

Table of Contents

Table of Contents	1-5
GcExcel .NET Overview	6
Key Features	7-8
Get Started	9-13
License Information	13-16
Upgrade to Latest Version	16
Technical Support	16
Redistribution	16
End User License Agreement	16-17
Features	18
Comments	18-19
Chart	19
Conditional Format	19
Data Validation	19-20
Formula	20
Group	20
Hyperlinks	20
Page Setup	20
PDF Export	20
Rich Text	21
Shape and Picture	21
Style	21
Sparkline	21
Theme	21-22
Table	22
Customize User Interaction	23
Manage Workbook	23
Create Workbook	23
Open, Save and Protect Workbook	23-25
Cut or Copy Across Sheets	25
Enable or Disable Calculation Engine	25-26
Apply Theme	26-27

Work with Workbook Views	27
Manage Worksheet	27-28
Work with Worksheets	28-30
Range Operations	30-31
Access a Range	31-32
Access Areas in a Range	32
Access Cells, Rows and Columns in a Range	32-33
Cut or Copy Cell Ranges	33-35
Cut or Copy Shape, Slicer, Chart and Picture	35-37
Find and Replace Data	37-38
Get Row and Column Count	38-39
Hide Rows and Columns	39
Insert And Delete Cell Ranges	39-40
Insert and Delete Rows and Columns	40-41
Merge Cells	41-42
Set Values to a Range	42
Set Row Height and Column Width	42
Auto Fit Row Height and Column Width	42-43
Work with Used Range	44
Freeze Panes in a Worksheet	44-45
Work with Shape And Picture	45-49
Customize Worksheets	49-50
Work with Worksheet Views	50-52
Allow Sort	52-55
Apply Filters	55-57
Manage Hyperlinks	57-58
Apply Grouping	58-59
Create Row or Column Group	59-60
Remove a Group	60-61
Set Summary Row	61
Apply Style	61-62
Set Sheet Styling	62-65
Create and Set Custom Named Style	65-67

Apply Comments	67-69
Set Rich Text in a Cell	69-72
Manage Formulas	73
Set Formula to Range	73-74
Set Table Formula	74-75
Set Array Formula	75-76
Manage Custom Functions	77
Work With Custom Functions	77-82
Manage Data	83
Use Chart	83
Create and Delete Chart	83
Configure Chart	83-84
Chart Title	84
Chart Area	84-86
Plot Area	86
Customize Chart Objects	86-87
Series	87-90
Configure Chart Series	90-97
Walls	97-98
Axis and Other Lines	98-100
Configure Chart Axis	100-102
Floor	102-103
Data Label	103-104
Legends	104-105
Use Sparkline	105-108
Use Table	108
Create and Delete Tables	108-109
Modify Tables	109-110
Apply Table Sort	110-111
Set Table Filters	111
Add and Delete Table Columns and Rows	111-112
Apply Table Style	112
Modify Table with Custom Style	112-113

Modify Table Layout	113-114
Use Pivot Table	114
Create Pivot Table	114-115
Pivot Table Settings	115-117
Apply Style To Pivot Table	117-121
Use Slicer	121
Add Slicer in Table	121-122
Add Slicer in Pivot Table	122-124
Use Do Filter Operation	124
Apply Slicer Style	124-125
Modify Slicer with Custom Style	125
Modify Table Layout for Slicer Style	125-126
Manage File Operations	127
Import and Export .xlsx Document	127-128
Export to a PDF File	128-129
Configure Fonts and Set Style	129-130
Export Pivot Table Styles And Format	130-132
Export Shapes	132-133
Export Vertical Text	133-134
Shrink To Fit With Text Wrap	134-135
Control Pagination	135
Render Excel Range Inside PDF	135-138
Export Multiple Sheets To One Page	138-139
Keep Rows Together Over Page Breaks	139
Delete Blank Pages From Middle	139-140
Export Different Headers On Different Pages	140-141
Export Last Page Without Headers	141-142
Export Custom Page Information	142-143
Export Specific Pages To PDF	143-144
Save Multiple Workbooks to Single PDF	144-145
Working With Page Setup	145-146
Import and Export CSV File	146-148
Import and Export CSV File with Delimiters	148-149

Import and Export JSON Stream	149-151
Import and Export Macros	151-152
Import and Export OLE Objects	152
Apply Conditional Formatting	153
Add Cell Value Rule	153
Add Date Occurring Rule	153-154
Add Average Rule	154
Add Color Scale Rule	154
Add Data Bar Rule	154-155
Add Top Bottom Rule	155-156
Add Unique Rule	156
Add Icon Sets Rule	156
Add Expression Rule	157
Apply Data Validations	158
Add Validations	158-160
Delete Validation	160
Modify Validation	160-161
Configure Print Settings via Page Setup	162
Configure Page Header and Footer	162-163
Configure Page Settings	163-164
Configure Page Breaks	164-165
Configure Paper Settings	165-166
Configure Print Area	166
Configure Columns to Repeat at Left and Right	166-167
Configure Rows to Repeat at Top and Bottom	167-168
Configure Sheet Print Settings	168
API Reference	169
Index	170-174

GcExcel .NET Overview

GrapeCity Documents for Excel, .NET Edition is a new small-footprint, high-performance spreadsheet component that can be used in your server or desktop applications. It gives developers a comprehensive API to quickly create, manipulate, convert, and share Microsoft Excel-compatible spreadsheets. Further, you can call it from nearly any application and platform.

GcExcel .NET targets multiple platforms including .NET Framework, .NET Core and Mono; thus making it the perfect solution for all your spreadsheet challenges.

The best part about using GcExcel .NET is that it models its interface-based API on Excel's document object model. This means that users can import, calculate, query, generate, and export any spreadsheet scenario as and when required. Moreover, the imported or generated spreadsheets can contain references to one another, such as you can reference full reports, sort and filter tables, sort and filter pivot tables, add charts, sparklines, conditional formats, and dashboard reports etc.

What GcExcel .NET offers you

- Facilitates server-side spreadsheet generation, manipulation, and serialization.
- Requires low memory footprint.
- Robust calculation engine.
- Produces output in varied formats including .xlsx and ssjson.
- Provides multi-platform support including .NET Framework, .NET Core and Mono.
- Compatible to run in environments including Winforms, WPF, ASP.NET etc.

For an introduction to GcExcel .NET features, the following documentation is available:

- [Features](#)

For product details, the following reference documentation is available:

- [API Reference](#)

Key Features

With a set of class libraries, collections, interfaces, pre-defined functions, properties and methods that comes packaged with GcExcel .NET; developers can quickly build everything right from the scratch to organize and structure business-critical data for maximum productivity and enhanced analysis.

GcExcel.NET provides users with the following essential features in order to facilitate developers in creating powerful spreadsheets using .Net Core:

- **Lightweight API Architecture for Improved Efficiency**
GcExcel .NET enables users to save a considerable amount of time, storage memory and efforts by improving the overall efficiency with its lightweight API architecture that can be used to generate, load, edit, save and convert spreadsheets.
- **Flexible Themes and Components**
For complete customization, GcExcel .NET allows you to set up custom themes, configure components, summarise data, customize styles, embed drawing objects, apply cell formatting and integrate calculation engine.
- **Seamless Excel Compatibility**
While executing the import operation, you can include pivot tables, comments, charts, conditional formatting, data validation, filters, formulas, shapes, pictures, slicers, sparklines and tables etc. in the spreadsheets without any compatibility issues.
- **Extensive Support for Major Operating Systems**
GcExcel .NET core applications can be deployed on all major operating systems including Microsoft Windows, Linux and macOS.
- **Based on Excel Object Model**
The interface-based API model enables users to import data, calculate formulas, query, generate, and export complex spreadsheet scenarios as per specific preferences.
- **No Dependency on MS Excel**
In order to work with GcExcel .NET, users don't need to install MS Office Suite and access MS Excel on their systems.
- **Use Built-in Templates for Simple Forms**
Using built-in templates, you can quickly create simple forms like invoice etc. while working with spreadsheets.
- **Create Interactive Experience with SpreadJS Sheets**
GcExcel .NET can be used with Spread.Sheets for a completely interactive and user-friendly spreadsheet experience.
- **Workbook and Worksheets**
You can create workbook and add worksheets while also performing the import and export operations. Further, you can activate worksheets, configure its display, delete it and protect it by encrypting it with a password.
- **Formulas and Functions**
With support for implementing formulas, creating custom functions and using 450+ built-in functions, you can execute complex spreadsheet calculations without any hassle.
- **Pivot and Excel Tables**
Users can create tables and pivot tables in order to automatically calculate the count, total or average of the data entered in the spreadsheets.
- **Export to PDF**
Using the export to pdf feature, users can save spreadsheets to PDF files with different page settings and features.
- **Deploy Apps with Excel Spreadsheets to the Cloud**
With GcExcel .NET, you can apply cloud based deployments and deploy your applications on Azure and AWS Lambda.

For more information on the complete list of supported features in GcExcel .NET, refer to the [Features](#) topic in the documentation.

Get Started

This topic includes:

- **System Requirements**
- **Setting up an application**

System Requirements

GcExcel .NET requires the following system requirements depending upon the framework you are using to create an application.

.NET Core

- Operating System Support
 - Ubuntu 14.04 & 16.04
 - Mac OS X 10.11+
 - Windows 7+ / Server 2012 R2+
 - Windows Nano Server TP5
 - Windows Server 2016
- .NET Core 2.0+ NuGet Package

.NET Framework 4.6.1

- Operating System Support
 - Windows 7+ / Server 2012 R2+
 - Windows Server 2016

Mono

- Operating System Support
 - Linux
 - Mac OS X
 - Sun Solaris
 - Windows 7+

Setting up an application

GcExcel .NET reference is available through NuGet, a Visual Studio extension that automatically adds libraries and references to your project.

Installing the product involves the following tasks. You can either get the [GcExcel .NET NuGet package](#) or [download it locally](#) on your machine.

If you are installing the GcExcel .NET package through NuGet, refer to the following tasks:

- **To find and install the GrapeCity.Documents.Excel NuGet package**

If you are installing the GrapeCity.Documents.Excel package manually, refer to the following tasks:

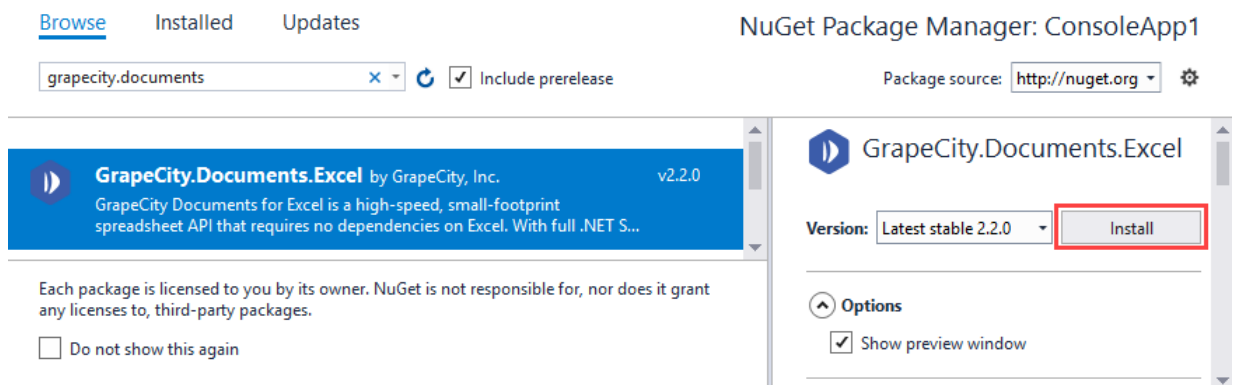
- **To manually create NuGet package source**
- **To install the GrapeCity.Documents.Excel package using command line interface**
- **To add GrapeCity.Documents.Excel package reference**

.NET Core Console Application

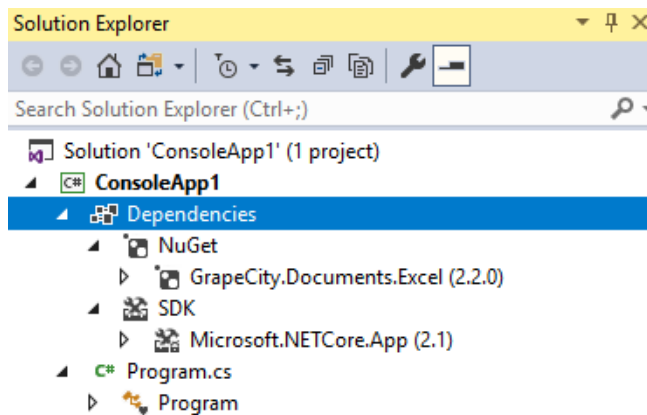
To find and install the GrapeCity.Documents.Excel NuGet package

Complete the following steps to add **GrapeCity.Documents.Excel** NuGet package in your .NET Core Console Application in Visual Studio.

1. In **Solution Explorer**, right-click either **Dependencies** or a project and select **Manage NuGet Packages**.
2. In the **Browse** tab, select **nuget.org** from the **Package source** dropdown.
3. In the **Browse** tab, type "grapecity.documents" or "GrapeCity.Documents" in the search text box at the top and find the package "GrapeCity.Documents.Excel" as shown in the below image.




- Click **Install** to install the **GrapeCity.Documents.Excel** package and its dependencies into the project. When the installation is complete, make sure you check the NuGet folder in your solution explorer and confirm whether or not the GrapeCity.Documents.Excel package is added to your project dependencies.



To manually create NuGet package source

In order to manually create Nuget feed source, you need to complete the following steps to add the Nuget feed URL to your Nuget settings in Visual Studio. Before you proceed with this step, make sure you first [download the GcExcel .NET Nupkg file](#) and put it in a local folder, for example - "D:\Nupkg".

- From the Tools menu, select **NuGet Package Manager | Package Manager Settings**. The **Options** dialog box appears.
- In the left pane, select **Package Sources**.
- Click the  button in the top right corner. A new source is added under **Available Package Sources**.
- Set a **Name** for the new package source.
- To add source in the **Source** field, click the ellipsis button next to the Source field to browse for the **Nupkg** folder.
- After you select the **Nupkg** folder, click the **Update** button and finally click OK.

To install the GrapeCity.Documents.Excel package using command line interface

- Open the CommandPrompt window on your Windows system.
- Use the **cd** command to navigate to your project folder. For example, `cd C:\Users\Admin\Documents\Visual Studio 2017\Projects\GcExcel.NetCore-ConsoleApplication`.
- Install GcExcel .NET NuGet package using the following command:
dotnet add package GrapeCity.Documents.Excel

To add GrapeCity.Documents.Excel package reference

GcExcel .NET is a cross-platform spreadsheet component that can be used on multiple platforms including Windows, Linux and Mac operating system.

