Table of Contents

Overview

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

GrapeCity Documents for Word library, referred to as **GcWord**, is a part of GrapeCity Documents that aims to be a complete solution to program and work with Word documents, without using any external word processor like MS Word. GcWord 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

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

- **Work with Word documents programmatically**
  Using GcWord, 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**
  GcWord is a high performance library which is faster alternative to the Microsoft Word automation.

- **Work with Document content**
  GcWord allows you to work with inbuilt document properties, text objects, range objects, and macros.

- **Document Formatting**
  GcWord supports document formatting which includes character, paragraph, list, table, page, and section formatting.

- **Extensive support for Content controls**
  GcWord 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**
  GcWord supports DOCX, DOTM, DOCM, DOTX file formats completely and can read files with FlatOPC, FlatOpcMacroEnabled, FlatOpcTemplate, and FlatOpcTemplateEnabled formats.

- **Export to PDF**
  GcWord library allows you to export a Word document to PDF 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 to PDF without impacting their formats.

- **View Options**
  GcWord library lets you set the view and zoom options to specify how a document appears when it is opened in an application.

- **Theme**
  GcWord library allows you to customize themes applied on a Word document, hence giving it a consistent styling and professional look.

- **Rich set of features**
  GcWord library provides a rich set of features that allow you to generate Word documents with content including formatted text and paragraphs, images, hyperlinks, bookmarks, comments, tables, lists, headers, footers, footnotes, endnotes and more.

For additional information about the supported features in GcWord, see Features topic.

> Note that the following MS Word objects are not yet supported by GcWord object model. These objects are preserved in a load, modify or save scenario but are not accessible via GcWord object model and are not parsed by GcWord except for removal.
>
> - Drawing objects including shapes, charts, diagrams (excluding real pictures)

- Math objects, such as formulas
- VML objects
- Embedded Objects, such as controls and OLE objects
- Custom XML
- Structured Document Tags. However, children of these tags can be parsed and if the child type is known then appropriate content object can be created.
- Form fields, such as check boxes, dropdowns, text boxes
- Revisions
- Permissions. As they are used to protect content from editing, so this is not a limitation for PDF export.
- Ruby Phonetic Guide
- Sub Document References
- Formatting, such as Font and Picture effects

Getting Started

System requirements

GcWord system requirements, depending upon the framework you are using to create an application, are:

- Our packages include two targets, .NET Standard 2.0 and .NET Framework 4.6.1. In order to use them, your application needs to target either of the following:
  - .NET Core 2.0 or later
  - .NET Framework 4.6.1 or later
- Visual Studio 2015+/Visual Studio for MAC/Visual Studio Code for Linux

For OS versions supported in .NET Core 2.0+, see .NET Core 2.0+ - Supported OS versions.

Setting up an application

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

Reference	Purpose
GrapeCity.Documents.Word	To use GcWord in an application, you need to reference (install) just the GrapeCity.Documents.Word package. It pulls in the required infrastructure packages.
GrapeCity.Documents.Layout	To enable saving Word documents to PDF, install the GrapeCity.Documents.Layout package (GcLayout for short). It provides extension methods allowing to save GcWordDocument as PDF.
GrapeCity.Documents.Imaging	For image handling, you need to reference (install) the GrapeCity.Documents.Imaging package.
GrapeCity.Documents.Common	GrapeCity.Documents.Common is an infrastructure package used by other packages. You do not need to reference it directly.
GrapeCity.Documents.Common.Windows	On a Windows system, you can optionally install GrapeCity.Documents.Common.Windows. It provides support for font linking specified in the Windows registry, and access to native Windows imaging APIs, improving performance and adding some features (e.g. TIFF support).
GrapeCity.Documents.DX.Windows	GrapeCity.Documents.DX.Windows is an infrastructure package used by GrapeCity.Documents.Common.Windows. You do not need to reference it directly.

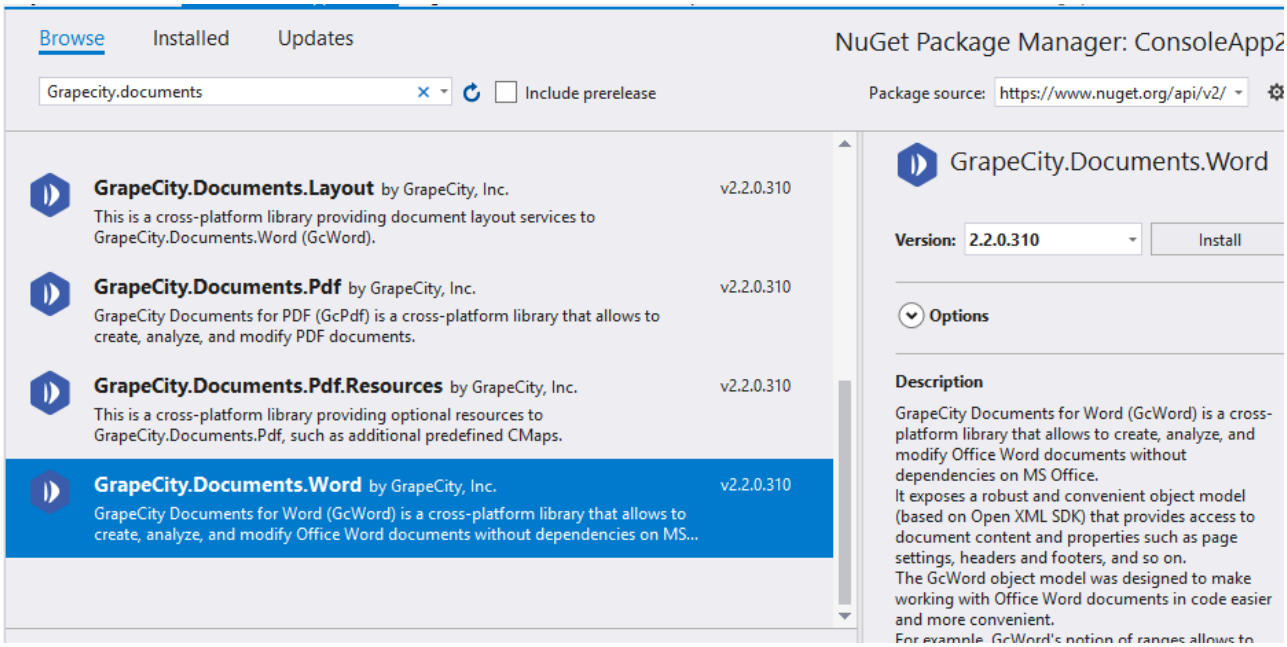
Add reference to GcWord in your application from nuget.org

In order to use GcWord 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 "Grapecity.Documents".

6. On the left panel, select **GrapeCity.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 "GrapeCity.Documents".
5. From the list of packages displayed in the left panel, select **GrapeCity.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 "**Grapecity**" and press Enter.
   Observe: GrapeCity packages get displayed in the dropdown.
10. Choose **GrapeCity.Documents.Word**.
11. Type following command in the Terminal window: "`dotnet restore`"

This adds references of the package to your application.

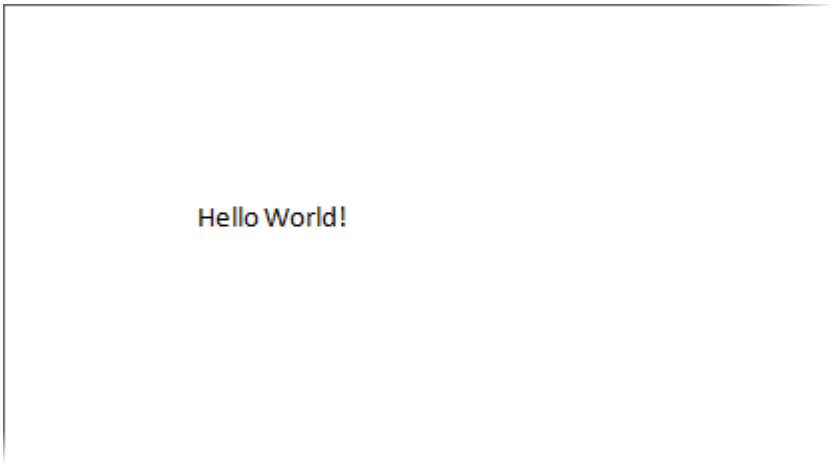
Quick Start

The following quick start sections help you in getting started with the GcWord 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 GcWord document**
2. **Add text to the document**
3. **Save the document**



Step 1: Create a new GcWord document

1. Create a new application (.NET Core Console App\Windows Forms App) and install the GrapeCity.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

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

Back to Top

Step 3: Save the document

Save the document using Save method of the **GcWordDocument** class.

