc.Save("RentalAgreement.docx");
```

The output of the House Rental Agreement is shown as below:

## House Rental Agreement

| Landlord Information                                                                                              |                              | Tenant Information                    |                |
|-------------------------------------------------------------------------------------------------------------------|------------------------------|---------------------------------------|----------------|
| <b>Name</b><br>Jillene Holtaway                                                                                   |                              | <b>Name</b><br>Nadine Rousell         |                |
| <b>Address</b><br>725 Barby Junction, 85 Grayhawk Center<br>San Antonio, Texas, 78220<br>United States            |                              | <b>Email</b><br>aibrahim@example.com  |                |
| <b>Phone Number</b><br>(192) 472-5503                                                                             |                              | <b>Phone Number</b><br>(123) 456-7890 |                |
|                                                                                                                   |                              | <b>Number of Occupants</b><br>2       |                |
| Rental Information                                                                                                |                              |                                       |                |
| <b>House Rental Address</b><br>321 Ken Junction, 12 Blueowl Plaza,<br>San Antonio, Texas, 33322,<br>United States | <b>Rent Amount per Month</b> |                                       | \$3,200.00     |
| <b>Start Date of Agreement</b><br>Thursday, December 31, 2020                                                     | <b>Security Deposit</b>      |                                       | \$5,000.00     |
| <b>End Date of Agreement</b><br>Saturday, December 31, 2022                                                       | <b>Late Charges</b>          |                                       | \$767.00       |
| <b>Date of First Payment Due</b><br>Friday, January 1, 2021                                                       | <b>Payment Method</b>        |                                       | Option 2       |
| <b>Pay Period:</b> 1 month                                                                                        | <b>Collected by</b>          |                                       | Nadine Roussel |

### Create Separate Word Documents using BatchProcess Method

The **BatchProcess** method can be used to process the template layout and save multiple rows of data into separate Word documents.

The following example saves data of different employees to separate Word documents. The template layout is created in Word and is populated with actual data after binding with data source. You can also download the [Template layout](#) here.

1. Create a template layout in a Word document and define the following fields:
  - o **Static Fields:** Define the static fields in template layout, that is, the fields whose values will remain constant in the final document. For example Name, Designation, Years of Experience etc.
  - o **Bound Fields:** Specify the datasource bound fields in mustache braces {{ }} like {{name}}, {{designation}}, {{experience}} etc.

Name: {{name}}  
Designation: {{designation}}  
Department: {{department}}  
Years of Experience: {{experience}}

2. Load the template created in above step, in DsWord.

```
C#
GcWordDocument doc = new GcWordDocument();
//Load template layout
doc.Load("EmployeeData[Template].docx");
```

3. Add data which will be used as data source by using **DataSources** method of **DataTemplate** class.

```
C#
doc.DataTemplate.DataSources["ds"] = new object[] {
 new {
 name = "Derek Clark",
 designation = "HOD",
 department = "marketing",
 experience = "23"
 },
 new {
 name = "Jessica Adams",
 designation = "Senior Executive",
 department = "sales",
 experience = "5"
 },
 new {
 name = "Anil Mittal",
 designation = "Software Engineer",
 department = "development",
 experience = "7"
 }
};
```

4. Process the template by using **BatchProcess** method and save the Word documents separately.

```
C#
// produces a document for each root item in the data source collection
int i = 0;
doc.DataTemplate.BatchProcess(() =>
{
 doc.Save($"Employee-Data-{{++i}}.docx");
});
```

The output of Employee Report Templates is saved to separate documents as shown below:

## [Employee-Data-1.docx](#)

Name: Derek Clark

Designation: HOD

Department: marketing

Years of Experience: 23

## [Employee-Data-2.docx](#)

Name: Jessica Adams

Designation: Senior Executive

Department: sales

Years of Experience: 5

## [Employee-Data-3.docx](#)

Name: Anil Mittal

Designation: Software Engineer

Department: development

Years of Experience: 7

### **Limitation**

The template layout document with multiple sections is not batch processed correctly. Hence, in case of batch processing, keep the template layout in a single section only.



## Generate Document using JSON Data Source

Report Templates allow you to bind the template with JSON data source by using **Add** method of **IDictionary** interface. The data can be added from a JSON file, stream, string etc. and bound to populate actual data upon template processing. For more information on different ways to work with JSON data source, refer [Data Source Binding](#).

The following example takes you through a step by step process to generate a Word document with the list of oceans provided as a JSON string.

1. Create a template layout in a Word document as shown below. You can also download the [Template layout](#) from here.

### List of Oceans

```
{{#ds}}{{ds.seas.name}:toupper(){{/ds}}
```

2. Load the template created in above step, in DsWord.

```
C#
GcWordDocument doc = new GcWordDocument();
//Load template layout
doc.Load("OceansList[Template].docx");
```

3. Pass the data in a JSON string.