In case you are creating an application using the Visual Studio, user can edit the ****.*.csproj** file and a package reference as shown in the image below:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Folder Include="wwwroot\" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="GrapeCity.Documents.Excel" Version="2.2.0" />
    <PackageReference Include="Microsoft.AspNetCore.App" />
    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" PrivateAssets="All" />
  </ItemGroup>

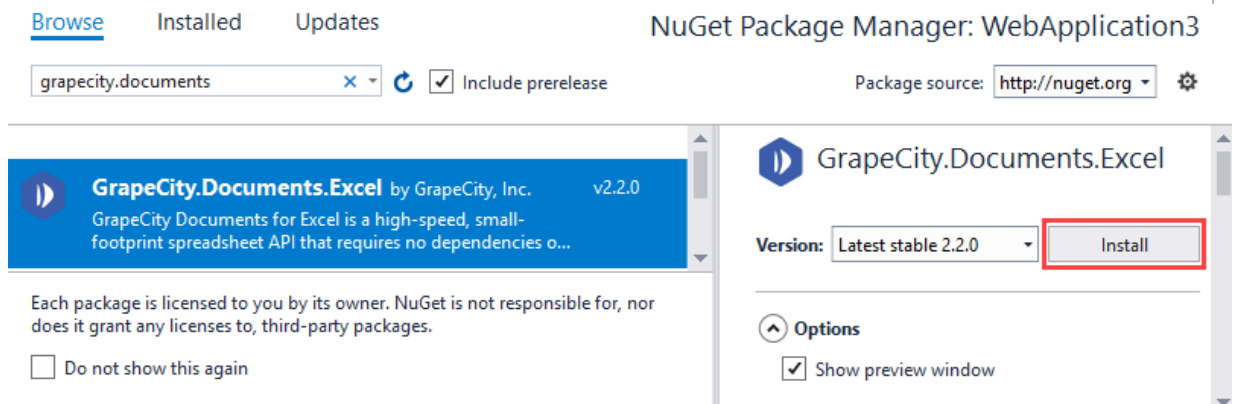
</Project>
```

.NET Core MVC Application

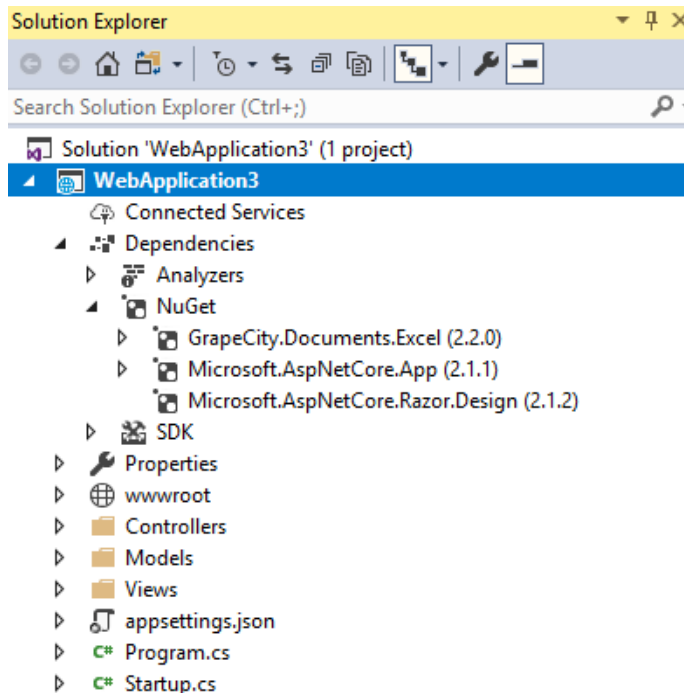
To find and install the GrapeCity.Documents.Excel NuGet package

Complete the following steps to add **GrapeCity.Documents.Excel** NuGet package in your Visual Studio application.

1. In **Solution Explorer**, right-click either **Dependencies** or a project and select **Manage NuGet Packages**.
2. In the **Browse** tab, select **nuget.org** from the **Package source** dropdown.
3. In the **Browse** tab, type "grapecity.documents" or "GrapeCity.Documents" in the search text box at the top and find the package "GrapeCity.Documents.Excel" as shown in the below image.




4. Click **Install** to install the **GrapeCity.Documents.Excel** package and its dependencies into the project. When the installation is complete, make sure you check the NuGet folder in your solution explorer and confirm whether or not the GrapeCity.Documents.Excel package is added to your project dependencies.



To manually create NuGet package source

In order to manually create Nuget feed source, you need to complete the following steps to add the Nuget feed URL to your Nuget settings in Visual Studio. Before you proceed with this step, make sure you first [download the GcExcel .NET Nupkg file](#) and put it in a local folder, for example - "D:\Nupkg".

1. From the Tools menu, select **NuGet Package Manager | Package Manager Settings**. The **Options** dialog box appears.
2. In the left pane, select **Package Sources**.
3. Click the  button in the top right corner. A new source is added under **Available Package Sources**.
4. Set a **Name** for the new package source.
5. To add source in the **Source** field, click the ellipsis button next to the Source field to browse for the **Nupkg** folder.
6. After you select the **Nupkg** folder, click the **Update** button and finally click OK.

To install the GrapeCity.Documents.Excel package using command line interface

1. Open the CommandPrompt window on your Windows system.
2. Use the **cd** command to navigate to your project folder. For example, `cd C:\Users\Admin\Documents\Visual Studio 2017\Projects\NETCoreMVCAApplication`.
3. Install GcExcel .NET NuGet package using the following command:
dotnet add package GrapeCity.Documents.Excel

To add GrapeCity.Documents.Excel package reference

GcExcel .NET is a cross-platform spreadsheet component that can be used on multiple platforms including Windows, Linux and Mac operating system.

In case you are creating an application using the Visual Studio, user can edit the *****.csproj** file and a package reference as shown in the image below:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Folder Include="wwwroot\" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="GrapeCity.Documents.Excel" Version="2.2.0" />
    <PackageReference Include="Microsoft.AspNetCore.App" />
    <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" PrivateAssets="All" />
  </ItemGroup>

</Project>
```

License Information

This topic includes:

- **Types of Licenses**
- **Apply License To GcExcel .NET**

Types of Licenses

GcExcel .NET supports the following types of license:

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

Unlicensed

When you download GcExcel for the first time, the product works under No-License i.e Unlicensed mode with a few limitations, that are highlighted below.

Maximum time of opening and saving Excel files

Every time a user runs an application, he/she can open or save up-to 100 Excel files using GcExcel .NET.

- If a user has opened 100 files, and trying to open the 101th file, exceptions will be thrown saying that you have exceeded the number of files you can open when the license is not found.
- If a user has saved 100 Excel files, and trying to save the 101th file, an Excel file with just a watermark sheet will be saved. The content of watermark tells users that no license is found.

Note that this limitation is triggered every time when users run the program, so that they can continue to open or save another 100 times after they restart their application.

Maximum Operating Time

While executing an application program, the duration of operating GcExcel .NET will last up-to 10 hours.

Once you complete the 10 hours of operation, you may notice the following:

- An exception will be thrown while creating an instance of Workbook, saying that you have exceeded the maximum operating time, and cannot create a new instance.
- The following API's will stop working.

API

IRange

IWorkbook.Worksheets.Add()

Remark

Throws an exception, same as create an instance of Workbook.

Returns null.

Note that this limitation will be reset every time when users run the program, so that they can continue to use these APIs after they restart their program.

Watermark Sheet

When saving an Excel file, a new worksheet with watermark will be added. This sheet will be the active sheet of your workbook. The content of the watermark will tell users that no license is found and will provide our sales and contact information so that you can directly connect to our support team.

When saving a PDF file, a PDF file with a watermark on the top of each exported page will be added. The content of the watermark

will tell users which license is applied and will provide our sales and contact information.

The following watermark will be displayed:

"Unlicensed copy of GrapeCity Documents for Excel, .NET Edition. Contact us.sales@grapecity.com to get your 30-day evaluation key to remove this text and other limitations."

Evaluation License

GcExcel .NET trial license is available for one month for users to evaluate the product and see how it can help with their comprehensive project requirements.

In order to evaluate the product, you can contact us.sales@grapecity.com and ask for the evaluation license key. The evaluation key is sent to users via email and holds valid for 30 days. After applying the evaluation license successfully, the product can be used without any limitations until the license date expires.

After the expired date, the following limitations will be triggered:

Cannot create new instance

When your evaluation license expires, an exception specifying that the evaluation license is expired will be thrown on creating a new instance of the workbook.

Open and Save Excel Files

- If a user opens an Excel file, an exception will be thrown saying that the evaluation license is expired.
- If a user saves a file, an Excel file with only the watermark sheet will be saved.

Save PDF Files

- If a user saves a PDF file, a PDF file with watermark on the top of each exported page will be saved.

API Limitations

The following API's will stop working after your evaluation license has expired:

API	Remark
IRange	Throws an exception, same as create an instance of Workbook.
IWorkbook.Worksheets.Add()	Returns null.

Watermark

When saving an Excel file, an Excel file with a watermark sheet will be saved. The content of watermark will tell users that no license is found and will provide our sales and contact information. When saving a PDF file, a PDF file with a watermark on the top of each exported page will be saved. The content of watermark will tell users which license is applied and will provide our sales and contact information.

In case you're using an evaluation license, the following watermark will appear:

"Expired Evaluation copy of GrapeCity Documents for Excel, .NET Edition. Contact us.sales@grapecity.com to purchase license."

Licensed

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

Watermark Sheet

No watermark will be displayed when you have a production license.

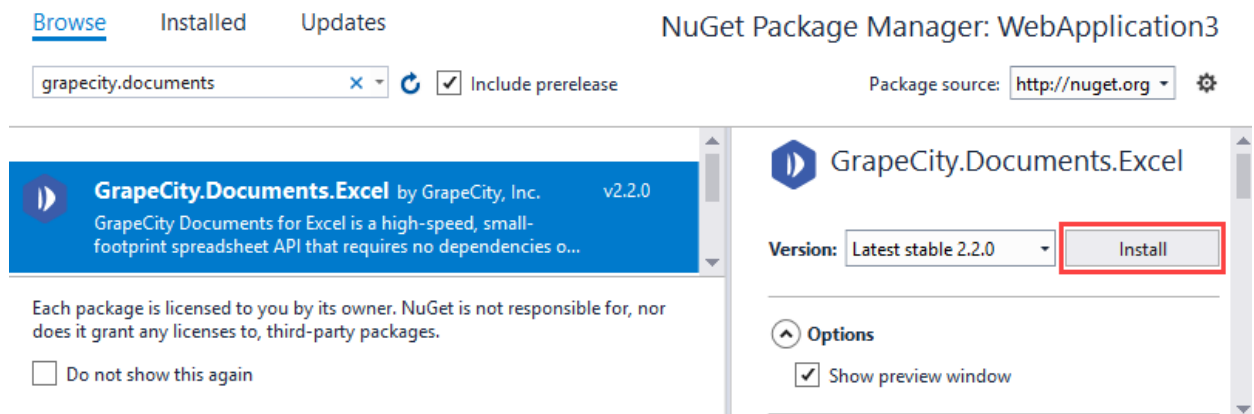
Apply License To GcExcel .NET

Applying license to **GcExcel .NET** involves completing the following steps to create and license a WorkBook:

- **Step 1: Create a new Web Application (.NET Core)**
- **Step 2: Add a Controller**
- **Step 3: Add a View**
- **Step 4: Build and Run the Project**

Step 1: Create a new Web Application (.NET Core)

1. In Visual Studio, select **File | New | Project** to create a new ASP.NET Core Web Application.
2. Under installed templates, select **Visual C# | Web | ASP.NET Core Web Application**, and then click **OK**.
3. In the **New ASP.NET Core Web Application(.NET Core)** dialog, select **Web Application**, and then click **OK**.
4. Add the GcExcel .NET references to the project. In the **Solution Explorer**, right click **Dependencies** and select **Manage NuGet Packages**. In **NuGet Package Manager**, select **nuget.org** as the Package source. Search for **GcExcel .NET** package, and click **Install**.



Step 2: Add a Controller

1. In the **Solution Explorer**, right click the folder **Controllers**.
2. From the context menu, select **Add | New Item**. The **Add New Item** dialog appears.
3. Complete the following steps in the **Add New Item** dialog:
 - Expand the **Installed** tab towards left, and select **ASP.NET|MVC Controller Class**.
 - Set name of the controller (For example: GcExcel .NETController).
 - Click **Add**.
4. Add the product license key by calling the **SetLicenseKey** ('SetLicenseKey Method' in the on-line documentation) method using the following code:

```
Workbook.SetLicenseKey("Your License Key");
```

- Add the following code to replace the Index() method.

SpreadController.cs

```
public IActionResult Index()
{
    //Apply license before using the API, otherwise it will be considered as
    no license.
    Workbook.SetLicenseKey("Your License Key");
    Workbook workbook = new Workbook();
    workbook.Open(@"C:\open.xlsx");
    workbook.Save(@"C:\save.xlsx");
    workbook.Save(@"C:\SavePDF.pdf", SaveFileFormat.Pdf);
    var worksheet = workbook.Worksheets[0];
}
```

A new controller is added to the application within the folder **Controllers**.

Step 3: Add a View

- a. From the **Solution Explorer**, right click the folder **Views** and select **Add | New Folder**.
- b. Name the new folder. Provide the same name as the name of your controller, minus the suffix Controller (in our example: GcExcel .NET).
- c. Right click the folder GcExcel .NET, and select **Add | New Item**. The **Add New Item** dialog appears.
- d. Complete the following steps in the **Add New Item** dialog:
 - a. Expand the **Installed** tab towards left, and select **ASP.NET|MVC View Page**.
 - b. Set name of the view (for example: Index.cshtml).
 - c. Click **Add**.
- e. Add the following code in Index.cshtml file.

Index.cshtml

```
<script src="http://code.jquery.com/jquery-3.2.1.min.js"></script>
<script type="text/javascript">
    $(document).ready(function () {
        alert("Workbook Saved !!!");
    });
</script>
```

Step 4: Build and Run the Project

- a. Click **Build | Build Solution** to build the project.
- b. Press **F5** to run the project. Once you execute the project, you will notice that **Workbook.xlsx** file is created at projects root location.

Upgrade to Latest Version

In order to upgrade the license of GcExcel.NET to the latest version 2.2.0, you need to set a new license key (the new version doesn't work with the license keys of older versions). This is a free upgrade for existing customers who already have licensed version of GcExcel .NET. The GrapeCity Sales team will provide existing users with the new license key via email.

After receiving the new license key, follow the steps shared below:

1. Open an existing .NET core application created with GcExcel.NET previous license.
2. Right-click the project in **Solution Explorer** and choose **Manage Nuget Packages**.
3. In the **Package Source** on top right, select **nuget.org**
4. Click **Updates** tab on the top to display the list of all the installed Nuget packages.
5. On the left panel, select the **Select all packages** check box and click **Update**.
6. In the **Preview Changes** dialog, click **Ok** and choose **I Accept** in the next screen.
7. Switch to the code view and replace the old key with the new license key received via email.
 - To upgrade the license of a particular instance:
var doc = new GcExcelDocument("new key");
 - To upgrade the license of all the instances:
GcExcelDocument.SetLicenseKey("new key");

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 deploy GcExcel .NET, you need to make sure that you have at least one of the following frameworks installed on your system:

- .NET Core 2.0+
- .NET Framework 4.6.1
- Mono 5.4

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

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	(800) 858-2739 or (412) 681-4343 outside the U.S.A.
Fax	(412) 681-4384

End User License Agreement

The GrapeCity licensing information, including the GrapeCity end-user license agreements, 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](#).