C#

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

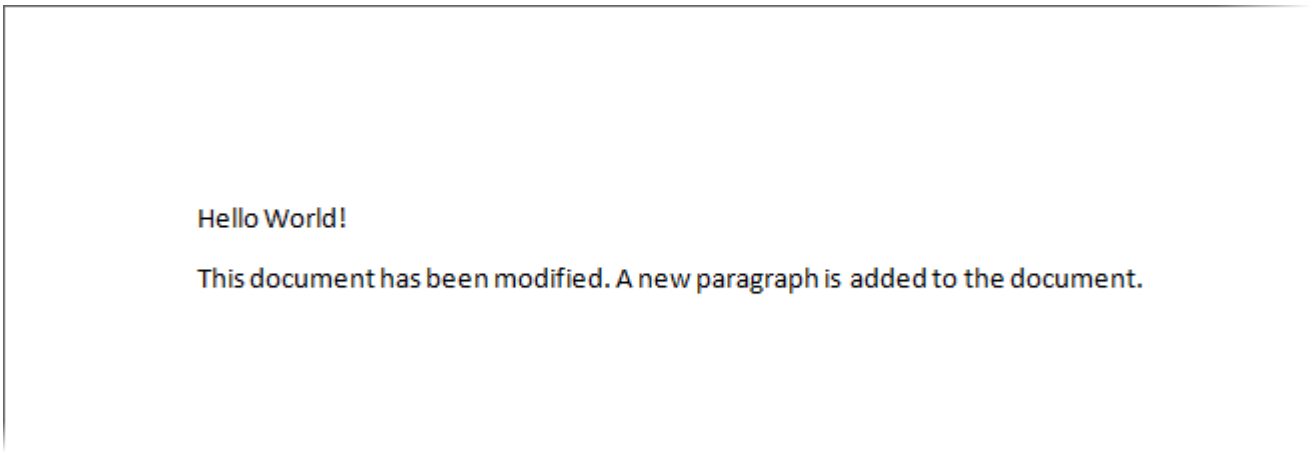
Note that the file is saved to the default location, which is the "bin/Debug" folder of the application.

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 GcWord**
2. **Modify the document**
3. **Save the document**