```
C#
var oceans = @"[
 {
 ""name"": ""Pacific"",
 ""areaOfWorldOcean"": 0.466,
 ""volumeOfWorldOcean"": 0.501,
 ""seas"": [
 {""name"": ""Australasian Mediterranean Sea""},
 {""name"": ""Philippine Sea""},
 {""name"": ""Coral Sea""},
 {""name"": ""South China Sea""}],
 },
 {
 ""name"": ""Atlantic"",
 ""areaOfWorldOcean"": 0.235,
 ""volumeOfWorldOcean"": 0.233,
 ""seas"": [
 {""name"": ""Sargasso Sea""},
 {""name"": ""Caribbean Sea""},
 {""name"": ""Mediterranean Sea""},
 {""name"": ""Gulf of Guinea""}],
 },
 {
 ""name"": ""Indian"",
```

```
 ""areaOfWorldOcean"": 0.195,
 ""volumeOfWorldOcean"": 0.198,
 ""seas"": [
 {""name"": ""Arabian Sea""},
 {""name"": ""Bay of Bengal""},
 {""name"": ""Andaman Sea""},
 {""name"": ""Laccadive Sea""}],
 },
 {
 ""name"": ""Southern"",
 ""areaOfWorldOcean"": 0.061,
 ""volumeOfWorldOcean"": 0.054,
 ""seas"": [
 {""name"": ""Weddell Sea""},
 {""name"": ""Somov Sea""},
 {""name"": ""Riiser-Larsen Sea""},
 {""name"": ""Lazarev Sea""}],
 },
 {
 ""name"": ""Arctic"",
 ""areaOfWorldOcean"": 0.043,
 ""volumeOfWorldOcean"": 0.014,
 ""seas"": [
 {""name"": ""Barents Sea""},
 {""name"": ""Greenland Sea""},
 {""name"": ""East Siberian Sea""},
 {""name"": ""Kara Sea""}],
 }
];
```

4. Add above data as a data source for the template.

C#

```
doc.DataTemplate.DataSources.Add("ds", oceans);
```

5. Process the template by using **Process** method.

C#

```
//Process the template
doc.DataTemplate.Process();
```

6. Save the final Word document.

C#

```
//Save the final document
doc.Save("List of Oceans.docx");
```

The output of the above code is shown as below:

## List of Oceans

AUSTRALASIAN MEDITERRANEAN SEA

PHILIPPINE SEA

CORAL SEA

SOUTH CHINA SEA

SARGASSO SEA

CARIBBEAN SEA

MEDITERRANEAN SEA

GULF OF GUINEA

ARABIAN SEA

BAY OF BENGAL

ANDAMAN SEA

LACCADIVE SEA

WEDDELL SEA

SOMOV SEA

RIISER-LARSEN SEA

LAZAREV SEA

BARENTS SEA

GREENLAND SEA

EAST SIBERIAN SEA

KARA SEA

## Generate Document using Multiple Data Sources

You can use multiple data sources with Report Templates to generate Word documents upon template processing. The following example demonstrates how to generate a Word document with the list of car brands followed by the list of car body styles by using two data sources.

1. Define data in two JSON datasources, containing:
  - o A list of some popular car makers
  - o A list of various car body styles
2. Create a new Word document by instantiating the **GcWordDocument** class.
3. Add both datasources with specified data by using **Add** method of the **IDictionary** interface.
4. Add template tags in a paragraph by using **Add** method of the **ParagraphCollection** class.
5. Process the template by using **Process** method and save the final Word document.

C#

```
{
var makes = "[" +
 "{ \"make\": \"Toyota\" }," +
 "{ \"make\": \"General Motors\" }," +
 "{ \"make\": \"Volkswagen\" }," +
 "{ \"make\": \"Ford\" }," +
 "{ \"make\": \"BMW\" }," +
 "{ \"make\": \"Nissan\" }," +
 "{ \"make\": \"Hyundai\" }," +
 "{ \"make\": \"Honda\" }," +
 "{ \"make\": \"Mazda\" }," +
 "{ \"make\": \"Jaguar\" }," +
 "];

var bodyStyles = "[" +
 "{ \"style\": \"Sedan\" }," +
 "{ \"style\": \"Coupe\" }," +
 "{ \"style\": \"Hatchback\" }," +
 "{ \"style\": \"SUV\" }," +
 "{ \"style\": \"Crossover\" }," +
 "{ \"style\": \"Minivan\" }," +
 "{ \"style\": \"Pickup\" }," +
 "{ \"style\": \"Wagon\" }," +
 "];

var doc = new GcWordDocument();

// Add the data sources
doc.DataTemplate.DataSources.Add("makes", makes);
doc.DataTemplate.DataSources.Add("styles", bodyStyles);

// Print the list of some car makes
doc.Body.Paragraphs.Add("Popular Car Makers",
doc.Styles[BuiltInStyleId.Heading2]);
doc.Body.Paragraphs.Add("#{#makes}{{makes.make}}{/makes}",
doc.Styles[BuiltInStyleId.ListBullet]);

// Print the list of car body styles
```

```
doc.Body.Paragraphs.Add("Car Body Styles",
doc.Styles[BuiltInStyleId.Heading2]);
doc.Body.Paragraphs.Add("{{#styles}}{{styles.style}}{/styles})",
doc.Styles[BuiltInStyleId.ListBullet]);