Features

GcExcel .NET provides some remarkable features that empowers users to include spreadsheets with high-performance attributes, customizable themes, configurable components, summarized data, application of custom styles, embedded drawing objects, integrated calculation engine, and so much more.

Below is a complete list of features that are available for use in GcExcel .NET. Each topic name refers to the corresponding topic in the documentation that gives you more information about that particular feature.

- [Comments](#)
- [Chart](#)
- [Conditional Format](#)
- [Data Validation](#)
- [Formula](#)
- [Group](#)
- [Hyperlinks](#)
- [Page Setup](#)
- [PDF Export](#)
- [Rich Text](#)
- [Shape and Picture](#)
- [Style](#)
- [Sparkline](#)
- [Theme](#)
- [Table](#)

Comments

GcExcel .NET enables users to annotate a worksheet by allowing them to write comments on cells in order to specify additional information about the data it contains.

For instance, let us assume you want users to enter only the numeric information in an individual cell of a worksheet. To accomplish this, instead of populating a small cell with large notes, it is more ideal to use a short comment (something like "Please enter only numeric characters in this cell") in order to provide additional context for the data represented in that cell.

The cells annotated with comments will display a small red indicator (at the corner of the cell) which appear when your mouse pointer is placed on that particular cell. The text in the comments can be edited, copied and formatted. Also, the comments can be moved, resized or deleted, can be made hidden or visible and their indicators can also be customized as per your preferences.

	A	B	C	D	E	F	G	H
1								
2								
3		Area	Jan	Feb	Mar	Total Sales (Q1)		
4		Washington	893	657	334	1884		
5		New York	456	777	213	1446		
6		Wales	534	433	434	1401		
7								
8								
9								
10								
11								
12								
13								
14								
15								

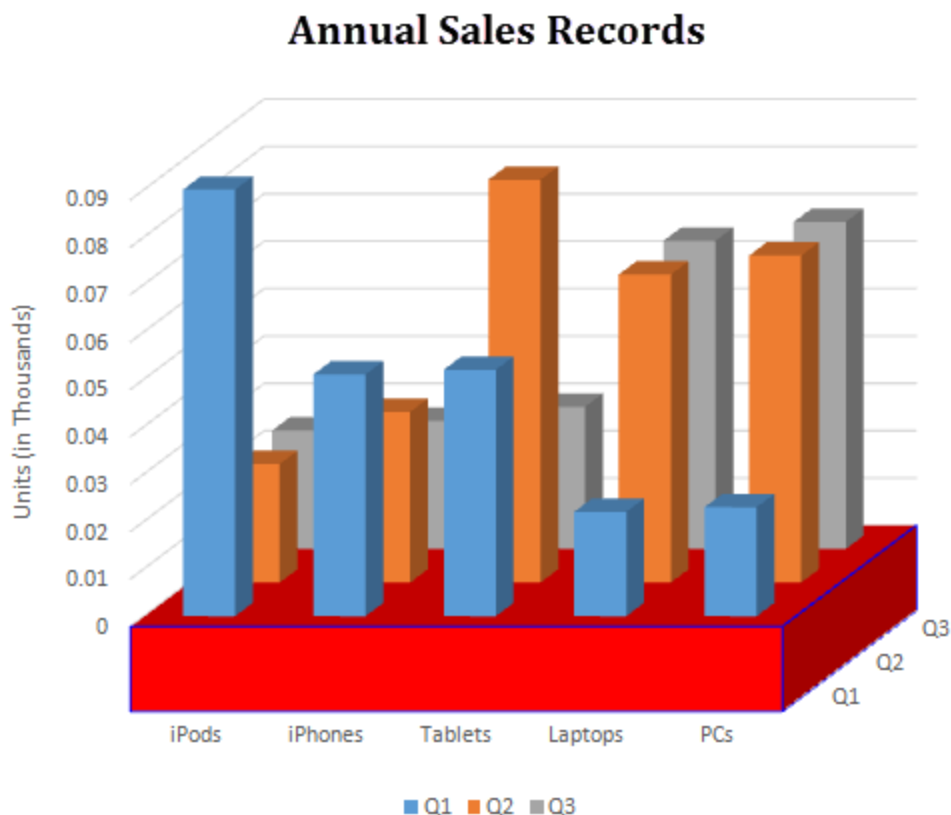
This is the highest sales figure for Quarter 1

This is the lowest sales figure for Quarter 1

For more information on comments, refer to [Apply Comments](#) in this documentation.

Chart

GcExcel .NET empowers users with the capability to graphically display information in charts so as to help business analysts compare numbers, analyse patterns and visualize trends quickly and efficiently.



For more information on using charts in GcExcel .NET, refer to [Work with Chart](#).

Conditional Format

In order to enable you to highlight important information in rows or columns of a worksheet, GcExcel .NET allows users to create conditional formatting rules for individual cells or a range of cells based on cell values. If the format condition matches with the cell value, it is assumed as true and the cell is formatted as per the specified rule.

For instance, let us assume you want a cell or a range of cells to appear in italics when the value entered in them is lower than 90. To accomplish this, you would apply a conditional formatting rule that will change the format of the cell as soon as the condition is met. Other cells will appear in general format which is the default format of the cells in a spreadsheet.

For more information, refer to [Apply Conditional Formatting](#).

Data Validation

GcExcel .NET provides users with the ability to validate data by restricting the type of information format and the values that can be entered in cells of a worksheet. You can create distinct validation scenarios for individual cells or a range of cells as per your requirements.

Using the data validation feature, you can perform the following tasks in a spreadsheet:

- Generate a list of entries by putting a check on the values allowed in cells.
- Prompt messages to describe the type of data values that can be entered in a cell.
- Figure out if entry in a particular cell or a range of cells is correct or not on the basis of calculations performed on other cells.
- Set a range of values (numeric or alphabetic) allowed in cells or a range of cells.
- Display error alert messages when invalid data is entered in a cell.

For more information on data validation, refer to [Apply Data Validations](#).

Formula

GcExcel .NET provides you with the ability to create and use formulas - expressions to calculate the value of a cell quickly and accurately, in the spreadsheets. You can also use some built-in functions and operators to generate formulas and calculate values in cells. For more information about formulas, refer to [Manage Formulas](#) in this documentation.

Also, GcExcel .NET supports the usage of both built-in and custom functions. You can either create a custom function or use an existing one to perform complex spreadsheet calculations without any hassle. For more information about formulas, refer to [Manage Custom Functions](#) in this documentation.

Group

Spreadsheets with tons of data can often become a hassle to read, consuming a lot of time in visualizing, managing and analyzing the information which is relevant for you.

GcExcel .NET provides you with the ability to summarize large amounts of information in groups so that complex spreadsheets are easier to navigate. After rows of data in a spreadsheet are grouped, users can expand and collapse details with the click of a square button placed in the group header row.

To see how grouping can be used in GcExcel .NET to organize data in groups, refer to [Apply Grouping](#).

Hyperlinks

GcExcel .NET allows users to create references to the data in the form of hypertext links that point towards another document or a section within the same document. A worksheet or a range can have multiple hyperlinks.

For more information on Hyperlinks in GcExcel .NET, refer to [Manage Hyperlinks](#).

Page Setup

GcExcel .NET supports the Page Setup options in order to enable users to manage printing in an efficient manner.

With different page set up options, you can customize the page layout including size, header, footer, margins, orientation etc. along with other important paper settings while printing.

For more information, see [Configure Print Settings via Page Setup](#).

PDF Export

GcExcel .NET allows users to export workbook to a PDF file. For each worksheet in the workbook, you can set pagination and export it to several pages in a PDF file.

Using this feature, you can export all the spreadsheets or any specific spreadsheet in a workbook to PDF format. You can also apply styles and customize fonts while performing the export operation.

For more information, see [Export to a PDF File](#).

Rich Text

GcExcel.NET provides users with the ability to set rich text in the cells of the worksheet. With the rich text feature, users can incorporate multiple styles to the text entered within a cell by highlighting important characters or alphabets using different colors, font family, font effects (bold, underline, double underline, strikethrough, subscript, superscript) and font size etc.

For more information on how to set rich text in a cell, refer to [Set Rich Text in a Cell](#).

Shape and Picture

GcExcel .NET provides you the ability to embed drawing objects like shapes and pictures on cells of a worksheet. You can draw and insert arrows, lines, pictures, general shapes etc.

To know how shape and picture can be integrated in a worksheet, refer to [Work with Shape And Picture](#) in this documentation.

Style

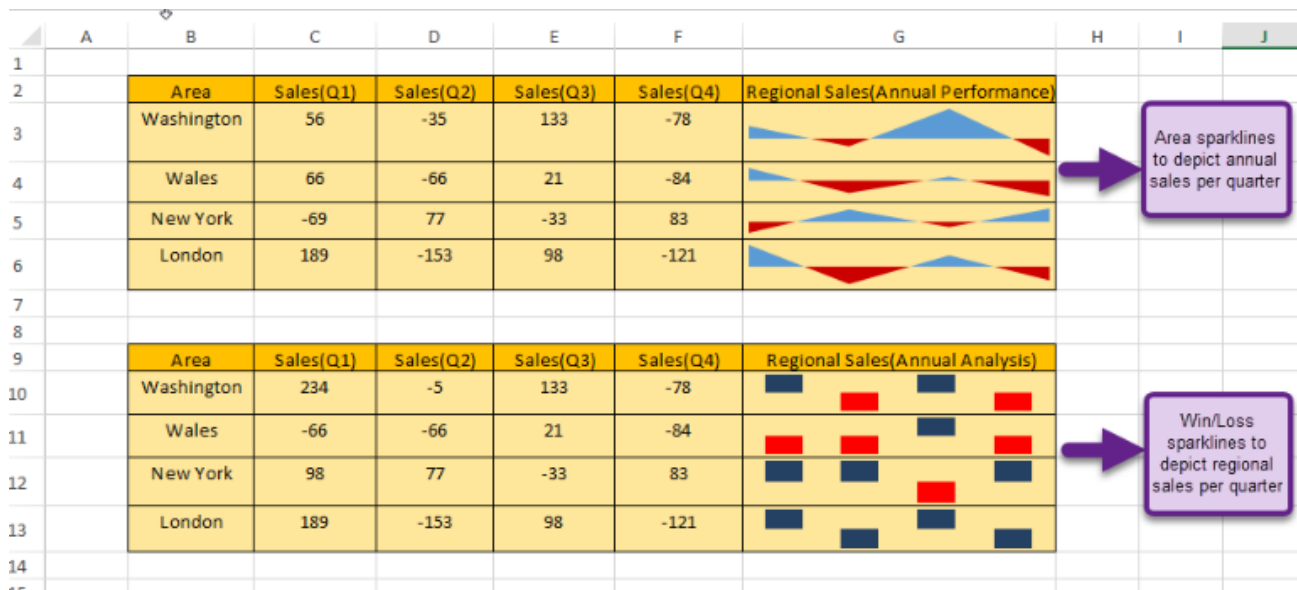
GcExcel .NET allows you to format the cells in a spreadsheet with custom styles. A cell style includes characteristics such as fill (solid fill, gradient fill, pattern fill), fonts, borders, name style, and display format. You can apply, create or remove style as and when required.

To know more about the cell styles in GcExcel .NET, refer to [Apply Style](#) in this documentation.

Sparkline

GcExcel .NET allows you to highlight specific information and see how it varies over time using Sparklines. Sparklines can be understood as small, lightweight charts that are drawn inside cells to quickly visualize data for improved analysis.

Sparklines fit inside a cell and use data from a range of cells which is specified at the time of creating it. Typically, they are placed next to the selected cell range in the spreadsheet in order to enhance readability of data. You can mark data values to depict high, low, first, last, and negative values with distinct colors as per your requirement.



For more information on Sparklines, refer to [Use Sparklines](#) in this documentation.

Theme

GcExcel .NET provides users with a set of built-in themes to enable them to change the overall appearance of the workbook. Besides, it also allows users to create custom theme and apply it in order to set up a workbook as per their own preferences and requirements.

When a theme is changed, it affects all areas including the theme font, theme color, range, chart title etc. For instance: if you apply a built-in or a custom theme to your workbook, it is likely that the color of the range as well as the font will also be changed in accordance to the modified theme.

For more information on Themes, please refer to [Apply Theme](#) in this documentation.

Table

GcExcel .NET allows users to manage and analyse related data easier and faster with the help of tables and pivot tables that can be created using a range of cells in a spreadsheet. Typically, a table consists of rows and columns that can be formatted and managed independently in a worksheet.

For more information on working with tables, refer to [Use Table](#) in this documentation.

For more information on working with pivot tables, refer to [Use Pivot Table](#) in this documentation.

Customize User Interaction

GcExcel.NET provides you the ability to customize the user interaction as per your own preferences. Whether you want to integrate calculation engine, modify default worksheet settings, apply custom styles on individual worksheets, perform specific operations on the data in the cells, or trigger workbook events – GcExcel.NET has got you covered.

The following list includes the tasks corresponding to customizing the way users interact with the spreadsheet in GcExcel .NET.

- [Manage Workbook](#)
- [Manage Worksheet](#)
- [Allow Sort](#)
- [Apply Filters](#)
- [Manage Hyperlinks](#)
- [Apply Grouping](#)
- [Apply Style](#)
- [Apply Comments](#)
- [Set Rich Text in a Cell](#)

Manage Workbook

A workbook is a spreadsheet document that comprises of one or more worksheets that are stored within the **Worksheets** collection.

GcExcel .NET provides all the necessary properties and methods required to create a workbook, perform complex operations on the data residing in the spreadsheets and make use of several workbook events that are triggered when called explicitly by the user through code

Managing a workbook involves the following tasks:

- [Create Workbook](#)
- [Open, Save and Protect Workbook](#)
- [Cut or Copy Across Sheets](#)
- [Apply Theme](#)
- [Enable or Disable Calculation Engine](#)
- [Work with Workbook Views](#)

Create Workbook

In GcExcel, you can create a new instance of a workbook by using the constructor of **Workbook ('Workbook Class' in the on-line documentation)** class.

A workbook may contain one or more worksheets that are kept in the Worksheets collection. By default, a workbook contains one empty worksheet with the default name **Sheet1**, which is created as soon as the user generates a new instance of the Workbook class.

Refer to the following example code to see how you can create a workbook using GcExcel.

C#

```
//Initialize the WorkBook  
Workbook workbook = new Workbook();
```

In order to add more worksheets to your workbook, refer to [Work with Worksheets](#) in this documentation.

Open, Save and Protect Workbook

Once you create a workbook, you can open the workbook to make modifications, save the changes back to the workbook and protect it with a password to ensure security.

This topic includes the following tasks:

- **Open a workbook**
- **Save a workbook**
- **Protect a workbook**

Open a workbook

You can open an existing workbook by calling the **Open ('Open Method' in the on-line documentation)** method of the Workbook class.