Step 1: Load an existing document in GcWord

1. Create a new application (.NET Core Console App\Windows Forms App) and install the GrapeCity.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 instance of the **GcWordDocument** 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 **GcWordDocument** 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 modifed Word file
doc.Save("SampleDoc_Modified.docx", DocumentType.Document);
```

Back to Top

License Information

Types of Licenses

GrapeCity Documents 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 GrapeCity Word. The document is limited to 200 paragraphs. Contact us.sales@grapecity.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 GrapeCity Word. Contact us.sales@grapecity.com to get your 30-day evaluation key.'

Evaluation License

GcWord 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@grapecity.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 GrapeCity Word. The document is limited to 200 paragraphs. Contact us.sales@grapecity.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 GrapeCity Word. Contact us.sales@grapecity.com to purchase license.'

Licensed

GcWord 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 GcWord without any limitations.

Apply License

To apply evaluation/production license in GcWord, 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 source:

- Product forum: https://www.grapecity.com/forums
- Email: us.sales@grapecity.com

Redistribution

In order to distribute the application, make sure you meet the installation criteria specified in the System Requirements page in this documentation. Further, you also need to have a valid Distribution License to successfully distribute the application.

> GcWord 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://www.grapecity.com/
E-mail	us.sales@grapecity.com
Phone	1.800.858.2739 or 412.681.4343
Fax	(412) 681-4384

End-User License Agreement

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

Product Architecture

Packaging

GcWord is a collection of .NET Standard 2.0 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.

GcWord works on all platforms supported by .NET Standard, including .NET Core, ASP.NET Core, .NET Framework and so on.

GcWord and supporting packages are available on nuget.org:

Package	Description
GrapeCity.Documents.Word	Main package which automatically pulls in the other required infrastructure packages.
GrapeCity.Documents.Layout	Enables saving Word documents as PDF.
GrapeCity.Documents.Imaging	Provides image handling.
GrapeCity.Documents.Common	An infrastructure package used by other packages.
GrapeCity.Documents.Common.Windows	Provides support for font linking specified in the Windows registry. On a non-Windows system this library can be referenced, but will do nothing.
GrapeCity.Documents.DX.Windows	Provides access to the native graphics APIs when running on a Windows system.

Document Overview

A Word document in GcWord 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	• Sections • tables • paragraphs
Inline elements	Elements that must be placed inside another elements	• 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.	• 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 GcWord, 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 GcWord 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.

ContentObect

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, GcWord 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. GcWord 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

GcWord 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

GcWord 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 GcWord. 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 GcWord.

Comments
: Add, modify, and delete comments and comment replies in GcWord.

Content Controls
: Work with content controls in GcWord.

Glossary Document
: Maintain the collection of building blocks in Glossary document in GcWord.

Document
: Work with document properties and styles in GcWord.

Export
: Export a Word document to PDF in GcWord.

Fields
: Add, modify, and delete fields in GcWord.

Footnotes and Endnotes
: Add, modify, and delete footnotes and endnotes from a Word document in GcWord.

Header and Footer
: Add, modify, and delete header and footer from a Word document in GcWord.

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

Links
: Add, modify, and delete hyperlinks and bookmarks from a Word document in GcWord.

Lists
: Word with lists and list templates in GcWord.

Paragraph
: Work with paragraph and its properties in GcWord.

Range Objects
: Work with range objects in GcWord.

Sections
: Work with sections and section breaks in GcWord.

Table
: Work with tables, cells and rows in GcWord.

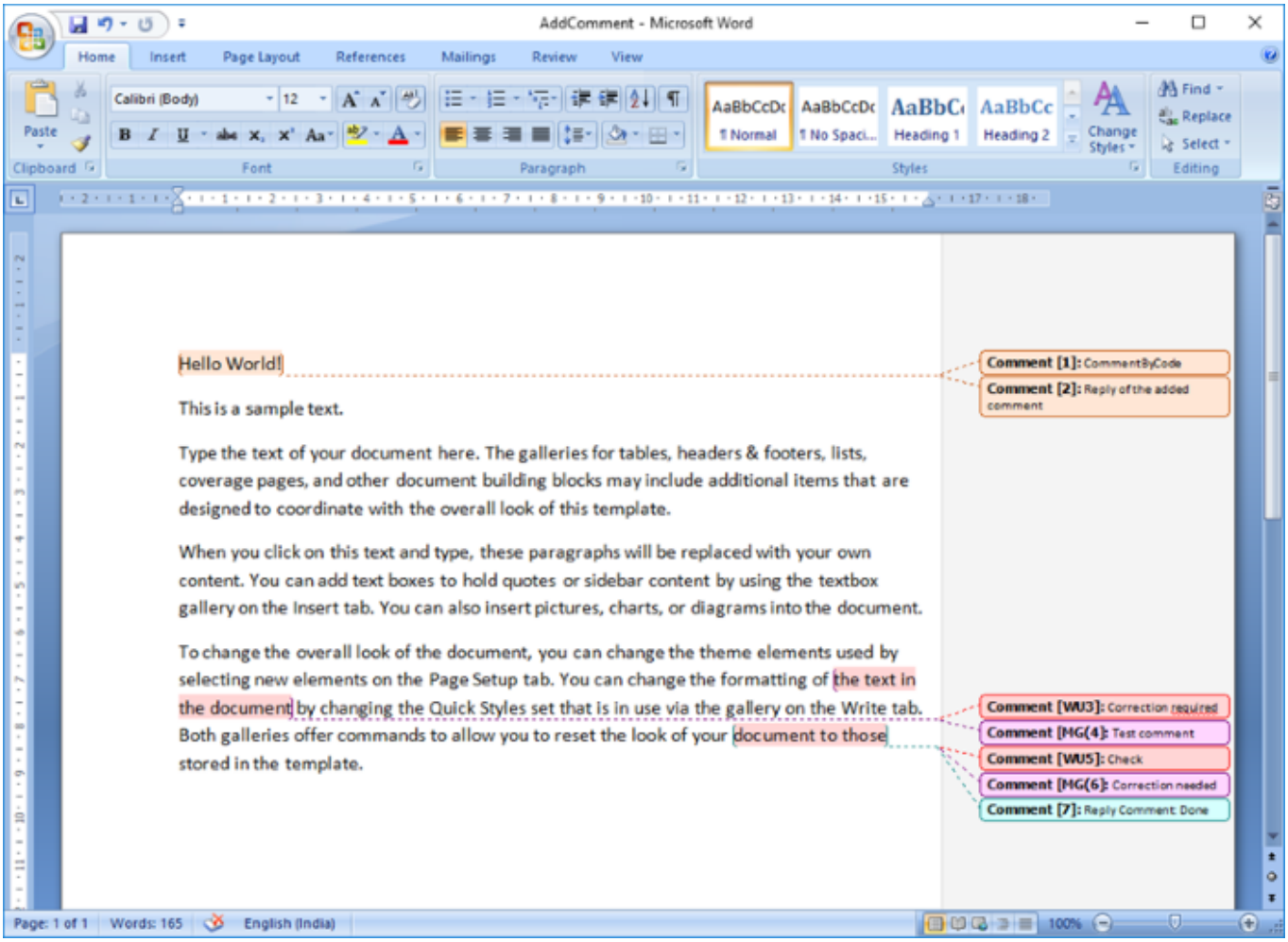
Text
: Work with text objects in GcWord.

Themes
: Change the theme color and font of a Word document in GcWord.

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 GcWord, 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. GcWord 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 GcWordUser 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 = "GcWordUser";
par.GetRange().Comments[0].Initials = "G.C.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", "Grapecity");

//Add a comment reply
Comment cReply = c.Reply("Reply of the added comment", "GC");

//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 GcWord, see GcWord sample browser.

Content Controls

GcWord 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.

GcWord 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.

GcWord 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.
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.
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

GcWord 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. GcWord 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

GcWord 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

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. GcWord 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. GcWord 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 bb1 = doc.GlossaryDocument.BuildingBlocks.Add("Mission_Statement",
"CompanyInfo", BuildingBlockGallery.AutoText, BuildingBlockInsertOptions.Content,
BuildingBlockType.None);
Style paraStyle = bb1.Body.Document.Styles.Add("paraStyle", StyleType.Paragraph);
paraStyle.Font.Bold = true;
bb1.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.
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 GcWord, 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
- 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.

GcWord provides following document properties through the GcWordDocument class:

Body
GcWord 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.

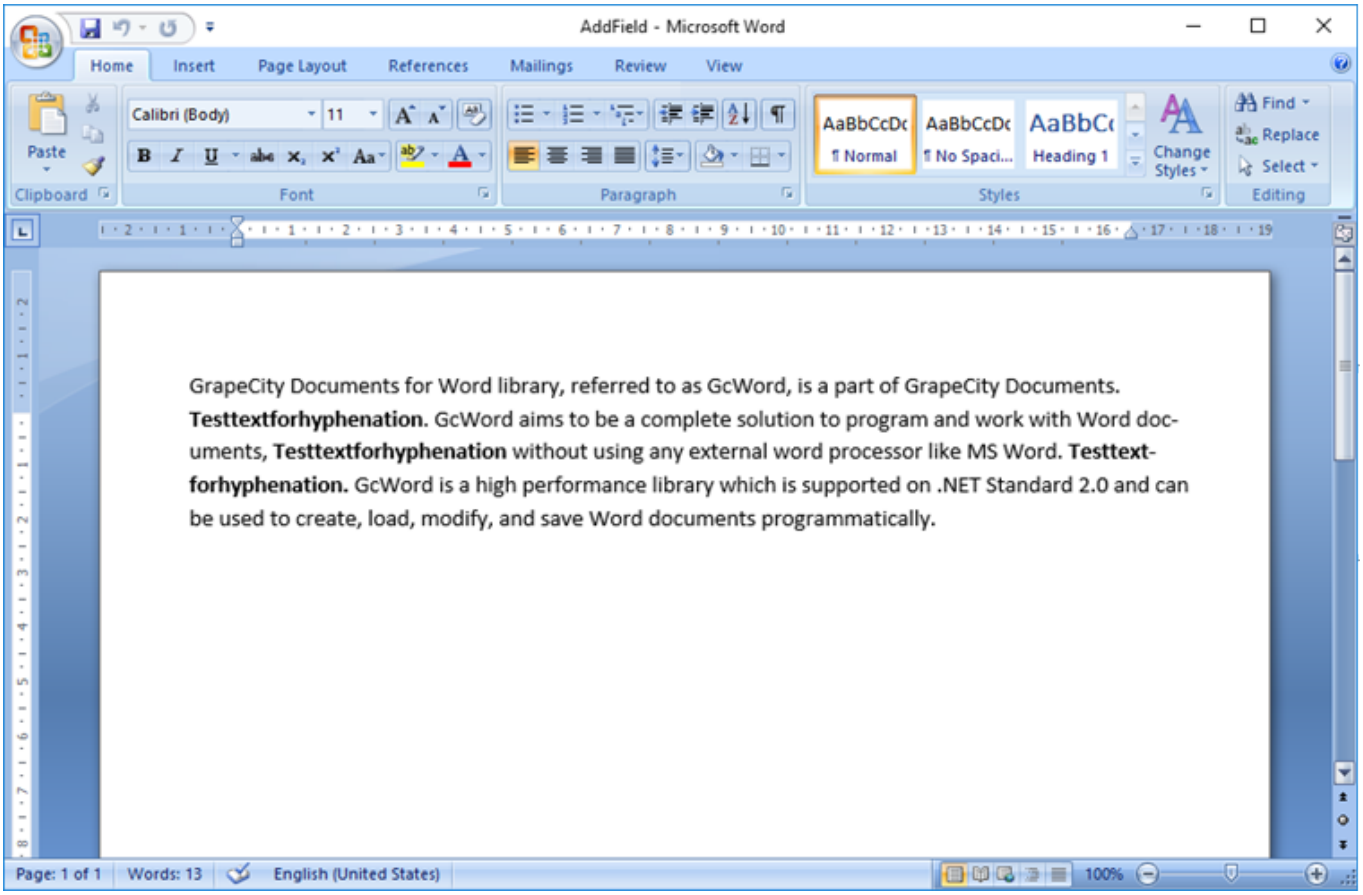
List Templates
GcWord provides the ListTemplates property to get a collection of list templates in the document.

Document Styles
GcWord allows you to access the collection of styles defined in the document using the Styles property.

Theme
GcWord provides the Theme property to get a theme that holds all the different formatting options available to a document through the theme.

Settings
GcWord 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.



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.CompatibiltyMode;

//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.CompatibiltyMode = WordVersion.Word2007;

//Enable automatic hyphenation for the document
doc.Settings.HyphenationOptions.AutoHyphenation = true;

//Save the modifed Word file
doc.Save("SetDocProperties.docx");
```

Back to Top

Set View Options

GcWord 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. GcWord 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 **ViewOption**s 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 GcWord, see GcWord sample browser.

Document 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.

GcWord 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, GcWord 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

For more information on how to apply different document styles using GcWord, see GcWord sample browser.

Page Settings

GcWord 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. GcWord 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, GcWord 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 GcWord, see GcWord sample browser.

Export

Exporting a Word document to PDF is a very common use case and is generally required for various benefits that PDF format offers. Some of them are security, cross-platform compatibility, availability of free readers, reduced file size etc.

GcWord allows you to convert and save a Word document as PDF using SaveAsPdf method of the GcWordDocumentExtensions class.

> In order to use **GcWordDocument.SaveAsPDF** method, you need to install Grapecity.Document.Layout package in your application.