doc.DataTemplate.Process();
doc.Save("MultipleDataSources.docx");
```

The output of above code will look like below in the Word document:

## Popular Car Makers

- Toyota
- General Motors
- Volkswagen
- Ford
- BMW
- Nissan
- Hyundai
- Honda
- Mazda
- Jaguar

## Car Body Styles

- Sedan
- Coupe
- Hatchback
- SUV
- Crossover
- Minivan
- Pickup
- Wagon

### Limitation

While using different datasources, there may be identical paths (like `{{ds1.employee.name}}` and `{{ds2.employee.name}}`). Hence, the fully qualified template tag should be used which includes the name of datasource. The unqualified tags (like `{{employee.name}}`) will throw an exception.

## Samples

All the DsWord samples are available through the online [sample browser](#). You can browse the source code of samples, run them on the server, download the generated DOCX and view the generated PDF online, or download individual samples to build and run on your own system (Windows, Mac or Linux). For more information, see [Quick Start](#), introductory page for the samples.

If you choose to download the samples, you can run them using following simple steps:

1. Click the **Download** action on the top right of the sample page.
2. Unzip the downloaded .zip file of sample.
3. Run the sample.

## API Reference

This section contains documentation for all the assemblies required to create applications using DsWord.

| Assembly                          | Description                                                                                                            |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <a href="#">DS.Documents.Word</a> | Cross-platform library for creating, analyzing, and modifying Office Word documents without dependencies on MS Office. |

## Release Notes

Refer to the following release notes for the major releases of the product.

- [Breaking Changes](#)
- [Release Notes for Version 7.1.0](#)
- [Release Notes for Version 7.0.0](#)
- [Release Notes for Version 6.2.0](#)
- [Release Notes for Version 6.1.0](#)
- [Release Notes for Version 6.0.0](#)
- [Release Notes for Version 5.2.0.800](#)
- [Release Notes for Version 5.1.0.790](#)
- [Release Notes for Version 5.0.0.762](#)
- [Release Notes for Version 4.2.0.719](#)
- [Release Notes for Version 4.2.0.715](#)
- [Release Notes for Version 4.1.0.658](#)
- [Release Notes for Version 4.0.0.616](#)
- [Release Notes for Version 3.2.0.548](#)
- [Release Notes for Version 3.1.0.508](#)
- [Release Notes for Version 3.0.0.414](#)
- [Release Notes for Version 2.2.0.310](#)
- [Release Notes for Version 2.1.0.260](#)

For details about latest hotfixes, see the [nuget page](#).



## Breaking Changes

Refer to the breaking changes for specific versions:

- [Version 5.0.0.762](#)
- [Version 4.2.0.715](#)
- [Version 4.0.0.632 \(Maintenance\)](#)
- [Version 3.1.0.508](#)
- [Version 3.0.0.414](#)
- [Version 2.2.0.310](#)
- [Version 2.1.0.260](#)

## Version 7.1.0

### New Features and Improvements

- When adding an SVG image, a complementary raster copy of the image is also added to the document.
- When a CheckBox or a Date content control (with CalendarType.Gregorian) is changed, its appearance is automatically updated.
- Added ImageData.ComplementaryVectorData property: Gets the complementary vector data of the image.
- Added VectorGraphicImageData class: Represents the image data of an SVG vector image.
- Added WebVideoProperties class: Represents the properties used to show an online video to the user.
- Added ImageData.WebVideoProperties property: Gets the properties for displaying an online video to the user.

## Version 7.0.0

### Important Note

This is the initial release of the DS.Documents.Word package. This package replaces GrapeCity.Documents.Word, and provides the same functionality, ensures future enhancements, and is backwards compatible with GrapeCity.Documents.Word. Existing subscriptions will continue to apply to this new package.

### New Features and Improvements

- Added `ContentObject.CanAdd()` method to check whether a `ContentObject` of the specified type can be added to the current object.
- Added `ContentObject.CanAddContentControl()` method to check whether a `ContentControl` of the specified type can be added to the current object.
- Added `Body.CanAdd()` method to check whether a `ContentObject` of the specified type can be added to the current body.
- Added `Body.CanAddContentControl()` method to check whether a `ContentControl` of the specified type can be added to the current body.
- Added `Section.CanAdd()` method to check whether a `ContentObject` of the specified type can be added to the current section.
- Added `Section.CanAddContentControl()` method to check whether a `ContentControl` of the specified type can be added to the current section.
- Added the following support for picture effects:
  - [Picture effects] `RecolorType` enum: specifies the type of an image color change.
  - [Picture effects] `ImageEffectType` enum: specifies the image effect type.
  - [Picture effects] `UserColor` class: represents an extended color.
  - [Picture effects] `WebVideoProperties` class: represents properties for displaying an online video to the user.
  - [Picture effects] `ImageEffect` class (derived from `FormattingBag`): the abstract base class for image effects.
  - [Picture effects] `AlphaBiLevel` class: represents an Alpha bi-level effect.
  - [Picture effects] `AlphaCeiling` class: represents an alpha ceiling effect.
  - [Picture effects] `AlphaFloor` class: represents an alpha floor effect.
  - [Picture effects] `AlphaInverse` class: represents an alpha inverse effect.
  - [Picture effects] `AlphaModulation` class: represents an alpha modulate fixed effect.
  - [Picture effects] `AlphaModulationComplex` class: represents an alpha modulate complex effect.
  - [Picture effects] `AlphaReplace` class: represents an alpha replace effect.
  - [Picture effects] `BiLevel` class: represents a bi-level (black/white) effect.
  - [Picture effects] `ColorChange` class: represents a color change effect.
  - [Picture effects] `ColorReplacement` class: represents a solid color replacement effect.