While opening a workbook, you can also choose from several import options listed in the below table:

Open Options	Description
Import Flags	NoFlag=0 Default
	Data=1 Read only the data from the worksheet
	Formulas=2 Read only the data, formula, defined names and table from the worksheet. Table is included for table formula.
DoNotRecalculateAfterOpened	Do not recalculate when getting formula value after loading the file. Default is false

Refer to the following example code to open a workbook.

C#

```
// Opening a workbook
workbook.Open(@"Source.xlsx", OpenFileFormat.Xlsx);

//Opening a workbook with Import options

//Import only data from .xlsx document.

XlsxOpenOptions options = new XlsxOpenOptions();
options.ImportFlags = ImportFlags.Data;
workbook.Open(@"DemoOpen.xlsx", options);

//Don't recalculate after opened.
XlsxOpenOptions options1 = new XlsxOpenOptions();
options1.DoNotRecalculateAfterOpened = true;
workbook.Open(@"DemoOpen.xlsx", options1);
```



Note: While opening the workbook, you can check whether it is password protected or not by using the **IsEncryptedFile ('IsEncryptedFile Method' in the on-line documentation)** method of the Workbook class. If your workbook is password protected, you would need to provide a password everytime you open it.

Save a workbook

You can save the changes made in the existing workbook by calling the **Save ('Save Method' in the on-line documentation)** method of the Workbook class.

Refer to the following example code to save your workbook.

C#

```
// Save the Excel file
workbook.Save(@"createWorkbook.xlsx", SaveFileFormat.Xlsx);
```

Protect a workbook

GcExcel provides you extensive security in terms of enabling users to protect a workbook by encrypting it with a password. This is important when you have a business critical workbook containing sensitive data that you don't want to share with everyone.

Refer to the following example code to make your worksheet password protected.

C#

```
// Save the Excel file and protect it using password.
xlsxSaveOptions options = new XlsxSaveOptions();
options.Password = "123456";
workbook.Save(@"createWorkbook.xlsx", options);
```

Cut or Copy Across Sheets

In GcExcel .NET, it is possible to cut or copy data across a range of cells or several worksheets without the need to copy and paste the information into each of the cells or sheets individually.

For instance, let's say you want the same title text to be put into different worksheets within a workbook. To accomplish this, if you type the text in one worksheet and copy, paste it into every other worksheet, the process can turn out to be both cumbersome and time-consuming.

A quick way of doing this would be to cut or copy information across cells or sheets using:

- The Copy method to copy rows, columns, or a range of cells and paste them to destination.
- The Cut method to cut rows, columns, or a range of cells and paste them to destination.

Copy across sheets

Refer to the following example code to perform copy operation in a workbook.

C#

```
// Copy across sheets
worksheet.Range["A5"].Copy(worksheet2.Range["A1"]);
```

Cut across sheets

Refer to the following example code to perform cut operation in a workbook.

C#

```
// Cut across sheets
worksheet.Range["A2"].Cut(worksheet2.Range["A3"]);
```

Enable or Disable Calculation Engine

GcExcel offers exceptional computing features with its built-in calculation engine that is capable of performing even the most complex operations on the data in the spreadsheets with complete accuracy and within fraction of seconds. This calculation engine can be integrated with spreadsheets to achieve the desired results. Some of the advantages of using a calculation engine are as follows:

1. **Bulk Data analysis:** Involves less programming to handle complex spreadsheet calculations and provides the ability to fetch data from cells within the spreadsheets, perform calculations on it and display results for unparalleled data analysis of tons of data.
2. **Ease of use:** Easy-to-configure calculation engine.
3. **Saves Time and Efforts:** Pre-defined functions and methods to reduce implementation time and efforts.

Enable calculation engine

Refer to the following example code to enable calculation engine.

C#

```
//enable calc engine.
worksheet2.Range["A1"].Value = 1;
worksheet2.Range["A2"].Formula = "=A1";
workbook.EnableCalculation = true;

//calc formula when get value. A2's value is 1d.
var value1 = worksheet2.Range["A2"].Value;
```

Disable calculation engine

Refer to the following example code to disable calculation engine.

C#

```
//disable calc engine.
workbook.EnableCalculation = false;
worksheet.Range["A1"].Value = 1;
worksheet.Range["A2"].Formula = "=A1";

//A2's value is 0.
var value = worksheet.Range["A2"].Value;
```

Apply Theme

You can apply either a built-in theme or a custom theme to a workbook. The default theme of a workbook is the standard Office theme. In GcExcel, the current theme of a workbook is represented by the **ITheme interface (on-line documentation)**.

To change the current theme of the workbook, you need to first get the existing theme using the indexer notation of the **Themes class (on-line documentation)**.

Applying theme in a workbook involves the following tasks:

- Apply built-in theme to the workbook
- Add a custom theme and set to workbook

Apply built-in theme to the workbook

In order to enable you to maintain consistency in the appearance across all the worksheets in the workbook, GcExcel offers a set of built-in themes for you to choose from.

Refer to the following example code to apply a built-in theme to the workbook.

C#

```
//Change workbook's theme to Berlin.
worksheet.Range["E10"].Value = "Test";
worksheet.Range["E10"].Font.ThemeColor = ThemeColor.Accent6;
worksheet.Range["E10"].Interior.ThemeColor = ThemeColor.Accent5;
workbook.Theme = Themes.Berlin;
```

Add a custom theme and set to workbook

You can use the **Theme** object constructor in order to add a custom theme. After you add your custom theme, you can apply it to your workbook.

Refer to the following example code to add a custom theme and apply it to the workbook.

C#

```
//Add custom theme

Theme theme = new Theme("testtheme"); // Base theme is office theme, if parameters
```

are not given

```
theme.ThemeColorScheme[ThemeColor.Light1].RGB = Color.AntiqueWhite;
theme.ThemeColorScheme[ThemeColor.Accent1].RGB = Color.AliceBlue;
theme.ThemeFontScheme.Major[FontLanguageIndex.Latin].Name = "Buxton Sketch";
theme.ThemeFontScheme.Minor[FontLanguageIndex.Latin].Name = "Segoe UI";
workbook.Theme = theme;

// Applying theme
worksheet.Range["E10"].Value = "CustomTest";
worksheet.Range["E10"].Font.ThemeColor = ThemeColor.Light1;
worksheet.Range["E10"].Interior.ThemeColor = ThemeColor.Accent1;
```

Work with Workbook Views

GcExcel allows users to personalize the display of the workbook. You can use the **BookView ('BookView Property' in the on-line documentation)** property of the **IWorkbook ('IWorkbook Interface' in the on-line documentation)** interface to set the view of the workbook as per your preferences.

The following properties of the **IWorkbookView ('IWorkbookView Interface' in the on-line documentation)** interface allows users to further customize various display settings in the workbook.

1. **DisplayHorizontalScrollBar ('DisplayHorizontalScrollBar Property' in the on-line documentation)** - This property gets and sets the display of the horizontal scrollbar.
2. **DisplayVerticalScrollBar ('DisplayVerticalScrollBar Property' in the on-line documentation)** - This property gets and sets the display of the vertical scrollbar.
3. **DisplayWorkbookTabs ('DisplayWorkbookTabs Property' in the on-line documentation)** - This property gets and sets if the workbook tabs are displayed.
4. **TabRatio ('TabRatio Property' in the on-line documentation)** - This property gets and sets the ratio of the width of the tab area (of the workbook) to the width of the horizontal scroll bar (of the worksheet). The value of TabRatio can be any number between 0 and 1. By default, if the TabRatio is not set, the value is 0.6.

Refer to the following code snippet to set workbook view and customize other display settings.

C#

```
//Set workbook view

IWorkbook workbook = new Workbook();
var bookView = workbook.BookView;
bookView.DisplayHorizontalScrollBar = true;
bookView.DisplayVerticalScrollBar = true;
bookView.DisplayWorkbookTabs = true;
bookView.TabRatio = 0.8;
```

Manage Worksheet

A worksheet is a matrix of cells where you can enter and display data, analyse information, write formulas, perform calculations and review results. The cells in a worksheet are defined by rows (represented by numeric characters like 1,2,3) and columns ((represented by alphabetical letters like A,B,C etc.). For instance, in a worksheet, C6 represents the cell in column C and row 6.

In GcExcel .NET, you can use the methods of **IWorksheets ('IWorksheets Interface' in the on-line documentation)** to execute different tasks in a spreadsheet including insertion of a new worksheet in the workbook, deletion of a worksheet from the collection, assigning an active sheet, and so much more.

Managing a worksheet involves the following tasks:

- [Work with Worksheets](#)
- [Range Operations](#)
- [Freeze Panes in a Worksheet](#)

- [Work with Shape And Picture](#)
- [Customize Worksheets](#)
- [Work with Worksheet Views](#)

Work with Worksheets

While working with worksheets, you can perform the following operations to accomplish several important tasks in a workbook.

- **Access the default worksheet**
- **Add multiple worksheets**
- **Activate a worksheet**
- **Access a worksheet**
- **Protect a worksheet**
- **Delete worksheet**
- **Copy and Move worksheet**

Access the default worksheet

Whenever a new workbook is created, an empty worksheet with the name **Sheet1** is automatically added to the workbook. This worksheet is known as the default worksheet. For every workbook, only one default worksheet is added to it.

Refer to the following example code in order to access the default worksheet in your workbook.

C#

```
// Fetch the default WorkSheet
IWorksheet worksheet = workbook.Worksheets[0];
```

Add multiple worksheets

A workbook may contain any number of worksheets. You can add one or more worksheets before or after a specific sheet in your workbook.

Refer to the following example code to insert multiple worksheets in a workbook.

C#

```
//Initialize the WorkBook and add multiple WorkSheets
IWorksheet worksheet = workbook.Worksheets.Add();
IWorksheet worksheet2 = workbook.Worksheets.AddAfter(worksheet);
IWorksheet worksheet3 = workbook.Worksheets.AddBefore(worksheet2);
```

Activate a worksheet

While working with multiple worksheets in a workbook, you may require to make the current sheet to workbook's active sheet so as to execute certain operations on that particular worksheet. This can be done using the **Activate** ('**Activate Method**' in the on-line documentation) method of the **IWorksheet interface**. ('**IWorksheet Interface**' in the on-line documentation)

Refer to the following example code to activate a worksheet.

C#

```
IWorksheet worksheet3 = workbook.Worksheets.Add();

//Activate new created worksheet.
worksheet3.Activate();
```

Access a worksheet

All the worksheets within a workbook are stored in **Worksheets** collection. In order to access a specific worksheet

within a workbook, you can choose either of the two ways : using the **Index property (on-line documentation)** or using the **Name property (on-line documentation)** of the **IWorksheet interface (on-line documentation)**.

Refer to the following example code to access a worksheet within the workbook.

C#

```
//Use sheet index to access the worksheet.
IWorksheet worksheet4 = workbook.Worksheets[0];

////Use sheet name to access the worksheet.
IWorksheet worksheet5 = workbook.Worksheets["SampleSheet5"];
//worksheet5.Name = "SampleSheet5";
```

Protect a worksheet

To ensure the data lying in the cells of your spreadsheet is safe and can't be modified by anyone, you can protect your worksheet by transforming it into a read-only sheet. Further, you can use the properties of the **IProtectionSettings ('IProtectionSettings Interface' in the on-line documentation)** interface to explicitly setup your protected worksheet the way you want. Later, if you want to remove protection, you can unprotect your worksheet by setting the protection property to false.

Refer to the following example code to protect or unprotect a worksheet in GcExcel.

C#

```
//protect worksheet, allow insert column.
worksheet3.Protection = true;
worksheet3.ProtectionSettings.AllowInsertingColumns = true;

//Unprotect worksheet.
worksheet3.Protection = false;
```

Delete Worksheet

You can remove one or more worksheets from a workbook. When you delete a worksheet, it automatically gets deleted from the Worksheets collection.

Refer to the following example code to delete a specific sheet from the workbook.

C#

```
IWorksheet worksheet7 = workbook.Worksheets.Add();

//workbook must contain one visible worksheet at least, if delete the one visible
//worksheet, it will throw exception.
worksheet7.Delete();
```

Copy and Move Worksheet

You can copy the current spreadsheet on which you're working as well as copy a worksheet between workbooks and then move them to a specific location as per your custom requirements and preferences. This can be done by using the **Copy()** ('Copy Method' in the on-line documentation) method, the **CopyAfter()** ('CopyAfter Method' in the on-line documentation) method, the **CopyBefore()** ('CopyBefore Method' in the on-line documentation) method, the **Move()** ('Move Method' in the on-line documentation) method, the **MoveBefore()** ('MoveBefore Method' in the on-line documentation) method and the **MoveAfter()** ('MoveAfter Method' in the on-line documentation) method of the **IWorksheet interface**. Using these methods, the worksheet can easily be copied and relocated by placing it within the same workbook or another workbook as and when you want.