C#

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

//save word file to pdf
doc.SaveAsPdf("TestFile.pdf");
```

Back to Top

For more information on how to convert a Word document into PDF using GcWord, see GcWord sample browser.

Limitations

- The objects that are not supported by GcWord object model cannot be exported.
- Footnotes are not supported when exporting a Word document to a PDF.
- Comments are supported when exporting a Word document to a PDF but with following limitations:
  - If there are lot of comments on a page, the layout might get affected
  - If there is a reply, connecting line goes to the reply only
  - Background color remains the same for all comments
  - Accepted comments are not highlighted
- Only stored values in fields are supported (i.e. no recalculation), with the following exceptions:
  - Page numbering is supported
  - Partial hyperlink field is supported

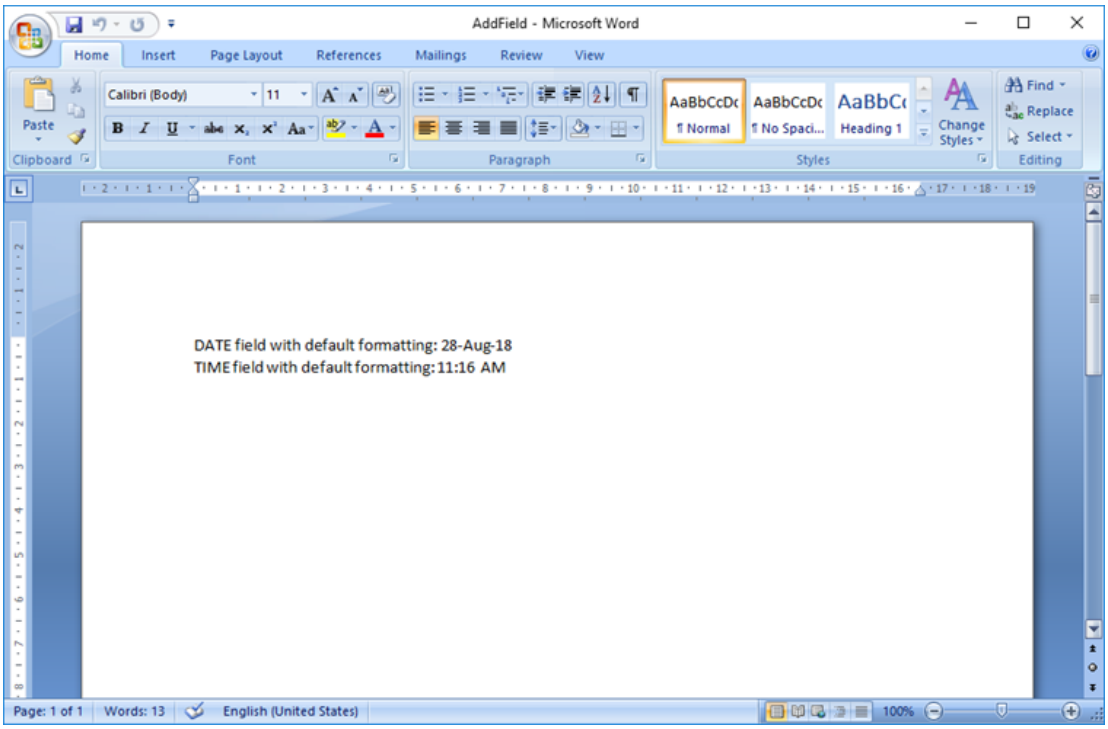
Fields

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 GcWord. 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.

GcWord 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.

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");
```

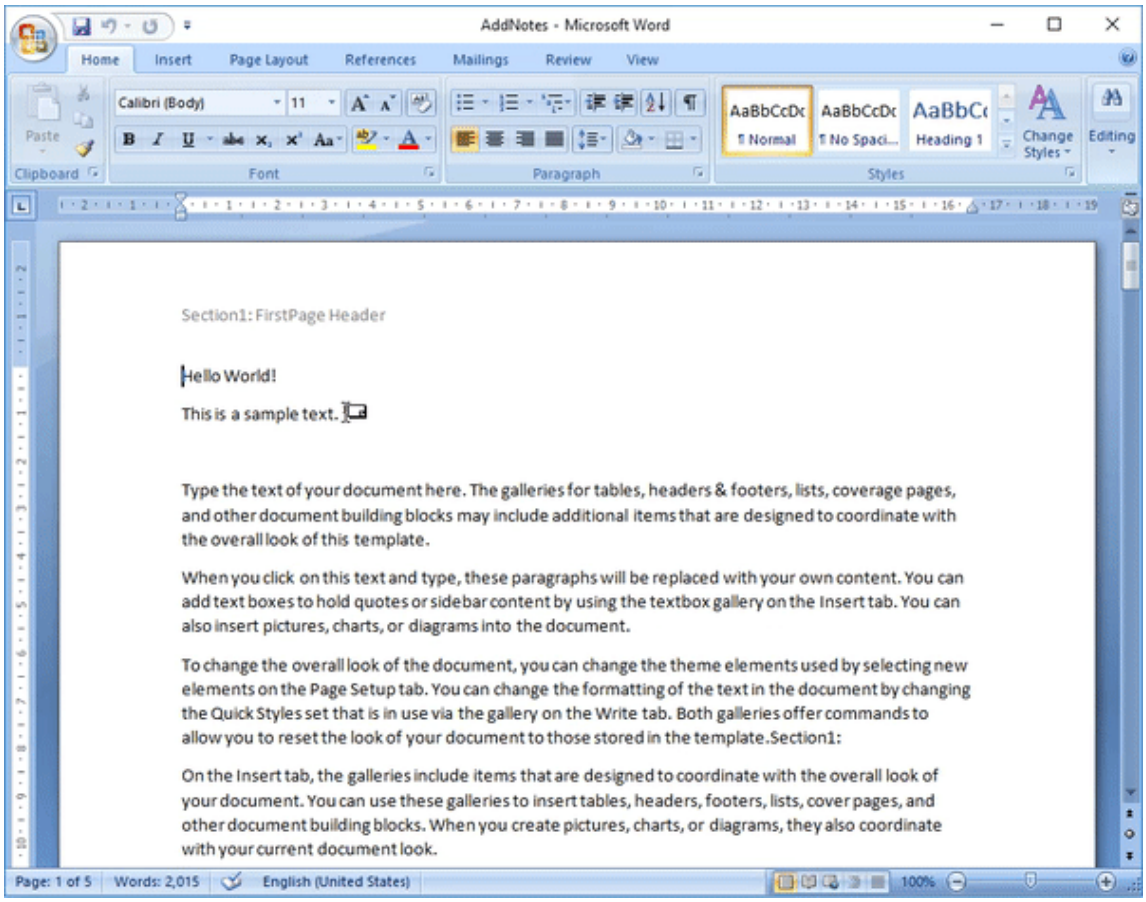
Back to Top

For more information on how to implement fields using GcWord, see GcWord 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.

GcWord 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 GcWord, see GcWord sample browser.