  - [Picture effects] `Duotone` class: represents a duotone effect.
  - [Picture effects] `Grayscale` class: represents a grayscale effect.
  - [Picture effects] `HslEffect` class: represents a hue, saturation, or luminance effect.
  - [Picture effects] `Luminance` class: represents a luminance effect.
  - [Picture effects] `Tint` class: represents a tint effect.
  - [Picture effects] `ImageEffectList` class: represents a list of image effects.
  - [Picture effects] `EmbeddedImageData` class: represents embedded image data.
  - [Picture effects] New members have been added to the `PicturePreset` enum.
  - [Picture effects] New members have been added to the `ImageData` class.
- Updated `OpenXml` library version to 2.20.
- [Picture effects] `Blur` and `FillOverlay` classes now derive from `ImageEffect` (previously they derived directly from `FormattingBag`).

## Version 6.2.0

### New Features and Improvements

- Added API supporting Office Math.
- Added OMath class: Represents inline math content.
- Added OMathParagraph class: Represents math content on its own line as a paragraph. Can contain one or more OMath instances.
- Added OMathStruct abstract base class: Used to represent a mathematical structure inside an OMath instance.
- Added Classes derived from OMathStruct: OMathBar, OMathBoundingBox, OMathBox, OMathDelimiter, OMathEquationArray, OMathFraction, OMathFunction, OMathGroupCharacter, OMathLimitLower, OMathLimitUpper, OMathMatrix, OMathNary, OMathPhantom, OMathPreSubSuperscript, OMathRadical, OMathSubscript, OMathSubSuperscript, OMathSuperscript.
- Added OMathElement class: Used to build specific structures in classes derived from OMathStruct.
- Added helper methods to add various types of content to parts of a document. Methods are defined for different part types (Paragraph, Run, OMath etc.) and ensure that only allowed content types can be added using these methods. Examples are Paragraph.AddRun(), Cell.AddTable(), OMath.AddHyperlink() etc.
- Added Classes containing helper methods to add allowed content: BidirectionalOverride, Body, CanvasShape, Cell, ControlContent, GroupShape, Hyperlink, OMath, OMathElement, OMathParagraph, Paragraph, Row, Run, Section, SimpleField, Table, TextFrame.
- Added ability to escape template tags: Prepend a valid template tag with a backslash to exclude it from template processing.

## Version 6.1.0

### New Features and Improvements

- Added support for linked (paragraph and character) styles.
- Added `DsWordDocument.HideLinkedCharacterStyles` property that hides linked character styles from the DsWord API, allowing users to use the paragraph part of a linked style pair in place of both styles.
- Added `StyleCollection.AddLinkedStyle` method that adds a pair of linked paragraph and character styles to the collection.
- Added support for 3D effects on shapes and text.
- Added `BevelType` enum that specifies the preset bevel type, which defines the look of a bevel.
- Added `LightRigType` enum that specifies the preset type of the light rig that is to be applied to the scene.
- Added `LightRigDirection` enum that specifies the direction from which the light rig is oriented in relation to the scene.
- Added `CameraPreset` enum that specifies the preset camera that defines a starting point for common preset rotations in space.
- Added `MaterialType` enum that specifies the preset material type to give the final look and feel of an object.
- Added `FontEffectsPreset` enum that specifies the built-in font effects preset.
- Added `ShapeEffectsPreset` enum that specifies the built-in shape effects preset.
- Added `IFixedFormattingBag` interface that defines properties and methods for working with inherited and direct formatting.
- Added `Point3D` class that represents a point in 3D space.
- Added `Vector3D` class that represents a vector in 3D space.
- Added `Rotation` class that represents a rotation in 3D space.
- Added `Bevel` class that represents a bevel on an object face.
- Added `ColoredLine` class that represents a colored line.
- Added `Backdrop` class that represents a plane in which effects, such as glow and shadow, are applied in relation to the shape they are applied to.
- Added `Lighting` class that represents a light rig associated with an object.
- Added `Camera` class that represents the placement and properties of a camera in a 3D scene.
- Added `ThreeDScene` class that represents a 3D scene effect that can be applied to an object.
- Added `ThreeDFormat` class that represents a 3D format effect that can be applied to an object.
- Added `TextEffects.ThreeDFormat` property that gets the 3D format effect.
- Added `TextEffects.ThreeDScene` property that gets the 3D scene effect.
- Added `TextEffects.Clean()` method that sets all effects to default values.
- Added `ShapeEffects.Clean()` method that sets all effects to default values.
- Added `IFixedFormattingBag` implementation to `FillFormat`, `LineFormatBase`, `OuterShadow`, `Reflection`, and `Glow` classes.
- Added `ApplyEffectsPreset()` method to the `Font`, `Shape`, `GroupShape`, and `Picture` classes.
- Added `TextFrameFormat.ThreeDFormat` property that gets the 3D format effect.
- Added `TextFrameFormat.ThreeDScene` property that gets the 3D scene effect.
- Added `TextFrameFormat.FlatText` property that indicates whether to keep text out of the 3D scene.
- Added state functions: `IsFirst()`, `IsLast()`, and `Index()`.
- Added `Format(expression[, format_string[, culture]])` calc function.
- Added `lif(condition, true_expression, false_expression)` calc function.
- Aggregate calc functions now can have complex arguments, including expressions and function calls.

## Version 6.0.0

### New Features and Improvements

- Added support for reflection effect in Font, Shape, Picture, GroupShape, and CanvasShape using TextEffects and ShapeEffects classes.
- Added support for glow effect in Font, Shape, Picture, GroupShape, and CanvasShape using TextEffects and ShapeEffects classes.
- Added support for blur effect in Shape, Picture, GroupShape, and CanvasShape using ShapeEffects class.
- Added support for soft-edge effect in Shape, Picture, GroupShape, and CanvasShape using ShapeEffects class.