Refer to the following example code in order to copy a worksheet.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
```

```
// Fetch the active worksheet
IWorksheet worksheet = workbook.ActiveSheet;

object[,] data = new object[,] {
    {"Name", "City", "Birthday", "Sex", "Weight", "Height", "Age"},
    {"Bob", "newyork", new DateTime(1968, 6, 8), "male", 80, 180, 56},
    {"Betty", "newyork", new DateTime(1972, 7, 3), "female", 72, 168, 45},
    {"Gary", "NewYork", new DateTime(1964, 3, 2), "male", 71, 179, 50},
    {"Hunk", "Washington", new DateTime(1972, 8, 8), "male", 80, 171, 59},
    {"Cherry", "Washington", new DateTime(1986, 2, 2), "female", 58, 161, 34},
    {"Coco", "Virginia", new DateTime(1982, 12, 12), "female", 58, 181, 45},
    {"Lance", "Chicago", new DateTime(1962, 3, 12), "female", 49, 160, 57},
    {"Eva", "Washington", new DateTime(1993, 2, 5), "female", 71, 180, 81}};

// Set data
worksheet.Range["A1:G9"].Value = data;

// Copy the active sheet to the end of current workbook
var copy_worksheet = worksheet.Copy();
copy_worksheet.Name = "Copy of " + worksheet.Name;

// Saving workbook to xlsx
workbook.Save(@"CopyWorkSheet.xlsx", SaveFileFormat.Xlsx);
```

Refer to the following example code in order to copy a worksheet between the workbooks.

C#

```
// Create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();

// Create another source_workbook
Workbook source_workbook = new Workbook();

// Fetch the active worksheet
IWorksheet worksheet = source_workbook.ActiveSheet;
object[,] data = new object[,] {
    {"Name", "City", "Birthday", "Sex", "Weight", "Height", "Age"},
    {"Bob", "newyork", new DateTime(1968, 6, 8), "male", 80, 180, 56},
    {"Betty", "newyork", new DateTime(1972, 7, 3), "female", 72, 168, 45},
    {"Gary", "NewYork", new DateTime(1964, 3, 2), "male", 71, 179, 50},
    {"Hunk", "Washington", new DateTime(1972, 8, 8), "male", 80, 171, 59},
    {"Cherry", "Washington", new DateTime(1986, 2, 2), "female", 58, 161, 34},
    {"Coco", "Virginia", new DateTime(1982, 12, 12), "female", 58, 181, 45},
    {"Lance", "Chicago", new DateTime(1962, 3, 12), "female", 49, 160, 57},
    {"Eva", "Washington", new DateTime(1993, 2, 5), "female", 71, 180, 81}};

// Set data
worksheet.Range["A1:G9"].Value = data;

// Copy data of active sheet from source workbook to current workbook before Sheet1
var copy_worksheet = worksheet.CopyBefore(workbook.Worksheets[0]);
copy_worksheet.Name = "Copy of Sheet1";
copy_worksheet.Activate();

// Saving workbook to xlsx
workbook.Save(@"CopyWorkSheetBetweenWorkBooks.xlsx", SaveFileFormat.Xlsx);
```

Range Operations

Range refers to a cell or a collection of cells and range operations are the operations performed on those cell collection using single line of code. The **Range ('Range Property' in the on-line documentation)** property of **IWorksheet ('IWorksheet Interface' in the on-line documentation)** allows you to execute multiple operations on cells, rows or columns.

The operations that can be handled using Range property are as follows:

- [Access a Range](#)
- [Access Areas in a Range](#)
- [Access Cells, Rows and Columns in a Range](#)
- [Cut or Copy Cell Ranges](#)
- [Cut or Copy Shape, Slicer, Chart and Picture](#)
- [Find and Replace Data](#)
- [Get Row and Column Count](#)
- [Hide Rows and Columns](#)
- [Insert And Delete Cell Ranges](#)
- [Insert and Delete Rows and Columns](#)
- [Merge Cells](#)
- [Set Values to a Range](#)
- [Set Row Height and Column Width](#)
- [Auto Fit Row Height and Column Width](#)
- [Work with Used Range](#)

Access a Range

Range refers to an array of cells defined in a spreadsheet.

GcExcel allows users to define a range and then access the rows and columns within the range to perform certain tasks like formatting of cells, merging of cells, insertion or deletion of cells along with other useful operations.

Refer to the following example code in order to access a range using different methods.

C#

```
//Use index to access cell A1.
worksheet.Range[0, 0].Interior.Color = Color.LightGreen;

//Use index to access range A1:B2
worksheet.Range[0, 0, 2, 2].Value = 5;

//Use string to access range.
worksheet.Range["A2"].Interior.Color = Color.LightYellow;
worksheet.Range["C3:D4"].Interior.Color = Color.Tomato;
worksheet.Range["A5:B7, C3, H5:N6"].Value = 2;

//Use index to access rows
worksheet.Rows[2].Interior.Color = Color.LightSalmon;

//Use string to access rows
worksheet.Range["4:4"].Interior.Color = Color.LightSkyBlue;

//Use index to access columns
worksheet.Columns[2].Interior.Color = Color.LightSalmon;

//Use string to access columns
worksheet.Range["D:D"].Interior.Color = Color.LightSkyBlue;

//Use Cells to access range.
worksheet.Cells[5].Interior.Color = Color.LightBlue;
```

```
worksheet.Cells[5, 5].Interior.Color = Color.LightYellow;

//Access all rows in worksheet
var allRows = worksheet.Rows.ToString();

//Access all columns in worksheet
var allColumns = worksheet.Columns.ToString();

//Access the entire sheet range
var entireSheet = worksheet.Cells.ToString();
```

Access Areas in a Range

While working with a large worksheet having non-contiguous selections, you can access specific areas in a multiple-area range by using the indexer notation of the **IAreas ('IAreas Interface' in the on-line documentation)** interface. The **Count ('Count Property' in the on-line documentation)** property of the IAreas interface represents the area count (number of areas) of the multiple-area range.

The **Areas ('Areas Property' in the on-line documentation)** property of the **IRange ('IRange Interface' in the on-line documentation)** interface represents all the selected ranges in the multiple area range.

Refer to the following example code to access areas in a range.

C#

```
//area1 is A5:B7.
var area1 = worksheet.Range["A5:B7,C3,H5:N6"].Areas[0];

//set interior color for area1
area1.Interior.Color = Color.Pink;

//area2 is C3.
var area2 = worksheet.Range["A5:B7,C3,H5:N6"].Areas[1];

//set interior color for area2
area2.Interior.Color = Color.LightGreen;

//area3 is H5:N6.
var area3 = worksheet.Range["A5:B7,C3,H5:N6"].Areas[2];

//set interior color for area3
area3.Interior.Color = Color.LightBlue;
```

Access Cells, Rows and Columns in a Range

You can access cells, rows and columns in a range by using the **Cells ('Cells Property' in the on-line documentation)** property, **Rows ('Rows Property' in the on-line documentation)** property and **Columns ('Columns Property' in the on-line documentation)** property of the **IRange ('IRange Interface' in the on-line documentation)** interface.

Refer to the following example code in order to access cells, rows and columns in a worksheet.

C#

```
var range = worksheet.Range["A5:B7"];

//Set value for cell A7.
range.Cells[4].Value = "A7";

//Cell is B6
range.Cells[1, 1].Value = "B6";
```



```
//Row count is 3 and range is A6:B6.
var rowCount = range.Rows.Count;
var row = range.Rows[1].ToString();

//Set interior color for row range A6:B6.
range.Rows[1].Interior.Color = Color.LightBlue;

//Column count is 2 and range is B5:B7.
var columnCount = range.Columns.Count;
var column = range.Columns[1].ToString();

//Set values for column range B5:B7.
range.Columns[1].Interior.Color = Color.LightSkyBlue;

//Entire rows are from row 5 to row 7
var entireRow = range.EntireRow.ToString();

//Entire columns are from column A to column B
var entireColumn = range.EntireColumn.ToString();
```

Cut or Copy Cell Ranges

GcExcel .NET provides users with the ability to cut or copy a cell or a range of cells from a specific area and paste it into another area within the same worksheet. Also, users can choose from multiple paste options and also combine different paste options while copying the data from the cell range.

You can refer to the following sections in order to cut or copy data from the cell range.

- **Copy Cell Range**
- **Working With Paste Options**
- **Cut Cell Range**

In order to cut or copy data across multiple sheets, refer to [Cut or Copy Across Sheets](#).

Copy cell range

GcExcel allows you to copy a cell or a range of cells in the worksheets by calling **Copy ('Copy Method' in the on-line documentation)** method of **IRange ('IRange Interface' in the on-line documentation)**. To copy a single cell or a range of cells, specify the cell range to be copied, for example **B3:D12**.

GcExcel provides the following different ways to use the Copy method.

Example

Description

Copy(sheet.Range["E5"])

This method copies data from cell range **B3:D12** and pastes the data to cell **E5** onwards.

Copy(sheet.Range["E5:G14"])

This method copies data from cell range **B3:D12** and pastes the data in cell range **E5:G14**. In case the range of cells copied does not fit into the destination cell range, the data is lost.

Refer to the following example code in order to copy the cell range in a workbook.

C#

```
// Copy the data of the range of cells
worksheet.Range["B3:D12"].Copy(worksheet.Range["E5"]);
//Or
worksheet.Range["B3:D12"].Copy(worksheet.Range["E5:G14"]);
```

Working With Paste Options

Users can choose from several paste options while copying the data from the cell range. The **PasteType**

('PasteType Enumeration' in the on-line documentation) enumeration can be used to work with multiple paste options as described in the table shared below.

Option	Description
Default	This option can be used to paste all the cell data to the destination range except the row heights and column widths.
Values	This option can be used to paste only the cell value to the destination.
Formulas	If you're working in a formula cell, this option can be used to paste the formula to the destination . However, for a non-formula cell, this option pastes the cell value to the destination.
Formats	This option can be used to paste formats.
NumberFormats	This option can be used to paste number formats.
RowHeights	This option can be used to paste the row height to the destination.
ColumnWidths	This option can be used to paste the column width to the destination.

Users can also combine the two different paste options. For instance - if users want to paste values and number formats concurrently in the worksheet, then they can use combinations like : **PasteType.Values | PasteType.NumberFormats** , **PasteType.Formulas | PasteType.NumberFormats**. Similarly other paste options can also be combined with each other.

Refer to the following example code in order to use the combination of paste options while copying data from the cell range in a workbook and paste it to the destination.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Set data of PC
worksheet.Range["A2"].Value = "PC";
worksheet.Range["A4:C4"].Value = new string[]
{
    "Device", "Quantity", "Unit Price"
};
worksheet.Range["A5:C10"].Value = new object[,]
{
    { "T540p", 12, 9850 },
    { "T570", 5, 7460 },
    { "Y460", 6, 5400 },
    { "Y460F", 8, 6240 }
};

// Set style
worksheet.Range["A2"].RowHeight = 30;
worksheet.Range["A2"].ColumnWidth = 40;
worksheet.Range["A2"].Font.Size = 20;
worksheet.Range["A2"].Font.Bold = true;
worksheet.Range["A4:C4"].Font.Bold = true;
worksheet.Range["A4:C4"].Font.Color = Color.White;
worksheet.Range["A4:C4"].Interior.Color =
Color.LightBlue;
worksheet.Range["A5:C10"].Borders[BordersIndex.InsideHorizontal].Color =
Color.Orange;
worksheet.Range["A5:C10"].Borders[BordersIndex.InsideHorizontal].LineStyle =
BorderLineStyle.DashDot;

// Copy only style and row height from cells A2:C10
```

```

worksheet.Range["H1"].Value = "Copy style & row height from previous cells.";
worksheet.Range["H1"].Font.Color = Color.Red;
worksheet.Range["H1"].Font.Bold = true;
worksheet.Range["A2:C10"].Copy(worksheet.Range["H2"],
PasteType.Formats | PasteType.ColumnWidths);

// Set data of mobile devices
worksheet.Range["H2"].Value = "Mobile";
worksheet.Range["H4:J4"].Value = new string[]
{
    "Device", "Quantity", "Unit Price"
};
worksheet.Range["H5:J10"].Value = new object[,]
{
    { "HW-P30", 20, 4200 },
    { "IPhone-X", 5, 9888 },
    { "IPhone-6s plus", 15, 6880 }
};

// Add new sheet
IWorksheet worksheet2 = workbook.Worksheets.Add();

// Copy only style of Cell A2:C10 to new sheet
worksheet.Range["A2:C10"].Copy(worksheet2.Range["A2"],
PasteType.Formats | PasteType.ColumnWidths);
worksheet2.Range["A3"].Value = "Copy style from sheet1.";
worksheet2.Range["A3"].Font.Color = Color.Red;
worksheet2.Range["A3"].Font.Bold = true;

// Saving workbook to xlsx
workbook.Save(@"PasteOptionsEnhancements.xlsx", SaveFileFormat.Xlsx);

```

Cut cell range

GcExcel allows you to cut a cell or a range of cells in the worksheet by calling the **Cut ('Cut Method' in the on-line documentation)** method of the **IRange ('IRange Interface' in the on-line documentation)** interface. To cut a cell or the range of cells, specify the cell range to be moved, for example **B3:D12**.

GcExcel provide the following different ways to use Cut method.

Example

Cut(sheet.Range["E5"])

Description

This method cuts the data from cell range **B3:D12** and pastes the data to cell **E5** onwards.

Cut(sheet.Range["E5:G14"])

This method cuts the data from cell range **B3:D12** and pastes the data in cell range **E5:G14**. In case the range of cells cut does not fit into the destination cell range, the data is lost.

Refer to the following example code to cut a range of cells in the workbook.

C#

```

// Cut the data of the range of cell
worksheet.Range["B3:D12"].Cut(worksheet.Range["E5"]);
// Or
worksheet.Range["B3:D12"].Cut(worksheet.Range["E5:G14"]);

```

Cut or Copy Shape, Slicer, Chart and Picture

GcExcel allows users to cut or copy shapes, charts, slicers and pictures from one workbook to another and from one worksheet to another.

In order to perform the copy operation, you can use the **Copy()** ('Copy Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface.

In order to perform the cut operation, you can use the **Cut()** ('Cut Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface.

Refer to the following example code to see how you can cut or copy shape, slicer, chart and picture.

C#

```
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];

//Create a shape in worksheet, shape's range is Range["A7:B7"]
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 1, 1, 100, 100);

//Range["A1:D10"] contains Range["A7:B7"], copy a new shape to Range["C1:F7"]
worksheet.Range["A1:D10"].Copy(worksheet.Range["C1"]);
worksheet.Range["A1:D10"].Copy(worksheet.Range["C1:G9"]);

//Range["A1:D10"] contains Range["A7:B7"], cut a new shape to Range["C1:F7"]
worksheet.Range["A1:D10"].Cut(worksheet.Range["C1"]);
worksheet.Range["A1:D10"].Cut(worksheet.Range["C1:G9"]);

// Cross-sheet cut, copy operation

Workbook workbook1 = new Workbook();
IWorksheet worksheet1 = workbook1.Worksheets[0];
IWorksheet worksheet2 = workbook1.Worksheets.Add();

//Create a shape in worksheet, shape's range is Range["A7:B7"]
IShape Shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 1, 1, 100, 100);

//Range["A1:D10"] contains Range["A7:B7"]. Copy a new shape to worksheet2's
Range["C1:F7"]
worksheet1.Range["A1:D10"].Copy(worksheet2.Range["C1"]);
worksheet1.Range["A1:D10"].Copy(worksheet2.Range["C1:G9"]);

//Range["A1:D10"] contains Range["A7:B7"]. Cut a new shape to worksheet2's
Range["C1:F7"]
worksheet1.Range["A1:D10"].Cut(worksheet2.Range["C1"]);
worksheet1.Range["A1:D10"].Cut(worksheet2.Range["C1:G9"]);
```

In order to duplicate a shape to the current worksheet, you can use the **Duplicate()** ('Duplicate Method' in the on-line documentation) method of the **IShape** ('IShape Interface' in the on-line documentation) interface.

Refer to the following example code to see how you duplicate an existing shape, slicer, chart and picture.

C#

```
//Create shape, chart, slicer, picture
IShape Shape1 = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 100, 100, 200, 200);
IShape chart = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 300, 300, 300);
ISlicerCache cache1 = workbook.SlicerCaches.Add("Category", "catel");
ISlicer slicer = cache1.Slicers.Add(workbook.Worksheets["Sheet1"], "catel", "Category", 300, 300, 100, 200);
IShape picture = worksheet.Shapes.AddPicture("C:/Pictures", 1, 1, 100, 100);

//Duplicate shape
IShape newShape = Shape1.Duplicate();
//Duplicate chart
IShape newShape1 = chart.Duplicate();
//Duplicate slicer
```

```
slicer.Shape.Duplicate();
//Duplicate picture
IShape newPicture = picture.Duplicate();
```

Find and Replace Data

In a spreadsheet with hundreds of rows and columns, it becomes difficult to look for specific chunks of data across the entire worksheet and even more cumbersome to edit this information. The find and replace feature makes it easy for users to locate information and replace it within seconds, thereby saving both time and efforts.

GcExcel .NET enables users to locate data in a cell range, find specific information (and all its occurrences) across the worksheet and replace it with the desired information. Using this feature, you can find and replace specific values and formulas in a range as per custom requirements and preferences with the help of the following methods.