Header and Footer

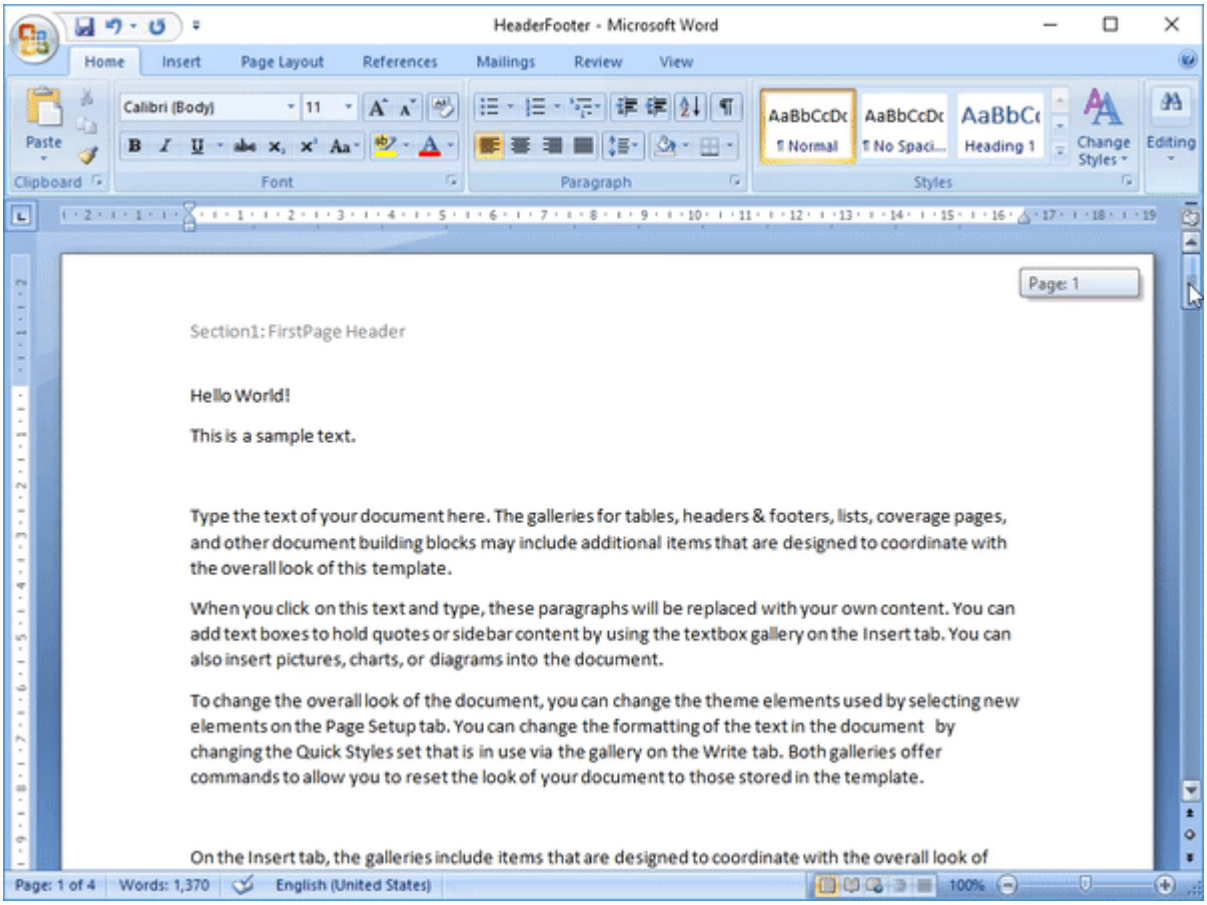
Headers and footers are generally used to display document name, date, page number, etc. In GcWord, headers and footers of a section are represented by the HeaderFooter class which provides access to the content and settings of header and footer. These headers and footers have their own body which represents 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, GcWord 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#

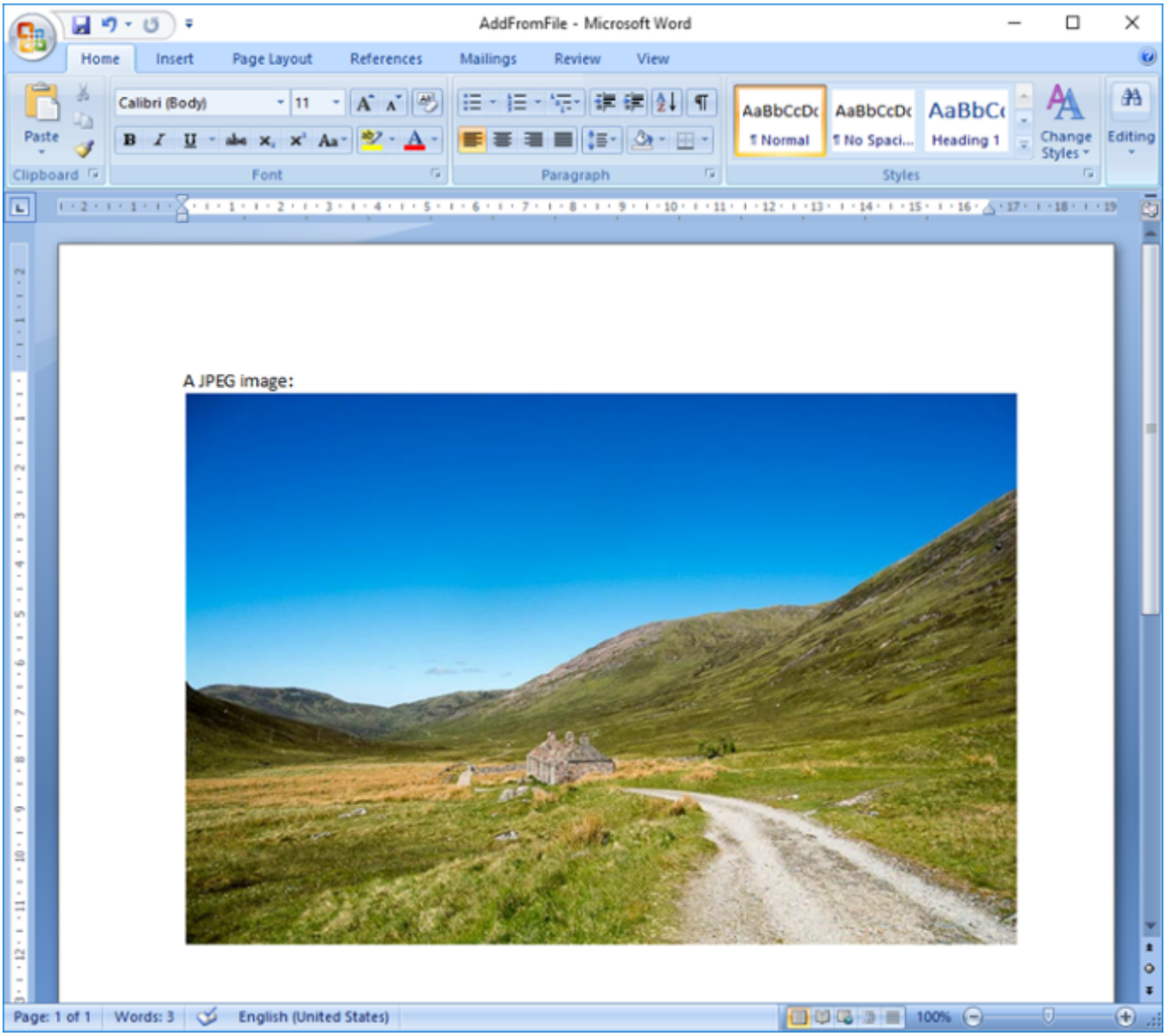
```
doc.Load("HeaderFooter.docx");

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 GcWord, see GcWord sample browser.

Images

Images are generally used to illustrate important information in your document and highlight points raised in the text. In GcWord, an image or a picture element is represented by the Picture class which allows you to access all the image properties. GcWord 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

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 GcWord, see GcWord sample browser.

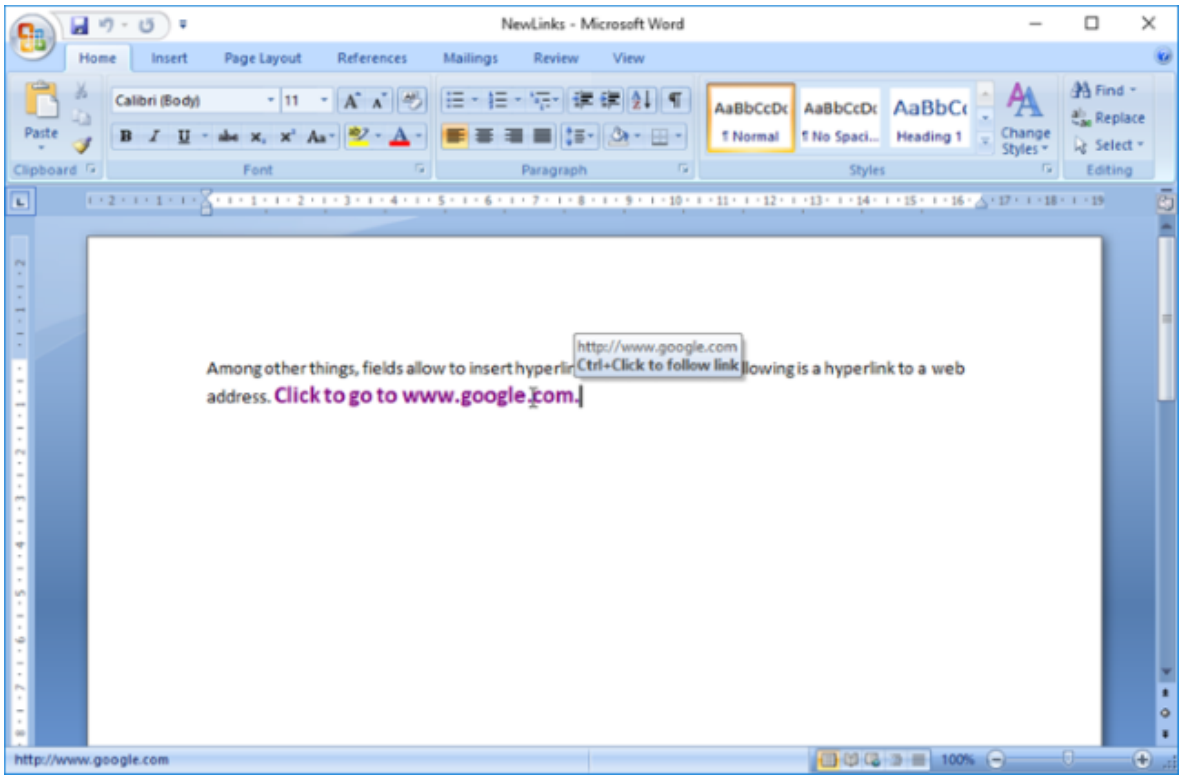
Links

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

GcWord allows you to add, modify, and delete hyperlinks in a document. In GcWord, 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 in GcWord instance
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://www.grapecity.com");
link1.GetRange().Texts[0].Value = "Click to visit Grapecity 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#

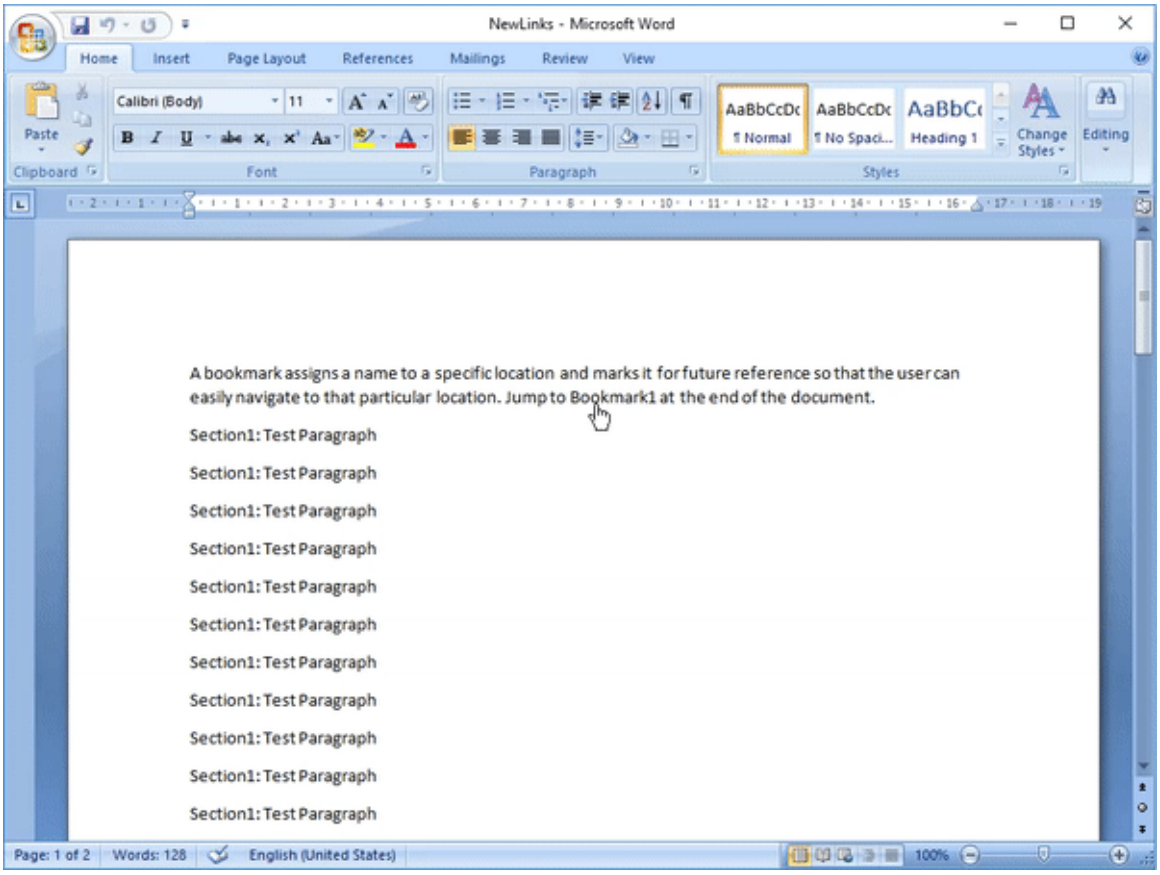
```
//Load the existing word document in GcWord instance
doc.Load("AddHyperlink.docx");