- Added support for fill-overlay effect in Shape, Picture, GroupShape, and CanvasShape using ShapeEffects class.
- Added hide-show blocks depending on conditions like `{{if }} a {{else}} b {{endif}}`.
- Updated report templates to remove paragraphs that are generated using template tags.

### Resolved Issues

- Fixed issue where format scheme's images were not serialized while saving documents.

## Version 5.2.0.800

### New Features and Improvements

- Added support for Decimal, DateTime and DateTimeOffset to the primitive types in the report templates.
- Added DataSourceDictionary.Add(string, object, CultureInfo) method in report template to include data source with an associated culture while parsing and formatting data.
- Added shadow effects for text and shapes.
- Added BuiltInShadowId enum that defines the types of built-in shadow effects.
- Added PresetShadowType enum that defines the preset shadow types.
- Added ShadowBase class that represents shadow effect as an object.
- Added PresetShadow class that represents a preset shadow effect as an object.
- Added InnerShadow class that represents an inner shadow effect as an object.
- Added OuterShadow class that represents an outer shadow effect as an object.
- Added TextEffects class that represents a text formatting effect as an object.
- Added ShapeEffects class that represents a shape formatting effect as an object.
- Added ShapeEffectsList class that represents a set of shape styles effects to be used within a theme.
- Added StyleEffects class that represents a reference to shape styles effect of FormatScheme.Effects list.
- Added TextEffects property in Font class which gets the text formatting properties.
- Added ShapeEffects property in Shape class which gets the shape formatting properties.
- Added ShapeEffects property in GroupShape class which gets the shape formatting properties.
- Added ShapeEffects property in CanvasShape class which gets the shape formatting properties.
- Added ShapeEffects property in Picture class which gets the shape formatting properties.
- Added StyleEffects property in ShapeStyle class which gets the reference to style properties in a FormatScheme.Effects list.
- Added ShapeEffectsList property in FormatScheme class which gets a list of shape effects to be used within the theme.

### Changes From the Previous Version

- Improved diagnostics when processing report templates which contained errors.

### Resolved Issues

- In Report Templates, fixed issues which occurred when processing templates using non-US cultures.
- In Report Template, fixed issues which caused errors when parsing valid templates.

## Version 5.1.0.790

### New Features and Improvements

- Added `Font.Fill` property, which gets the fill formatting properties.
- Added `Font.Line` property, which gets the outline text effect formatting properties.
- Added `FillFormat.Reset()` method, which resets the fill format to inherited defaults.
- Added `LineFormatBase.Reset()` method, which resets the line format to inherited defaults.
- Added `SaveOptions` class, which provides various options controlling how a document is saved.
- Added `FontInfoCollection.Add()` and `Append()` methods, which enables adding `GrapeCity.Documents.Text.Font` instances to `FontInfoCollection`, optionally embedding those fonts.
- Added `Settings.SaveOptions` property, which gets the document save options.
- Added `Settings.LocaleName` property, which gets or sets the default locale name (language) of formatted characters.
- Added `Settings.LocaleNameBi` property, which gets or sets the locale name (language) of complex script characters.
- Added `Settings.LocaleNameFarEast` property, which gets or sets the locale name (language) of formatted Asian characters.
- In Report Templates, added `DataTemplateOptions` class, which represents options used in data template processing. In particular, `MissingFieldsHandling` property allows ignoring missing data fields.
- In Report Templates, added `DataTemplate.Options` property, which gets options controlling the behavior of the template processing engine.
- In Report Templates, added `todouble()` formatter, which converts a string value to a double.
- In Report Templates, added `tobool()` formatter, which converts a string value to a Boolean.
- Added `GrapeCity.Documents.Word.Layout.Page.SaveAsSvg()` methods which save the current page to a stream of file in SVG format.
- Added `GrapeCity.Documents.Word.Layout.Page.ToSvgz()` method which saves the page to a byte array in SVGZ format.



## Version 5.0.0.762

### New Features and Improvements

- [Data Templates] Ability to use arrays of elementary types as data sources. The virtual "value" tag expands to the value of an array element.
- Basic support for embedded fonts.
- Property `DocumentBase.Fonts`: gets the collection of `FontInfo` objects associated with this document.
- Several properties added to `ThemeFont` class to support embedded fonts.
- Class `FontInfo`: represents the properties of a font used in a document.
- Class `FontInfoCollection`: represents a collection of `FontInfo` objects.
- Class `FontSignature`: specifies code pages and Unicode subranges for which a font provides glyphs.
- Class `EmbeddedFont`: represents an embedded font and its binary data.
- Class `EmbeddedFontCollection`: represents a collection of `EmbeddedFont` objects.
- Enumeration `FontCharSet`: defines character sets that may be supported by a font.
- Enumeration `FontFamily`: specifies values representing the possible font families.
- Enumeration `FontPitch`: specifies the pitch of a font.
- Enumeration `EmbeddedFontType`: specifies the type of an embedded font.
- Enumeration `FontDataType`: specifies the data type of an embedded font.

### Bug fixes

- [Data Templates] If `FindReplaceOptions.RemoveEmptyRuns` is true, template expansion produces incorrect results.

### Breaking changes

- Removed method overloads that accept `System.Drawing.Image` type, as it is no longer supported on non-Windows systems: `Picture.Add(Image)` and `ImageData.SetImage(Image)`.
- `EditableRangeCollection.Add()` and `EditableRangeCollection.Insert()` method overloads changed.