- The **Find ('Find Method' in the on-line documentation)** method of the **IRange ('IRange Interface' in the on-line documentation)** interface can be used to find the first, next or the previously matched cell range.
- The **Replace ('Replace Method' in the on-line documentation)** method of the **IRange ('IRange Interface' in the on-line documentation)** interface can be used to replace the data within the cell range.

Users can find basic information, locate cells with different formats, search data using various options, enumerate all occurrences across the worksheet, match the number of bytes occupied by the data and look for specific data in different places including comments, formula and text. Further, you can replace the basic information, replace via executing the search operation in loop and also replace using several options (like match case, match whole word and match byte).

Refer to the following example code in order to find cells in a target range starting from multiple positions and replace it with the desired information.

C#

```
// This example finds the word "newyork" in multiple searchRanges & replaces it with
"NewYork"

// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

object[,] data = new object[,] {
    {"Name", "City", "Birthday", "Sex", "Weight", "Height", "Age"},
    {"Bob", "newyork", new DateTime(1968, 6, 8), "male", 80, 180, 56},
    {"Betty", "newyork", new DateTime(1972, 7, 3), "female", 72, 168, 45},
    {"Gary", "NewYork", new DateTime(1964, 3, 2), "male", 71, 179, 50},
    {"Hunk", "Washington", new DateTime(1972, 8, 8), "male", 80, 171, 59},
    {"Cherry", "Washington", new DateTime(1986, 2, 2), "female", 58, 161, 34},
    {"Coco", "Virginia", new DateTime(1982, 12, 12), "female", 58, 181, 45},
    {"Lance", "Chicago", new DateTime(1962, 3, 12), "female", 49, 160, 57},
    {"Eva", "Washington", new DateTime(1993, 2, 5), "female", 71, 180, 81}};

// Set data
worksheet.Range["A1:G9"].Value = data;
worksheet.Range["I10:P19"].Value = data;
worksheet.Range["A21:G29"].Value = data;

object what = "newyork";
object replacement = "NewYork";
ReplaceOptions ro = new ReplaceOptions();
ro.MatchCase = true;
```

```
// Specify multiple ranges to search in
IRange searchRange = worksheet.Range["A1:G9, I10:P19"];

// Using Replace method to replace content in a specific range
searchRange.Replace(what, replacement, ro);

// Saving workbook to xlsx
workbook.Save(@"FindAndReplaceContentUsingReplaceOptions.xlsx", SaveFileFormat.Xlsx);
```

Refer to the following example code in order to find cells with the formula "SUM" and replace it with another formula "PRODUCT" simultaneously.

C#

```
// This code finds the "SUM" keyword & replaces it with "PRODUCT" keyword in the
// formula.

// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Set formulas
worksheet.Range["A1:H5"].Formula = "SUM(6,10)";

FindOptions fo = new FindOptions();
fo.LookIn = FindLookIn.Formulas;

IRange range = null;

// Specify range to search in formulas
IRange searchRange = worksheet.Range["A1:B4"];
do
{
    range = searchRange.Find("SUM", range, fo);
    if (range != null)
    {
        // using Replace method to replace formula in searched range
        range.Formula = range.Formula.Replace("SUM", "PRODUCT");
    }
} while (range != null);

// Saving workbook to xlsx
workbook.Save(@"FindAndReplaceFormulasUsingFindOptions.xlsx", SaveFileFormat.Xlsx);
```

Get Row and Column Count

In a large worksheet, manually fetching the number of rows and columns can be a tedious task.

GcExcel allows users to quickly get the row and column count of the specific areas or all the areas in a range.

The **Count ('Count Property' in the on-line documentation)** property of the **IRange ('IRange Interface' in the on-line documentation)** interface represents the cell count of all the areas in a range.

Refer to the following example code in order to get the row count and column count in a worksheet.

C#

```
var range = worksheet.Range["A5:B7"];

//cell count is 6.
```

```
var cellcount = range.Count;
//cell count is 6.
var cellcount1 = range.Cells.Count;
//row count is 3.
var rowcount = range.Rows.Count;
//column count is 2.
var columncount = range.Columns.Count;
```

Hide Rows and Columns

You can choose whether to hide or show rows and columns in a worksheet by using the **Hidden ('Hidden Property' in the on-line documentation)** property of the **IRange ('IRange Interface' in the on-line documentation)** interface.

Refer to the following example code in order to hide specific rows and columns in a worksheet.

C#

```
worksheet.Range["E1"].Value = 1;

//Hide row 2:6 using the Hidden property

worksheet.Range["2:6"].Hidden = true;

//Hide column A:D using the Hidden property
worksheet.Range["A:D"].Hidden = true;
```



Note: The range must either be entire rows or entire columns. The Hidden property doesn't work on a range of cells.

Insert And Delete Cell Ranges

GcExcel enables you to insert and delete a cell or a range of cells in order to help customization of worksheets as per your requirements.

Insert cell range

GcExcel allows you to add a cell or a range of cells in a worksheets by calling the **Insert ('Insert Method' in the on-line documentation)** method of **IRange ('IRange Interface' in the on-line documentation)**. To add a cell or a range of cells, specify the cell range, for example **A3** for single cell or **A3:A5** for a range of cells.

GcExcel provides following different options to insert a cell or a range of cells.

Method	Description
Insert	This method automatically inserts a cell or a range of cells.
Insert(InsertShiftDirection.Down)	This method inserts the range of cells and shifts the existing range of cells in downward direction.
Insert(InsertShiftDirection.Right)	This method insert the range of cells and shifts the existing range of cells to the right.

Refer to the following example code to see how you can insert a single cell and a cell range in the worksheet.

C#

```
//Insert the range of cell
worksheet.Range["A3"].Insert();

// Insert the range of cells
```

```
worksheet.Range["A3:A5"].Insert();
```

Refer to the following example code to see how you can insert cell range in a worksheet while specifying a direction to shift the existing cells in required direction.

C#

```
//Insert the range of cells in desired direction
worksheet.Range["A3:B10"].Insert(InsertShiftDirection.Down);
worksheet.Range["A5:C5"].Insert(InsertShiftDirection.Right);
```

Delete cell range

GcExcel allow you to delete a cell or a range of cells in the worksheets by calling **Delete ('Delete Method' in the on-line documentation)** method of **IRange ('IRange Interface' in the on-line documentation)**. To remove a cell or a range of cells, specify the cell range, for example **B4** for a single cell or **B4:C4** for a range of cells.

GcExcel provide following different options to delete a cell or range of cells.

Method	Description
Delete	This method automatically deletes a cell or the range of cells.
Delete(DeleteShiftDirection.Left)	This method deletes the range of cells and moves the existing range of cells to the left.
Delete(DeleteShiftDirection.Up)	This method delete the range of cells and move the existing range of cells in upward direction.

Refer to the following example code to see how you can delete single cell or a cell range in a worksheet.

C#

```
//Delete the range of cell
worksheet.Range["B4"].Delete();

// Delete the range of cells
worksheet.Range["B4:C4"].Delete();
```

Refer to the following example code to see how you can delete a single cell or a range of cells in a worksheet while specifying a direction to shift the existing cells in required direction.

C#

```
//Delete the range of cells from desired direction
worksheet.Range["B3:C8"].Delete(DeleteShiftDirection.Left);
worksheet.Range["B5:D5"].Delete(DeleteShiftDirection.Up);
```

Insert and Delete Rows and Columns

GcExcel provides you with the ability to insert or delete rows and columns in a worksheet.

Insert rows and columns

GcExcel allow you to add rows or columns in a worksheet by calling **Insert ('Insert Method' in the on-line documentation)** method of **IRange ('IRange Interface' in the on-line documentation)**.

When rows are added, the existing rows in the worksheet are shifted in downward direction whereas when columns are added, the existing columns in the worksheet are shifted to the right.

You can also use the **EntireRow ('EntireRow Property' in the on-line documentation)** property to insert rows in a worksheet which includes all the columns. While inserting rows using the EntireRow property, there is no need to provide the shift direction in the function parameters. If you provide the same, it will be ignored.

Refer to the following example code to insert rows in a worksheet.

C#


```
//Insert rows
worksheet.Range["A3:A5"].EntireRow.Insert();
// OR
worksheet.Range["3:5"].Insert(InsertShiftDirection.Down);
```

You can also use the **EntireColumn** ('**EntireColumn Property**' in the on-line documentation) property to insert columns in the worksheet which includes all rows. While inserting columns using the EntireColumn property, there is no need to provide the shift direction in the function parameters. If you provide the same, it will be ignored.

Refer to the following example code to insert columns in a worksheet.

```
C#
//Insert column
worksheet.Range["A3:B5"].EntireColumn.Insert();
// OR
worksheet.Range["B:C"].Insert(InsertShiftDirection.Down);
```

Delete row and column

GcExcel allows you to delete rows or columns in the worksheet by calling **Delete** ('**Delete Method**' in the on-line documentation) method of **IRange** ('**IRange Interface**' in the on-line documentation).

When rows are deleted, the existing rows in the worksheet are shifted in upwards direction, whereas when columns are deleted, the existing columns in the worksheet are shifted to the left.

Refer to the following example code to delete rows from the worksheet.

```
C#
//Delete rows
worksheet.Range["A3:A5"].EntireRow.Delete();
// OR
worksheet.Range["3:5"].Delete();
```

Refer to the following example code to delete columns from the worksheet.

```
C#
//Delete Columns
worksheet.Range["A3:A5"].EntireColumn.Delete();
// OR
worksheet.Range["A:A"].Delete(DeleteShiftDirection.Left);
```

Merge Cells

GcExcel allow you to merge several cells into a single cell using **Merge** ('**Merge Method**' in the on-line documentation) method of **IRange** ('**IRange Interface**' in the on-line documentation). When a cell range is merged, the data of top left cell stays in the final merged cell, and the data of other cells in the given range is lost.

Also if all the cells within the given range are empty, the formatting of range's top left cell is applied to the merged cell.

Refer to the following example code to merge the range of cells.

```
C#
// merge the cell range A1:B4 into one single cell
worksheet.Range["A1:B4"].Merge();
```

Refer to the following example code to merge only the rows of the specified range of cell into one.

```
C#
// merge the cell range C1:D4 by one single cell in one row
worksheet.Range["C1:D4"].Merge(true);
```

Set Values to a Range

GcExcel allows users to specify custom values for the cell range by using the properties and methods of the **IRange ('IRange Interface' in the on-line documentation)** interface.

Refer to the following example code in order to set custom values to cell ranges in the worksheet.

C#

```
worksheet.Range["A:F"].ColumnWidth = 15;
object[,] data = new object[,] {
    {"Name", "City", "Birthday", "Eye color", "Weight", "Height"},
    {"Richard", "New York", new DateTime(1968, 6, 8), "Blue", 67, 165},
    {"Nia", "New York", new DateTime(1972, 7, 3), "Brown", 62, 134},
    {"Jared", "New York", new DateTime(1964, 3, 2), "Hazel", 72, 180},
    {"Natalie", "Washington", new DateTime(1972, 8, 8), "Blue", 66, 163},
    {"Damon", "Washington", new DateTime(1986, 2, 2), "Hazel", 76, 176},
    {"Angela", "Washington", new DateTime(1993, 2, 15), "Brown", 68, 145}
};

// Set two-dimension array value to range A1:F7
worksheet.Range["A1:F7"].Value = data;

// Return a two-dimension array when get range A1:B7's value.
var result = worksheet.Range["A1:B7"].Value;
```

Set Row Height and Column Width

You can set the height of the rows and the width of the columns in a worksheet as per your preferences by using the **UseStandardHeight ('UseStandardHeight Property' in the on-line documentation)** property and **UseStandardWidth ('UseStandardWidth Property' in the on-line documentation)** property of the **IRange ('IRange Interface' in the on-line documentation)** interface respectively.

You can use the **ColumnWidth ('ColumnWidth Property' in the on-line documentation)** property to set custom width in characters for the individual columns of a range. In order to set custom width in pixels, you can use the **ColumnWidthInPixel ('ColumnWidthInPixel Property' in the on-line documentation)** property of the **IRange** interface.

You can also set custom height of the individual rows of a range in points and in pixels by using the **RowHeight ('RowHeight Property' in the on-line documentation)** property and **RowHeightInPixel ('RowHeightInPixel Property' in the on-line documentation)** property of the **IRange** interface. (**'RowHeight Property' in the on-line documentation**)

In order to specify custom total height and total width, you can use the **Height ('Height Property' in the on-line documentation)** (in points), **HeightInPixel ('HeightInPixel Property' in the on-line documentation)** (in pixels), **Width ('Width Property' in the on-line documentation)** (in characters) and **WidthInPixel ('WidthInPixel Property' in the on-line documentation)** (in pixels) properties of the **IRange** interface.

Refer to the following example code in order to customize the row height and column width in a worksheet.

C#

```
//set row height for row 1:2.
worksheet.Range["1:2"].RowHeight = 50;

//set column width for column C:D.
worksheet.Range["C:D"].ColumnWidth = 20;
```

Auto Fit Row Height and Column Width

GcExcel .NET provides support for automatic adjustment of row height and column width based on the data present in the rows and columns. The Auto Fit feature adjusts row height and column width so that every value in the rows or columns fits perfectly.

Advantage of Using Auto Fit Feature

When users need to work with spreadsheets containing huge amounts of data, some of the cells may contain values that appear cut off (if the cell width or height is too small) or contain extra spaces (if the cell width or height is too large). To avoid this anomaly and make the spreadsheets look much cleaner, GcExcel .NET enables users to automatically adjust the width of the columns and the height of the rows so as to auto fit the content inside the cell.

Further, the Auto fit feature is useful especially when you don't know how long every value is, how much space it will occupy and you also don't want to scroll through the entire spreadsheet to manually fix the row heights and column widths across the worksheet.

The following points should be kept in mind while working with the auto fit feature in GcExcel .NET:

- This feature supports the auto adjustment of column width and row height of specific cell ranges only.
- Users can use the **AutoFit()** ('AutoFit Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface in order to auto fit row height and column width.
- If the type of the cell range used is a column (**IRange.Columns** ('Columns Property' in the on-line documentation)/**IRange.EntireColumn** ('EntireColumn Property' in the on-line documentation) etc.), then only the column width will be adjusted to best fit but the row height would not be changed.