//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

For more information on how to implement hyperlinks using GcWord, see GcWord 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. GcWord provides you Bookmark class which represents a bookmark range in the content. Using GcWord, 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 in GcWord instance
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");
```

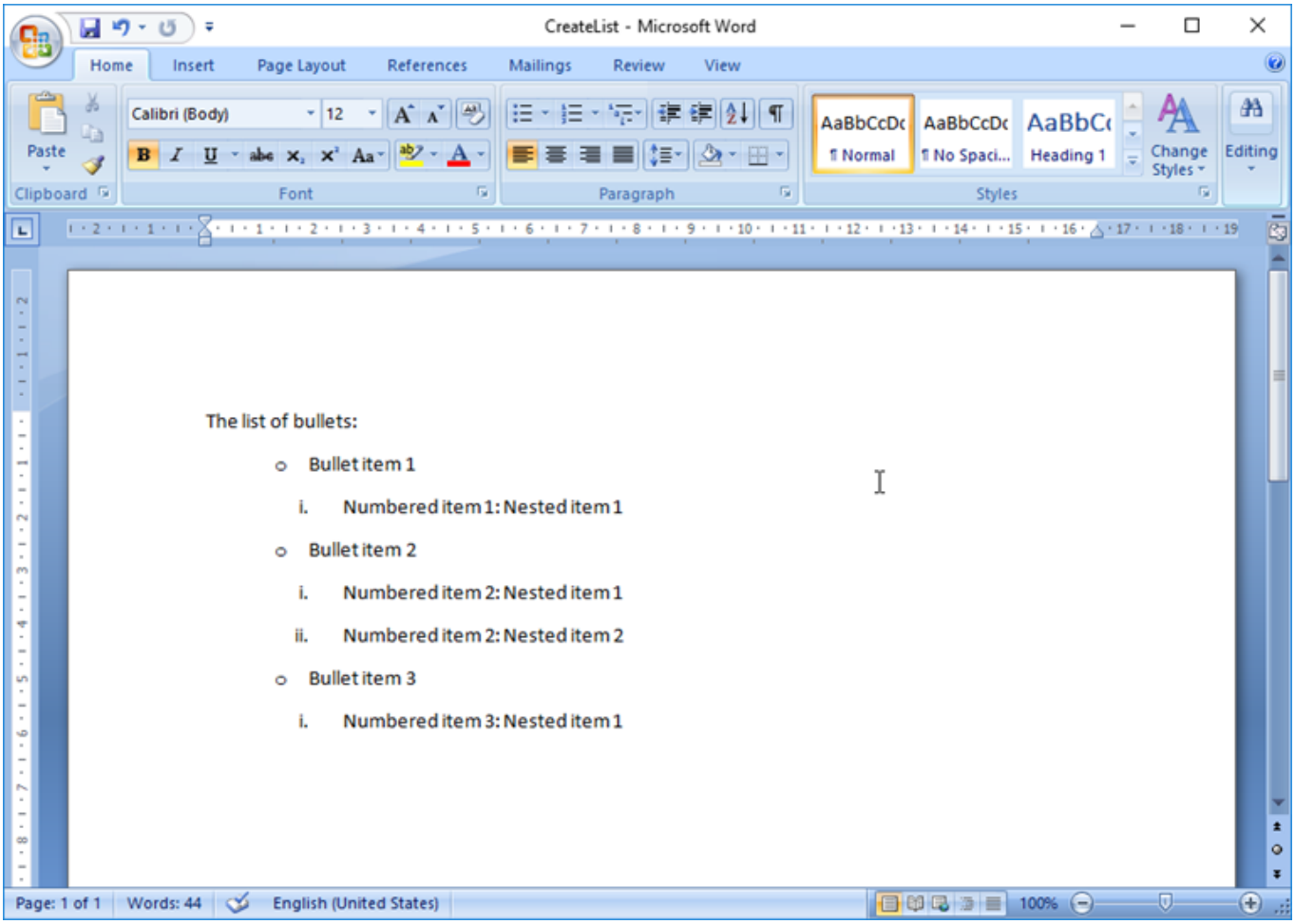
Back to Top

For more information on how to implement bookmarks using GcWord, see GcWord 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.

GcWord 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. GcWord 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 GcWordDocument 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 **GcWordDocument** 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("CreateList.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("CreateList.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

For more information on how to implement lists using GcWord, see GcWord sample browser.

Paragraph

In GcWord, a paragraph is represented by the Paragraph class. This class allows you to work with the paragraph element in a Word document. GcWord 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

GcWord 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.

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("Grapecity Document Solution:
GcWord");

//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 GcWord, see GcWord sample browser.

Indentation

GcWord 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 GcWord, see GcWord sample browser.

Line Spacing

GcWord 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 GcWord, see GcWord sample browser.

Borders

GcWord 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 GcWord, see GcWord sample browser.

Background Shading

GcWord allows you to specify the background shading for a paragraph using BackgroundPatternColor 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 **BackgroundPatternColor** 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.BackgroundPatternColor.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 BackgroundPatternColor 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.BackgroundPatternColor.RGB
=
    Color.LightPink;
```