Breaking changes affecting all `GrapeCity.Documents` packages:

- `GrapeCity.Documents.Common` package has been removed, types defined in it have been moved to `GrapeCity.Documents.Imaging`.
- `GrapeCity.Documents.Common.Windows` package has been replaced by `GrapeCity.Documents.Imaging.Windows`.
- `GrapeCity.Documents.Pdf.Resources` has been removed, types defined in it have been moved to `GrapeCity.Documents.Pdf`.

## Version 4.2.0.719

### New Features and Improvements

- Added Paragraph.RestartList() method that restarts the list numbering on the current paragraph.

## Version 4.2.0.715

### New Features and Improvements

- Added `FormattingCopyStrategy.KeepSource` member which copies styles from the source document unless an identical style that also has the same id exists in the destination document.
- [Data Templates] Support for several data sources.
- [Data Templates] Improved access to sibling data members.
- [Data Templates] Added 'sequence' (or 'seq') and 'follow' pair of formatters that allow iterating over two sequences in parallel.
- [Data Templates] Support for JSON data sources.
- [Data Templates] Added 'restart' list formatter that restarts nested numbered lists.

### Bug fixes

- Miscellaneous bug fixes.

### Breaking changes

- The default value of `RegularExpressions` property belonging to `FindOptions` class (`FindReplaceOptions` class inherits the `FindOptions` class) has been changed from `true` to `false`.

## Version 4.1.0.658

### New Features and Improvements

- Added property `FindReplaceOptions.RemoveEmptyRuns` which indicates whether empty runs should be removed after a replace operation.
- Added `Add()` and `Insert()` method overloads to `PictureCollection` and `ImageData` classes that accept `System.Drawing.Image` and `GrapeCity.Documents.Drawing.Image` types.
- Added `RangeBase.GetInnerCollection<T>()` method which gets an enumerator that does not include objects that start before the current range's start and end after the current range's end.
- Added property `GcWordDocument.MalformedUriRewriter` which specifies the malformed URI rewriter that can be used to implement a custom strategy for rewriting malformed URIs.
- Added `DefaultMalformedUriRewriter` class which implements the default rewriting strategy, can be assigned to `MalformedUriRewriter`.
- Added `GetChildren<T>()` method to `ContentObject`, `ContentRange`, `Body` and `Section` classes which gets the collection of the object's children of a specified type.

### Bug fixes

- Incorrect evaluation of toggle properties that use XOR-based calculation.
- Incorrect indentation of some table rows in exported PDFs.

## Version 4.0.0.616

### New Features and Improvements

- Introduced Data templates feature which provides a flexible mechanism for creating data-bound documents. Template tags are added to the document, and processed using the `GcWordDocument.DataTemplate` property.
- Added `GrapeCity.Documents.Word.Templates.DataTemplate` class which provides properties and methods that are used to associate a `GcWordDocument` with template data sources, and to process data templates.
- Added `DataTemplate.DataSources` property which gets the collection of data sources associated with the current document's data templates. However, only one data source is supported in this version.
- Added `DataTemplate.Process()` method which processes templates in the document, iterating over root items in `DataSources`, and replacing template tags in the document with data.
- Added `DataTemplate.BatchProcess()` method which iterates over root items in `DataSources`, starting with a fresh copy of the document and processing templates separately for each root data item.
- Added `GcWordDocument.DataTemplate` property which gets the `DataTemplate` object that provides properties and methods used to manage template data sources and process data templates.

### Version 4.0.0.632 (Maintenance) - Breaking Changes

- `isdaylightsavingtime` formatter has been removed from Report Templates.

## Version 3.2.0.548

### New Features and Improvements

- Enhanced support for shapes. New classes: CanvasShape, CanvasShapeCollection, InkShape, InkShapeCollection, GroupShape, GroupShapeCollection. Added support for shape styles, presets, line and fill properties.
- Added properties to Picture class: AlternativeText, Hidden, Name, ID.
- Added properties to ImageData class: Stretch, FillType, Tile, RotateWithObject, Dpi, Transparency.
- Implemented SolidColor.Brightness and SolidColor.TintAndShade properties.

## Version 3.1.0.508

### New Features and Improvements

- Added `RangeBase.Find()` and `RangeBase.Replace()` methods that allow to find and replace content and formatting.
- Added `FindFormatting` class that represents the formatting criteria for a find operation.
- Added `FindOptions` class that represents options for a find operation.
- Added `FindReplaceOptions` class that represents options for a find and replace operation.
- Added `FindResult` class that represents the result of a find operation.
- Added `MatchDetails` class that represents the details of a match.
- Added `ReplacingArgs` class that represents the details about a found text.
- Added `ReplacedArgs` class that represents a text range that replaced a search text.
- Added `ReplaceAction` enum that specifies the action to take when a match is found.
- Added `ShapeBase` class, the base class for all types of shapes.
- Added `Shape` class that represents a shape element in a body content.
- Added `ShapeCollection` class that represents a collection of `Shape` objects.
- Added `TextFrame` class that represents the text contents of a `Shape` and associates that textual information, referred to as a text frame story, with a story identifier.
- Added `TextFrameCollection` class that represents a collection of `TextFrame` objects.
- Added `LinkedTextFrame` class that represents a text frame that participates in a text frame story.
- Added `TextFrameFormat` class that represents the format of a text frame.
- Added `TextOrientation` enum that specifies text orientation.