Refer to the following example code in order to automatically fit the row height and column width in a worksheet.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch the active worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Auto fit column width of range 'A1'
worksheet.Range["A1"].Value = "Grapecity Documents for Excel";
worksheet.Range["A1"].Columns.AutoFit();

// Auto fit row height of range 'B2'
worksheet.Range["B2"].Value = "Grapecity";
worksheet.Range["B2"].Font.Size = 20;
worksheet.Range["B2"].Rows.AutoFit();

// Auto fit column width and row height of range 'C3'
worksheet.Range["C3"].Value = "Grapecity Documents for Excel";
worksheet.Range["C3"].Font.Size = 32;
worksheet.Range["C3"].AutoFit();

// Saving the workbook to.xlsx
workbook.Save("AutoFitRowHeightColumnWidth.xlsx");
```



Note: The Auto fit feature has the following limitations :

- 1) In a merged cell, the AutoFit methods will not be applied. This behavior is same as in Excel.
- 2) If the text in a cell is wrapped, the Auto fit feature will not be applied to the cell.
- 3) The AutoFit methods are time-consuming and impact the performance of the spreadsheet. In order to ensure the efficiency of spreadsheet applications, users should not call the these methods too frequently.

Work with Used Range

Used Range is a bounding rectangle of used cells that returns the **IRange ('IRange Interface' in the on-line documentation)** object of used range on the specified worksheet.

GcExcel provides users with an option to work with the already used range of cells in a worksheet in the following two ways:

- **Work with worksheet's used range**
- **Work with feature related used range**

Work with worksheet's used range

To work with worksheet's used range, you need to first get the used range by using the **UsedRange property (on-line documentation)** of the **IWorksheet interface (on-line documentation)**. After you accomplish this, you can customize the used range using the properties of the **IRange interface (on-line documentation)**.

Refer to the following example code in order to get used range and customize it.

C#

```
worksheet.Range["H6:M7"].Value = 1;
worksheet.Range["J9:J10"].Merge();

//Used Range is "H6:M10"
var usedrange = worksheet.UsedRange;

//Customize the used range
usedrange.HorizontalAlignment = HorizontalAlignment.Center;
```

Work with feature related used range

To work with feature related used range, you need to first get the feature related used range by using the **GetUsedRange method (on-line documentation)** of the **IWorksheet interface**. After you accomplish this, you can customize the feature related used range using the properties of the **IRange interface**.

Refer to the following example code to get feature related used range and customize it.

C#

```
IComment commentA1 = worksheet.Range["A1"].AddComment("Range A1's comment.");
IComment commentA2 = worksheet.Range["A2"].AddComment("Range A2's comment.");

//Comment used range is "A1:D5", contains comment shape plot area
IRange commentUsedRange = worksheet.GetUsedRange(UsedRangeType.Comment);

//Customize feature related used range
commentUsedRange.Interior.Color = Color.LightYellow;
```

After you get the used range of cells using any of the above methods, you can customize it as per your preferences. For instance- you can set the row height and column width; tweak the row hidden and column hidden settings; perform certain useful operations like group and merge; add value, formula and comment to the used range in your worksheet.

Freeze Panes in a Worksheet

GcExcel provides you with the ability to freeze panes in a worksheet. This feature enables you to keep some specific rows or columns visible while users are scrolling through the rest of the sheet.

This functionality is particularly useful when there is a large amount of data that spans across a number of rows or columns.

Freeze Panes

You can freeze panes in a worksheet using the **FreezePanes() method ('FreezePanes Method' in the on-line**

documentation) of the **IWorksheet interface (on-line documentation)**. This method will freeze the split panes according to the incoming row index and column index parameters.

In order to represent the row of freeze position and the column of freeze position, you can use the **FreezeRow ('FreezeRow Property' in the on-line documentation)** and **FreezeColumn ('FreezeColumn Property' in the on-line documentation)** properties respectively.

Refer to the following example code to see how you can freeze panes in a worksheet.

C#

```
// Adding worksheets to the workbook
IWorksheet worksheet1 = workbook.Worksheets[0];
IWorksheet worksheet2 = workbook.Worksheets.Add();
IWorksheet worksheet3 = workbook.Worksheets.Add();
IWorksheet worksheet4 = workbook.Worksheets.Add();
//Freeze Panes
worksheet1.FreezePanes(2, 3);
worksheet2.FreezePanes(0, 2);
worksheet3.FreezePanes(3, 0);
worksheet4.FreezePanes(3, 5);
```

Unfreeze Panes

You can unfreeze the split panes using the **UnfreezePanes() method ('UnfreezePanes Method' in the on-line documentation)** of the **IWorksheet interface (on-line documentation)**.

Refer to the following example code to unfreeze panes in a worksheet.

C#

```
//UnFreeze Panes
worksheet4.UnfreezePanes();
```

Work with Shape And Picture

GcExcel allows users to insert and customize shapes and pictures on cells of a worksheet. You can work with shape and picture by accessing the properties and methods of the **IShape interface (on-line documentation)** and the **IShapes interface (on-line documentation)**.

This topic includes the following tasks:

1. Create different shape types
 - **Connector**
 - **Shape**
 - **Picture**
2. Customize shape format and shape text
 - **Shape format**
 - Fill
 - **Solid Fill**
 - **Gradient Fill**
 - **Pattern Fill**
 - **Picture Fill**
 - **Texture Fill**
 - **Line**
 - **3D Formatting**
 - **Shape text**

GcExcel.NET also provides support for loading and saving GrapeCity SpreadJS JSON files with shapes. For more information, refer to [Import and Export JSON Stream](#).

Connector

A connector is used when you need to connect or disconnect two general shapes. In GcExcel, you can use the **BeginConnect method (on-line documentation)**, **EndConnect method (on-line documentation)**, **BeginDisconnect method (on-line documentation)** and **EndDisconnect method (on-line documentation)** of the **IConnectorFormat interface (on-line documentation)** to attach and detach the ends of the connector to other shapes.

Refer to the following example code to connect general shapes using the connector format.

C#

```
// To config the connector shape.
IShape shapeBegin = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 1, 1, 100, 100);
IShape endBegin = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 200, 200, 100, 100);
IShape connectorShape = worksheet.Shapes.AddConnector(ConnectorType.Straight, 1, 1, 101, 101);
connectorShape.Width = 10;
// To detach the ends of the connector to other shapes.
connectorShape.ConnectorFormat.BeginConnect(shapeBegin, 3);
connectorShape.ConnectorFormat.EndConnect(endBegin, 0);
```



Note: One of the limitations of using connector format is that you can add a connector to connect two general shapes and export it but the connector will be shown only after you drag the shape to your spreadsheet.

Shape

A shape is a drawing object and a member of the **Shapes** collection. In GcExcel, the Shapes collection represents the collection of shapes in a specified worksheet. All the drawing objects including chart, comment, picture, slicer, general shape and shape group are defined as Shape.

Picture

You can insert pictures on cells of a spreadsheet by using the **AddPicture method (on-line documentation)** of the IShapes interface. The **IPictureFormat interface (on-line documentation)** in GcExcel allows users to customize and format pictures while working in a spreadsheet.

Refer to the following example code when working with picture in GcExcel:

C#

```
// Add a picture
IShape picture = worksheet.Shapes.AddPicture(@"Images\flower.jpg", 480, 10, 100, 100);
// Fill the inserted picture
picture.Fill.Solid();
picture.Fill.Color.RGB = Color.AliceBlue;
//Customize the inserted picture
picture.PictureFormat.Crop.PictureWidth = 80;
```

Shape Format

In GcExcel, you can customize the shape format in three different ways. This includes setting the fill format for the inserted shape using the properties and methods of the **IFillFormat interface (on-line documentation)**, configuring the shape's line using the properties and methods of the **ILineFormat interface (on-line documentation)** and applying 3D formatting to the shape using the properties and methods of the **IThreeDFormat interface (on-line documentation)**.

Solid Fill

To format the shape with Solid fill, first you need to use the **Solid method (on-line documentation)** of the IFillFormat interface to specify the fill format and then set the **Color property ('Color Property' in the on-line documentation)** and **Transparency property (on-line documentation)** to set the shape's fill color and transparency degree respectively.

Refer to the following example code to fill the shape with solid fill.

C#

```
//Solid Fill
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Balloon, 10, 10, 100, 100);
IColorFormat color = shape.Fill.Color;
color.RGB = Color.Red;
shape.Fill.Solid();
```

Gradient Fill

In gradient fill, you first need to set the shape fill to the gradient fill using the **OneColorGradient method (on-line documentation)**, **TwoColorGradient method (on-line documentation)** or **PresetGradient method (on-line documentation)** of the IFillFormat interface. When you're done, you can then insert, delete or modify gradient stops; set the fill style rotation along with the shape and the angle of the gradient fill using the **GradientStops property (on-line documentation)**, **RotateWithObject property (on-line documentation)** and **GradientAngle property (on-line documentation)** of the IFillFormat interface.

Refer to the following example code to fill the shape with gradient fill.

C#

```
//Gradient Fill
IShape shape1 = worksheet.Shapes.AddShape(AutoShapeType.Heart, 120, 10, 100, 100);
shape1.Fill.PresetGradient(GradientStyle.Vertical, 3, PresetGradientType.Silver);
shape1.Fill.RotateWithObject = false;
```

Pattern Fill

In pattern fill, you first need to set the shape fill to pattern fill using the **Patterned method (on-line documentation)** of the IFillFormat interface. Afterwards, you can set the background color and the pattern color using **Color property (on-line documentation)** and **PatternColor property (on-line documentation)** of the IFillFormat interface.

Refer to the following example code to fill the shape with pattern fill.

C#

```
//Pattern Fill
IShape shape2 = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 240, 10, 100, 100);
shape2.Fill.Patterned(GrapeCity.Documents.Excel.Drawing.PatternType.Percent10);
shape2.Fill.Color.ObjectThemeColor = ThemeColor.Accent2;
shape2.Fill.PatternColor.ObjectThemeColor = ThemeColor.Accent6;
```

Picture Fill

In picture fill, you can use the **AddShape method (on-line documentation)** of the IShapes interface to first add the shape that you want to fill with a picture. Further, you can also set the picture format including characteristics like picture height, picture width, brightness, contrast ratio, re-coloring, x-axis and y-axis offset etc using the properties of the IPictureFormat interface.

Refer to the following example code to fill the shape with picture.

C#

```
// Add shape of picture type
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 20, 20, 100, 100);
string path = @"Images\flower.jpg";
FileStream stream = System.IO.File.Open(path, FileMode.Open);
shape.Fill.UserPicture(stream, ImageType.JPG);
stream.Dispose();
// Recolor the picture
shape.PictureFormat.ColorType = PictureColorType.Grayscale;
// Set picture's brightness and contrast ratio.
shape.PictureFormat.Brightness = 0.6;
```

```
shape.PictureFormat.Contrast = 0.3;
// Set height, width, x-axis offset and y-axis offset of the specified picture.
shape.PictureFormat.Crop.PictureOffsetX = 10;
shape.PictureFormat.Crop.PictureOffsetY = -5;
shape.PictureFormat.Crop.PictureWidth = 120;
shape.PictureFormat.Crop.PictureHeight = 80;
```

Texture Fill

In texture fill, you can fill the shape with texture using the **PresetTextured method (on-line documentation)**, or **UserTextured method (on-line documentation)** of the IFillFormat interface. Further, you can also use the **TextureAlignment property (on-line documentation)**, **TextureHorizontalScale property (on-line documentation)**, **TextureOffsetX property (on-line documentation)**, **TextureOffsetY property (on-line documentation)** and **TextureVerticalScale property (on-line documentation)** to configure the layout of the texture.

Refer to the following example code to fill the shape with texture fill.

C#

```
//Texture Fill
IShape shape3 = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 360, 10, 100, 100);
shape3.Fill.PresetTextured(PresetTexture.Canvas);
shape3.Fill.TextureAlignment = TextureAlignment.Center;
shape3.Fill.TextureOffsetX = 2.5;
shape3.Fill.TextureOffsetY = 3.2;
shape3.Fill.TextureHorizontalScale = 0.9;
shape3.Fill.TextureVerticalScale = 0.2;
shape3.Fill.Transparency = 0.5;
```

Line

Line is a kind of border around the shape. You can create lines around shapes inserted on cells of a spreadsheet using the properties and methods of ILineFormat interface.

Refer to the following example code to configure the line and line style for the shape.

C#

```
// To set shape's line style.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 10, 10, 100, 100);
shape.Line.DashStyle = LineDashStyle.Dash;
shape.Line.Style = LineStyle.Single;
shape.Line.Weight = 2;
shape.Line.Color.ObjectThemeColor = ThemeColor.Accent6;
shape.Line.Transparency = 0.3;
```



Shape's Line also supports solid fill, gradient fill and pattern fill and its usage is similar to the Shape Fill.

3D Formatting

GcExcel allows you to format the three-dimensional layout for the inserted shape by setting its rotation degree around x,y and z axis.

Refer to the following example code to apply 3D formatting to the embedded shape.

C#

```
// To set shape's rotation degree around x, y, z axis.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 50, 10, 100, 100);
shape.ThreeD.RotationX = 50;
shape.ThreeD.RotationY = 20;
shape.ThreeD.RotationZ = 30;
```



```
shape.ThreeD.Depth = 7;  
shape.ThreeD.Z = 20;
```

Shape Text

In GcExcel, you can configure the text and text style for the shape as per your own preferences by using the **TextFrame property (on-line documentation)** of the IShape interface.

Refer to the following example code to configure the text and text style for the inserted shape.

C#

```
// To config shape's text and text style.  
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 40, 40, 100, 100);  
shape.TextFrame.TextRange.Font.Color.RGB = System.Drawing.Color.FromArgb(0, 255, 0);  
shape.TextFrame.TextRange.Font.Bold = true;  
shape.TextFrame.TextRange.Font.Italic = true;  
shape.TextFrame.TextRange.Font.Size = 20;  
shape.TextFrame.TextRange.Font.Strikethrough = true;  
  
shape.TextFrame.TextRange.Paragraphs.Add("This is a rectangle shape.");  
shape.TextFrame.TextRange.Paragraphs.Add("My name is GcExcel.");  
shape.TextFrame.TextRange.Paragraphs[1].Runs.Add("Hello World!");  
  
shape.TextFrame.TextRange.Paragraphs[1].Runs[0].Font.Strikethrough = false;  
shape.TextFrame.TextRange.Paragraphs[1].Runs[0].Font.Size = 35;
```

Customize Worksheets

GcExcel allows you to customize worksheets using the properties of **IWorksheet Interface (on-line documentation)**. You can perform useful operations like customizing gridlines to modify row and column headers, setting color for the tabs, or setting default height and width for rows and columns, and so much more.

Customizing a worksheet to modify the default settings involves the following tasks:

- **Configure display**
- **Set the tab color**
- **Set visibility**
- **Set background image**
- **Define standard height and width**

Configure display

You can modify the display settings of your worksheet from left to right or right to left.

Refer to the following example code to configure the display of your worksheet in GcExcel.

C#

```
// Fetch the default WorkSheet  
IWorksheet worksheet = workbook.Worksheets[0];  
  
// Assign the values to the cells  
worksheet.Range["B1"].Value = "ABCD";  
worksheet.Range["B2"].Value = 3;  
worksheet.Range["C1"].Value = "GrapeCity Documents";  
worksheet.Range["C2"].Value = 4;  
worksheet.Range["D1"].Value = "GcExcel";  
worksheet.Range["D2"].Value = "ABCD";  
  
// Set the specified sheet to be displayed from left to right.  
worksheet.SheetView.DisplayRightToLeft = true;
```

Set the tab color

You can change the default tab color of your worksheet using the **TabColor property (on-line documentation)** of the **IWorksheet interface (on-line documentation)**.

Refer to the following example code to set the tab color for your worksheet.

C#