Back to Top

For more information on how to implement background shading using GcWord, see GcWord 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.

GcWord 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 GcWord, see GcWord sample browser.

Paragraph Numbering

GcWord allows you to specify paragraph numbering using the ListTemplate class which can be accessed using ListTemplates property of the GcWordDocument 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 GcWord, see GcWord sample browser.

Paragraph Styling

GcWord allows you to define a style using Style class which can be accessed using Styles property of the GcWordDocument 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 GcWord, see GcWord 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. GcWord 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 in GcWord instance
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 in GcWord instance
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 in GcWord instance
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 in GcWord instance
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[0].Delete();
}

//Save the document with the changes
doc.Save("RemoveRange.docx");
```

Back to Top

For more information on how to work with range objects using GcWord, see GcWord sample browser.

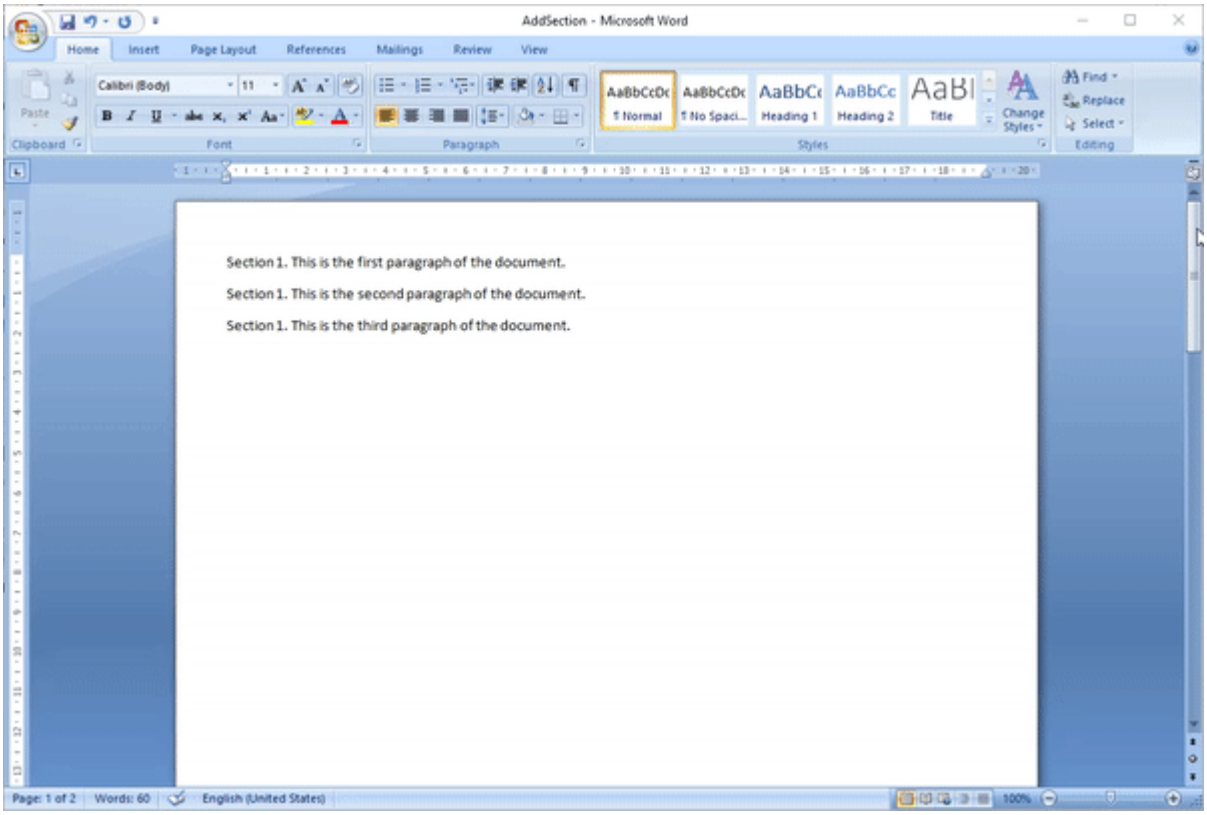
Sections

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 GcWord, 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 GcWordDocument.Body.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();
//Acess 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 GcWord, see GcWord sample browser.

Table

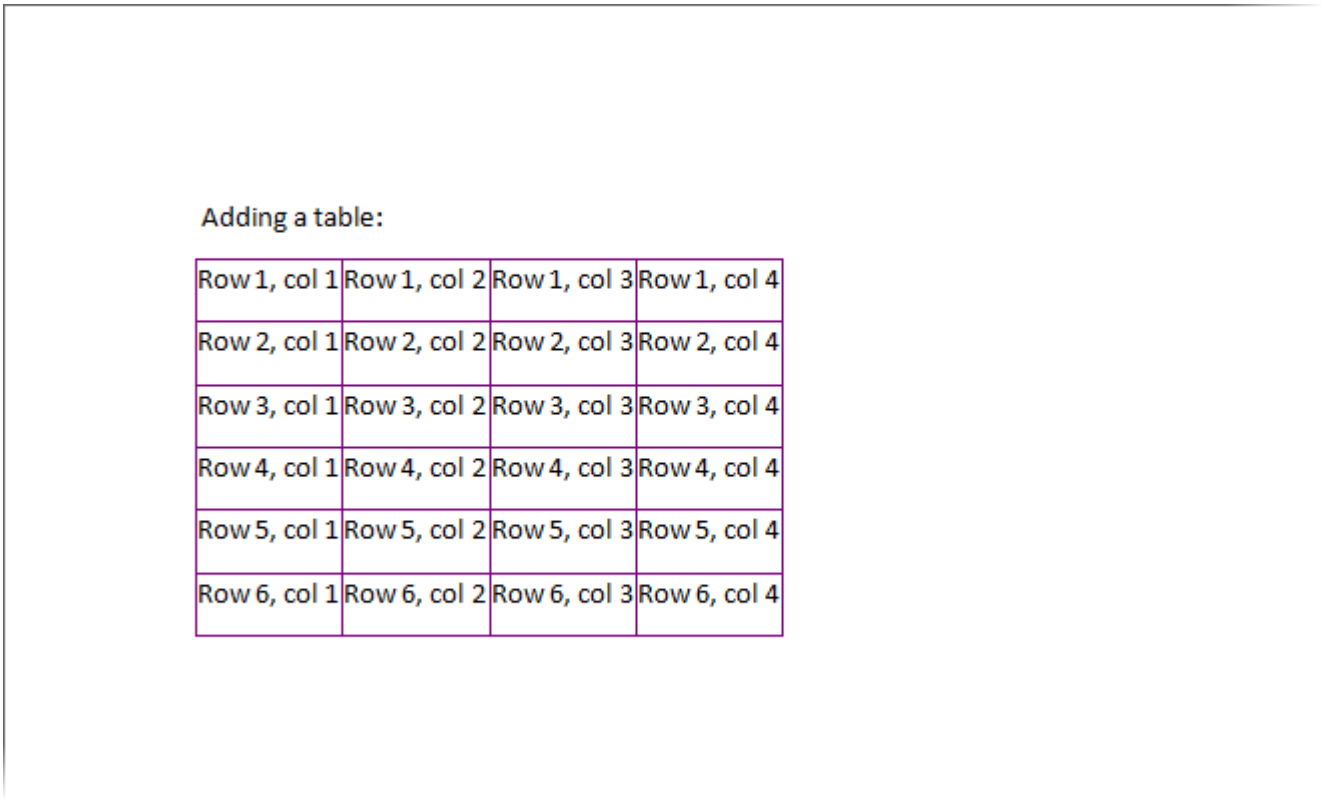
Tables in a document help in displaying the content in organized and structured manner. In GcWord, a table is a block content object and is represented by the Table class which lets you access the structure, content and formatting of the table.

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

- Table
- Cell
- Row

Table

GcWord 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.



Create Table

To create a table in Word document using GcWord:

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 spit a table in a Word document:

1. Access a table and then access its row. For example, access the second row oof 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 GcWord, see GcWord sample browser.

Cell

In GcWord, 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. GcWord 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 PictureColection. 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 GcWord, see GcWord sample browser.

Row

GcWord provides the Row class representing a table row element. A row can be added to a table through Add method of the RowCollection class. GcWord 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, GcWord 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 GcWord, see GcWord sample browser.

Text

GcWord 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, GcWord 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, GcWord 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 GcWord. ",
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.

2. Split the run text using Split method of the **Text** class and split the run using Split method of the **Run** class respectively.
3. 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:

1. Access the section where you want to delete text using the GetRange method. For example, access the first run of a section.
2. Access the text from the text collection using Texts property of the RangeBase class.
3. 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#

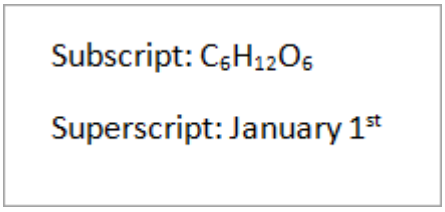
```
doc.Load("TestDoc.docx");