- Added `TextWrapFormat` enum that specifies the preset text shape geometries that can be used for a text frame.
- Added `TextVerticalAnchor` enum that specifies the available vertical anchoring types for text.
- Added `RangeBase.Shapes` property that gets the collection of shapes included in this range.
- Added `RangeBase.TextFrames` property that gets the collection of text frames included in this range.
- Added `RangeBase.GetDrawings()` method that gets the collection of all drawing (based on `ShapeBase` class) objects like picture, shape and other unknown shape types.
- Added `PictureBulletCollection.Add(byte[] imageBytes, string contentType)` that creates a new `PictureBullet` and adds it to the collection.

### Bug fixes

- Fixed incorrect rendering of some tables while exporting from to PDF from `DsWord`.

### Breaking changes

- `Picture` class is now based on `ShapeBase` class (was based on `ContentObject` class).
- Removed public constructor from `ImageData` class.
- Removed method `PictureBulletCollection.Add(ImageData)`

## Version 3.0.0.414

### New Features and Improvements

- Added formatting clear method to RangeBase.
- Added the feature to copy document content inside a document and preserve content formatting.
- Added the feature to move document content inside a document and preserve content formatting.
- Added the feature to copy document content between documents and preserve formatting.
- Added the feature to copy document content between documents and preserve formatting.
- Added the feature to split the document on several documents and preserve formatting.
- Added the feature to combine content from different Word documents into a single one.
- Added DocumentProtection feature.
- Added ReadOnly regions protection feature.
- Added ability to detect source of formatting properties.

### Bug fixes

- Fixed the issue where the Max length of password should be 15 digits.
- Fixed the issue where the password that contains special characters can not stop protection.
- Fixed the issue where WritePassword doesn't recognize current password implementation.

### Breaking changes

- The HMAC hash from Password.KnownHashes cannot be used anymore.
- The API has changed for Split/Merge.
- The CompareLocationWith method was moved from Range to RangeBase.



## Version 2.2.0.310

### New Features and Improvements

The following features have been added with this version of the product.

New classes:

- DocumentBase class - Represents the base class for GcWordDocument and GlossaryDocument classes.
- GlossaryDocument - Represents a supplementary document storage which stores the definition and content to be carried with the document for future insertion and/or use, but which should not be visible within contents of the main document story.
- FormattedMark class - Specifies the set of properties applied to the special content mark
- BuildingBlockCollection class - Represents a collection of building blocks.
- BuildingBlock class - Represents a building block in a document.
- Category class - Specifies the categorization for a building block. This categorization should not imply any behaviors around the building block, and is only used to organize set of building blocks within an application or user interface i.e. to disambiguate between two building blocks with the same entry name.
- CustomXmlPart class - Represents a custom XML data storage in the document.
- CustomXmlPartCollection class - Represents a collection of CustomXmlPart objects in the document.
- LastRenderedPageBreak class - Represents the position delimited the end of a page when this document was last saved by an application which paginates its content.
- ContentControl class - Represents an individual content control. Content controls are bound and potentially labelled regions in a document that serve as containers for specific types of content. Individual content controls may contain contents such as dates, lists, or paragraphs of formatted text.
- ContentControlCollection class - Represents a collection of ContentControl items.
- ControlContent class - Provides access to the content of a ContentControl.
- ContentControlEndMark class - Specifies set of properties applied to the mark present to delimit the end of control contents.
- CheckBoxSymbol class - Specifies a symbol to be used for a checkbox state.
- DropDownListItem class - Specifies a single list item within the content control. Each list item should be displayed in the list for the content control (if a user interface is present).
- DropDownListItemCollection class - Represents a list of DropDownListItem objects.
- XmlMapping class - Specifies the information to be used to establish a mapping between a content control and an XML node stored within a Custom XML Data part in the document.

New enumerations:

- BuildingBlockGallery enum - Specifies a building block gallery.
- BuildingBlockType enum - Specifies the type of a building block.
- BuildingBlockInsertOptions enum - Specifies how a building block is inserted into a document.
- ContentControlType enum - Specifies type of the content control.
- ContentControlLevel enum - Specifies layout level of the content control.
- ContentControlAppearance enum - Specifies appearance of the content control.
- CalendarType enum - Specifies the calendar type.
- DateStorageFormat enum - Specifies the date translation to be applied to the date content control.
- WebExtensionRelationship enum - Specifies relationship between a content control and an Office Web Extension.

New members:

- Cell CellCollection.Insert(string text, InsertLocation location) method - Inserts a Cell into this collection at the specified location.
- Row RowCollection.Insert(string[] texts, InsertLocation location) method - Inserts a Row into this collection at the specified location.

- `BodyType.BuildingBlock` property - Represents the body of a building block.
- `List<string> Settings.AttachedXmlSchemas` - Gets the list of custom XML schema whose target namespace is associated with this document when it is loaded, if such a schema is available to the hosting application. Applications can also load and utilize any additional schemas as well as those explicitly mentioned here. These custom XML schemas can then be used to validate the structure of the `CustomXmlPart.XmlDocument` in the document etc.
- `ContentControlCollection RangeBase.ContentControls` property - Gets the collection of content controls included in this range.
- `ContentControl Range.ParentContentControl` property - Gets the parent content control where the range content is stored.
- `Added ImageData.ContentType` property - Gets the image content type.
- `Added Marker.StartSection` property - Gets the Section which starts in this marker.
- `Added Marker.EndSection` property - Gets the Section which ends in this marker.
- `Added Paragraph.AddSectionBreak(SectionStart)` method - Breaks the parent section right after this paragraph.
- `Added the GcWordDocument.GlossaryDocument` property - Gets the glossary document.
- `Added the GcWordDocument.CustomXmlParts` property - Gets the collection of `CustomXmlPart` objects.

Improvements:

- Improved performance when calculating values of derived properties.
- Improved Word to PDF export.

## Changes From the Previous Release

This version of the product has the following change.

- `Range()` and `CompareLocationWith(Range range)` methods now accept `RangeBase` class in range argument instead of the `Range` class.

## Bug Fixes

The following issues have been resolved since the last release.

- Fixed the incorrect saving issue of background color shading. Now, MS Word doesn't show the message "not enough memory to update the display" while loading the saved document.
- Now, no wrong exception messages are shown when users apply a wrong style type to a content object.

## Breaking Changes

This version of the product has the following breaking changes.

- The type of `Document` property has been changed from `GcWordDocument` to `DocumentBase` on the following classes: `ContentObject`, `ContentRange`, `RangeBase`, `Style`, `StyleCollection`, `ListTemplateCollection`.
- The `ImageData(GcWordDocument document, bool isPictureBullet)` constructor has been changed to `ImageData(DocumentBase document, bool isPictureBullet)`. This means that it now accepts `DocumentBase` class in document argument instead of the `GcWordDocument` class.
- Earlier, theme colors could be modified directly via `GcWordDocument.Theme`. But now, theme colors cannot be modified directly via `GcWordDocument.Theme`. Instead, users need to use `Settings` to modify theme colors now.
- The `Theme.ColorScheme[ThemeColorId]` indexer has been removed. Instead, users can use the new `ThemeColor Settings.GetThemeColor(ThemeColorId)` method.
- The `ColorScheme.RemapColor(ThemeColorId, ThemeColorSchemeld)` method has been removed. Instead, users can use the new method - `Settings.RemapColor(ThemeColorId, ThemeColorSchemeld)`.
- Sections in `DsWord OM` are not objects but section type breaks (this provides consistency). Now, `Section` class does not inherit the `ContentObject` class.
- The `SectionCollection` class is inherited from `ContentCollection<Section>` instead of `ContentObjectCollection`

class.

- Removed Section.Guid and Section.ParentContent properties.
- Removed SectionCollection.Add(), SectionCollection.Insert() and Section.Split() methods. Instead of these methods, users can use the new Paragraph.AddSectionBreak() method.

## Version 2.1.0.260

### New Features and Improvements

- Added EastAsianLayout class which specifies any East Asian typography settings which shall be applied to the contents of a run.
- Added EastAsianTypography class which provides options for East Asian typography.
- Added HyphenationOptions class which allows to configure document hyphenation options.
- Added ColorScheme class which represents the color scheme of a Microsoft Office theme.
- Added ThemeColor class which represents a color in the color scheme of a theme.
- Added ThemeColorSchemeld enum which specifies the theme colors for document themes.
- Added ViewType enum which provides possible values for the view mode in application.
- Added ZoomType enum which provides possible values for how large or small the document appears on the screen in the application.
- Added ViewOptions class which provides various options that control how a document is shown in application.
- Filled CompatibilityOptions class which contains compatibility options (previously it contained only CompatibilityMode property).
- Added FontBase.EastAsianLayout property that gets East Asian typography settings which shall be applied to the contents of the run.
- Added FontBase.FitTextWidth property that gets or sets that the contents of the run which shall not be automatically displayed based on the width of its contents, rather its contents shall be resized to fit the width specified by the this property.
- Added FontBase.FitTextId property that gets or sets a unique ID which shall be used to link multiple contiguous runs with specified "FitTextWidth" property to each other to ensure that their contents are correctly merged into the specified width in the document.
- Added Settings.HyphenationOptions property which provides access to document hyphenation options.
- Added Settings.EastAsianTypography property which provides access to East Asian typography options.
- Added Settings.ViewOptions which provides various options that control how a document is shown in application.
- Added Theme.ColorScheme which gets a set of colors which are referred to as a color scheme.
- Improved FontBase.Name property behavior - If HintType is FontHintType.ComplexScript returns NameBi", if HintType is FontHintType.EastAsia returns NameFarEast.
- Improved performance of creating a new GcWordDocument instance more than 4 times.
- Allowed to open a DOCX while it is open in MS Word (use ReadWrite share mode instead of Read in GcWordDocument.Load()).

### Bug Fixes

- Fixed the issue where page color disappears after export.
- Fixed the issue with saving picture bullets for built-in list templates.

### Breaking Changes

- Renamed ThemeColor enum to ThemeColorId.
- Changed default value for WordColor.ThemeShade from 0 to 255.
- Changed default value for WordColor.ThemeTint from 0 to 255.