//Fetch text at document level
Console.WriteLine(doc.Body.Text);
```

Back to Top

For more information on how to work with text objects using GcWord, see GcWord sample browser.

Set Subscript and Superscript Text

GcWord 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.



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 dateFrags = Regex.Split(date, "(st)");

//Add text to the paragraph2
foreach (var frag in dateFrags)
{
    //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 GcWord, see GcWord sample browser.

Fit Text Width

GcWord 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.

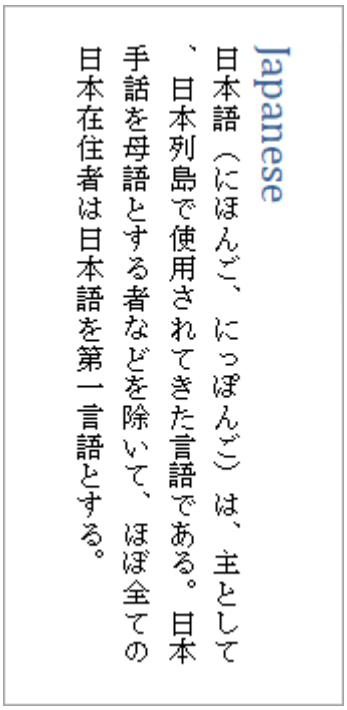
C#

```
//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

GcWord 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, GcWord 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");
```

For more information on how to work with text objects using GcWord, see GcWord 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.

GcWord 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("http://help.grapecity.com/gcdocs/gcword/onlinehelp/webframe.html"),
         "", "GcWord documentation.");

//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 GcWord, see GcWord sample browser.

Samples

All the GcWord 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.

List of samples

The complete list of available GcWord sample projects is mentioned below:

Basic Feature Samples

Features	Sample	Description
	Hello, World!	Demonstrates how to create a Word document that includes some text.
Content	Character Formatting	Demonstrates how to apply formatting to ranges of Word document text.
	Character Styles	Demonstrates how to split text, create and apply character styles.
	Vertical Text	Demonstrates how to use a top-to-bottom, right-to-left page layout.
	Multiple Languages	Demonstrates how to render text in multiple languages.
	Tate Chu Yoko	Demonstrates how to render horizontal runs in a vertical text (tate chu yoko).
	Create Many Documents	Demonstrates how to set the timing of GcWordDocument constructor.
	Sections	Demonstrates how to add sections to a document and format their page setup.
	Sub and Super Script	Rendering text as subscript and superscript.
	Columns	Demonstrates how to create a document with multi-column layout.
	Bullet List	Demonstrates how to create a simple bullet list.
	Numbered List	Demonstrates how to create a numbered list with three levels.
	Headers and Footers	Demonstrates how to add headers and footers to a document.
	Footnotes	Demonstrates how to add and format footnotes.
	Hyperlinks	Demonstrates how to insert hyperlinks into a Word document.
	Add Picture	Demonstrates how to add a JPEG image to a document.
	Picture Wrapping	Demonstrates how to insert pictures using different picture wrapping styles.
	Comments	Demonstrates how to add comments (including replies) to text selections or

		whole paragraphs.
	Comments (not in PDF)	Demonstrates how to add comments to document, but ignore them when exporting to PDF.
Styles	List of Built-in Styles	Demonstrates how to list all available built-in styles.
	Some Common Styles	Demonstrates some common built-in styles.
	Built-in Character Styles	Demonstrates built-in character styles.
	Built-in Paragraph Styles	Demonstrates built-in paragraph styles.
	Built-in Table Styles	Demonstrates all built-in table styles.
	Built-in List Templates	Demonstrates all built-in list templates.
Modification	Load DOCX files	Demonstrates how to load DOCX files and add notes to the content.
	Modify Formatting	Demonstrates how to modify formatting in an existing DOCX.
	Theme Colors	Demonstrates how to change theme colors used in a document.
	Replace Text	Demonstrates how to find and replace text in an existing document.
	Replace and Format Text	Demonstrates how to find a string in a document and replace its content and formatting.
	Replace and Format Text (2)	Demonstrates how to find a string in a document and replace its content using GetPersistentRange method.
	Replace Image	Demonstrates how to replace an image in an existing document.
	Copy Paragraphs	Demonstrates how to copy and join paragraphs in a document.
Tables	Simple Table	Demonstrates how to create tables, add rows and cells, remove rows, specify table styles.
	Grid Span	Demonstrates how to create a simple table with grid span.
	Alternating Rows	Demonstrates how to use conditional formatting to build a table with alternating row backgrounds.
	Conditional Styles	Demonstrates how to apply conditional formatting to table elements.
	Nested table	Demonstrates how to create a table with a nested table inside a cell.
Templates	MailMerge Bookmarks	Demonstrates how to use bookmarks for mail merge.
Fields	Hyperlinks	Demonstrates how to insert hyperlink fields into a Word document.
	Date/Time Fields	Demonstrates how to display date and time fields in a variety of formats.
	Complex Fields	Demonstrates how to generate a document that modifies its text depending on the current date using complex fields.

Large Documents	Long Document	Demonstrates how to load long DOCX documents in code.
Miscellaneous	Content Controls	Demonstrates how to add content controls to a document.
	Custom XML Parts	Demonstrates how to add custom XML parts to a document.
	Glossary Documents	Demonstrates how to add building blocks to a document.

Fancy Docs Samples

Sample	Description
Procurement Letter	Demonstrates how to build a nice-looking procurement business letter.
Simple Document	Demonstrates how to use basic elements, such as text, borders, and picture to create a Word document.