```
// Set the tab color of the specified sheet as green.
worksheet.TabColor = Color.Green;
```

Set visibility

You can show or hide your worksheet using the **Visible property (on-line documentation)** of the **IWorksheet interface**.

Refer to the following example code to set visibility of your worksheet.

C#

```
// Adding new sheet and set the visibility of the sheet as Hidden.
IWorksheet worksheet1 = workbook.Worksheets.Add();
worksheet1.Visible = Visibility.Hidden;
```

Set background image

You can set a custom background image to your worksheet using the **BackgroundPicture ('BackgroundPicture Property' in the on-line documentation)** property of the **IWorksheet ('IWorksheet Interface' in the on-line documentation)** interface. With this feature, users can insert any background image to the worksheet including their organization logo, custom watermark or a wallpaper of their choice without any hassles.

Refer to the following example code in order to set the custom background image in your worksheet.

C#

```
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Set Background Image
worksheet.BackgroundPicture = File.ReadAllBytes(@"GrapeCityLogo.png");
```

Define standard width and height

You can define the standard height and width of your worksheet using the **StandardHeight ('StandardHeight Property' in the on-line documentation)** and **StandardWidth ('StandardWidth Property' in the on-line documentation)** properties of the **IWorksheet interface**, respectively.

Refer to the following example code to define the standard width and height as per your requirements.

C#

```
// Setting the height and width of the worksheet
worksheet.StandardHeight = 20;
worksheet.StandardWidth = 40;
```

Work with Worksheet Views

GcExcel offers customization of several display settings that are applied to a worksheet.

In order to view a worksheet as per their own preferences, users can use the properties and methods of the **IWorksheet ('IWorksheet Interface' in the on-line documentation)** interface, **IPane ('IPane Interface' in the on-line documentation)** interface and **IWorksheetView ('IWorksheetView Interface' in the on-line documentation)** interface.

The following table describes some of the properties and methods that can be used to customize the view settings while working with worksheets.

Property/Method	Description
IWorksheet.SplitPanes(int row, int column) ('SplitPanes Method' in the on-line documentation)	This method can be used to lock the rows and columns in a worksheet in order to divide the worksheet into multiple areas that can be scrolled independently. Users need to provide the cell index as parameters in this method to specify the location where they want the split.
IWorksheet.UnsplitPanes(int row, int column) ('UnsplitPanes Method' in the on-line documentation)	This method can be used to unsplit the split panes. Using this method is similar to using IWorksheet.SplitPanes(0,0).
IWorksheet.SplitRow ('SplitRow Property' in the on-line documentation) / IWorksheet.SplitColumn ('SplitColumn Property' in the on-line documentation)	This method gets the split distances (row count and column count) from top (in case of row) or left (in case of column).
IWorksheet.Panes ('Panes Property' in the on-line documentation)	A range object that represents the frozen or split panes of the worksheet.
IWorksheet.ActivePane ('ActivePane Property' in the on-line documentation)	This property can be used to get the active pane in a worksheet.
IPane.Activate() ('Activate Method' in the on-line documentation)	This method activates the current pane.
IPane.Index ('Index Property' in the on-line documentation)	This property can be used to get the index of the current pane in IWorksheet.Panes.
IPane.ScrollColumn ('ScrollColumn Property' in the on-line documentation) / IPane.ScrollRow ('ScrollRow Property' in the on-line documentation)	This property can be used to get or set the top left cell position of the current pane.
IWorksheet.SheetView ('SheetView Property' in the on-line documentation)	This property can be used to get the view of the worksheet.
IWorksheetView.Zoom ('Zoom Property' in the on-line documentation)	This property can be used to get and set a variant numeric value that represents the display size of the worksheet as a percentage where the 100 equals normal size, 200 equals double size, and so on.
IWorksheetView.GridlineColor ('GridlineColor Property' in the on-line documentation)	This property can be used to get and set the gridline color.
IWorksheetView.ScrollColumn ('ScrollColumn Property' in the on-line documentation)	This property can be used to get and set the number of the leftmost column in the worksheet.
IWorksheetView.ScrollRow ('ScrollRow Property' in the on-line documentation)	This property can be used to get and set the number of the row that appears at the top of the worksheet.
IWorksheetView.DisplayRightToLeft ('DisplayRightToLeft Property' in the on-line documentation)	This property can be used to get and set whether the specified worksheet is displayed from right to left instead of from left to right.
IWorksheetView.DisplayFormulas ('DisplayFormulas Property' in the on-line documentation)	This property can be used to get and set whether the worksheet displays formulas.

on-line documentation)**IWorksheetView.DisplayGridlines**
(**'DisplayGridlines Property' in the on-line documentation**)

This property can be used to get and set whether the gridlines are displayed.

IWorksheetView.DisplayHeadings
(**'DisplayHeadings Property' in the on-line documentation**)

This property can be used to get and set whether the headers are displayed.

IWorksheetView.DisplayOutline
(**'DisplayOutline Property' in the on-line documentation**)

This property can be used to get and set whether the outline symbols are displayed.

IWorksheetView.DisplayRuler
(**'DisplayRuler Property' in the on-line documentation**)

This property can be used to get and set whether a ruler is displayed for the specified worksheet.

IWorksheetView.DisplayWhitespace
(**'DisplayWhitespace Property' in the on-line documentation**)

This property can be used to get and set whether the whitespace is displayed.

IWorksheetView.DisplayZeros
(**'DisplayZeros Property' in the on-line documentation**)

This property can be used to get and set whether the zero values are displayed.

The following code snippet shows how to set custom view for a worksheet using different properties of the IWorksheet interface.

C#

```
//Set worksheet view

IWorkbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
var custom_view = worksheet.SheetView;
custom_view.Zoom = 200;
custom_view.GridlineColor = Color.Red;
custom_view.ScrollColumn = 10;
var scrollRow = custom_view.ScrollRow;
```

The following code snippet shows how to use the SplitPanes() method to split the worksheet into panes.

C#

```
//Split worksheet using SplitPanes() method

Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.SplitPanes(worksheet.Range["A5"].Row, worksheet.Range["A5"].Column);

var splitRow = worksheet.SplitRow;
var splitColumn = worksheet.SplitColumn;
```

Allow Sort

GcExcel provides the **Sort ('Sort Method' in the on-line documentation)** method to perform data sorting based on a range of cells, range by value, color or icon in a worksheet. The **Apply ('Apply Method' in the on-line documentation)** method is used to apply the selected sort state and display the results.



Note: Sorting can be performed on merged cells as well, provided merged cells have the same size.

Following are the types of sorting available in GcExcel.

Sort by value

Sort by value performs sorting to arrange the data in order. **SortOrientation** property is used to specify the orientation category for sorting, that is, columns or rows.

Refer to the following code example to sort by value.

C#

```
//Sort by value, use Sort() method.
worksheet.Range["A1:B4"].Sort(worksheet.Range["A1:A4"], orientation:
SortOrientation.Columns);
```

Sort by value for multiple columns

Sort by value for multiple columns performs sorting on multiple columns using a single line of code.

ValueSortField method is used to define multiple sort field instances in one statement. **SortOrder** property is used to specify the orientation of columns in either ascending order or descending order.

Refer to the following code example to sort by value for multiple columns.

C#

```
//Sort by value, multi column sort.use Sort() method.
worksheet.Range["A1:B4"].Sort(SortOrientation.Columns, false, new ValueSortField[] {
new ValueSortField(worksheet.Range["A1:A4"],SortOrder.Descending), new
ValueSortField(worksheet.Range["B1:B4"], SortOrder.Ascending)});
```

Custom sort

Sorting is a common task, but not all data conforms to the common ascending and descending rule. For example, months cannot be sorted in a meaningful way when sorted alphabetically. In this case, GcExcel offers a custom sort. For custom sorting, string of values are defined in **ValueSortField** constructor.

Refer to the following code example to implement custom sorting.

C#

```
//give a custom sort values string.
var sortkey = new ValueSortField(worksheet.Range["A1:A2"], "1,2,3");
worksheet.Range["A2:A6"].Sort(SortOrientation.Columns, false, sortkey);
```

Sort by interior

Sort by interior performs sorting on the basis of interior color, pattern, pattern color, gradient color and gradient angle. However, interior sort cannot be performed on the basis of cell color.

Refer to the following code example to sort by interior.

C#

```
// Assigning pattern to the range
worksheet.Range["A3"].Interior.Pattern = Pattern.LinearGradient;
worksheet.Range["A4"].Interior.Pattern = Pattern.LinearGradient;
worksheet.Range["A5"].Interior.Pattern = Pattern.LinearGradient;
worksheet.Range["A6"].Interior.Pattern = Pattern.LinearGradient;
// Defining values to the range
worksheet.Range["A3"].Value = 1;
worksheet.Range["A4"].Value = 2;
worksheet.Range["A5"].Value = 3;
worksheet.Range["A6"].Value = 4;
// Assigning gradient to the range
( worksheet.Range["A3"].Interior.Gradient as ILinearGradient).ColorStops[0].Color =
Color.FromArgb(255, 0, 0);
( worksheet.Range["A3"].Interior.Gradient as ILinearGradient).ColorStops[1].Color =
```

```

Color.FromArgb(146, 208, 80);
( worksheet.Range["A3"].Interior.Gradient as ILinearGradient).Degree = 90;

( worksheet.Range["A4"].Interior.Gradient as ILinearGradient).ColorStops[0].Color =
Color.FromArgb(255, 0, 255);
( worksheet.Range["A4"].Interior.Gradient as ILinearGradient).ColorStops[1].Color =
Color.FromArgb(146, 208, 90);
( worksheet.Range["A4"].Interior.Gradient as ILinearGradient).Degree = 90;

( worksheet.Range["A5"].Interior.Gradient as ILinearGradient).ColorStops[0].Color =
Color.FromArgb(255, 0, 255);
( worksheet.Range["A5"].Interior.Gradient as ILinearGradient).ColorStops[1].Color =
Color.FromArgb(146, 208, 180);
( worksheet.Range["A5"].Interior.Gradient as ILinearGradient).Degree = 90;

( worksheet.Range["A6"].Interior.Gradient as ILinearGradient).ColorStops[0].Color =
Color.FromArgb(255, 0, 255);
( worksheet.Range["A6"].Interior.Gradient as ILinearGradient).ColorStops[1].Color =
Color.FromArgb(146, 208, 90);
( worksheet.Range["A6"].Interior.Gradient as ILinearGradient).Degree = 90;
//
worksheet.Sort.SortFields.Add(new CellColorSortField(worksheet.Range["A1:A2"],
worksheet.Range["A6"].DisplayFormat.Interior, SortOrder.Ascending));
worksheet.Sort.Range = worksheet.Range["A3:A6"];
worksheet.Sort.Orientation = SortOrientation.Columns;
worksheet.Sort.Apply();

```

Sort by font color

Sort by font color performs sorting by cell's display format font color. However, sorting is not performed on the basis of cell color.

Refer to the following code example to sort by font color.

C#

```

// Assigning Value to the range
worksheet.Range["A1"].Value = 2;
worksheet.Range["A2"].Value = 1;
worksheet.Range["A3"].Value = 1;
worksheet.Range["A4"].Value = 3;

worksheet.Range["B1"].Value = 2;
worksheet.Range["B2"].Value = 1;
worksheet.Range["B3"].Value = 1;
worksheet.Range["B4"].Value = 3;
// Assigning Color to the range
worksheet.Range["B1"].Font.Color = Color.FromArgb(0, 128, 0);
worksheet.Range["B2"].Font.Color = Color.FromArgb(128, 0, 0);
worksheet.Range["B3"].Font.Color = Color.FromArgb(0, 0, 128);
worksheet.Range["B4"].Font.Color = Color.FromArgb(128, 128, 0);
// Defining Sort by Color
worksheet.Sort.SortFields.Add(new FontColorSortField(worksheet.Range["B1:B4"],
worksheet.Range["B1"].DisplayFormat.Font.Color, SortOrder.Descending));
worksheet.Sort.Range = worksheet.Range["A1:B4"];
worksheet.Sort.Orientation = SortOrientation.Columns;
worksheet.Sort.Apply();

```

Sort by Icon

Sort by icon performs sorting on the basis of cell's conditional format icons.

Refer to the following code example to sort by icon.

C#

```
// Assigning Value to the range
worksheet.Range["A1"].Value = 2;
worksheet.Range["A2"].Value = 1;
worksheet.Range["A3"].Value = 1;
worksheet.Range["A4"].Value = 3;

worksheet.Range["B1"].Value = 2;
worksheet.Range["B2"].Value = 1;
worksheet.Range["B3"].Value = 1;
worksheet.Range["B4"].Value = 3;
// Defining Sort by Icon
IIconSetCondition iconset =
worksheet.Range["B1:B4"].FormatConditions.AddIconSetCondition();
iconset.IconSet = workbook.IconSets[IconSetType.Icon3TrafficLights1];

worksheet.Sort.SortFields.Add(new IconSortField(worksheet.Range["B1:B4"],
workbook.IconSets[IconSetType.Icon3TrafficLights1][0], SortOrder.Descending));
worksheet.Sort.Range = worksheet.Range["A1:B4"];
worksheet.Sort.Orientation = SortOrientation.Columns;
worksheet.Sort.Apply();
```

Apply Filters

Worksheets with bulk data can be difficult to manage. In such a scenario, applying filters can be a useful feature to view only the required information while hiding rest of the data. Filters are used to display only the relevant records that match to a certain criteria in a particular column.

In GcExcel, you can apply filters to a selected range of data. For example, you can apply date type filter from C4 to C7 range. To filter data in a range of cells or a table, you need to set the auto filter mode for the worksheet to boolean true or false using **AutoFilterMode property (on-line documentation)** of the **IWorksheet interface (on-line documentation)**.

There are several types of range filters responsible for executing distinct filter operations in a worksheet.

- **Apply number filters**
- **Apply multi select filters**
- **Apply text filters**
- **Apply date filters**
- **Apply dynamic date filters**
- **Apply filters by cell color**
- **Apply filters by no fill**
- **Apply filters by icon**
- **Apply filters by no icon**

Apply number filters

Refer to the following example code to see how you can apply number filters to display data that meets the specified criteria applied on a column containing numeric cell values.

C#

```
// Apply number filter
worksheet.Range["D3:I6"].AutoFilter(0, "<>2");
```

Apply multi select filters

Refer to the following example code to see how multi select filters can be applied to quickly filter data based on cell values with multiple selections.

C#

```
//filter condition is "multi select".
worksheet.Range["A1:E5"].AutoFilter(0, new object[] { "$2", "$4" },
AutoFilterOperator.Values);
```

Apply text filters

Refer to the following example code to see how text filters are applied to display rows with cell values that either match to the specified text or regular expression value in the column on which the filter is applied.

C#

```
//begin with "a".
worksheet.Range["D3:I9"].AutoFilter(1, "a*");
```

Apply date filters

Refer to the following example code to see how date filters can be applied to a range to display only those results that are falling within the specified dates.

Apply date filters

```
//Apply filter using Date criteria
var criterial = new DateTime(2008, 1, 1).ToString();
var criteria2 = new DateTime(2008, 8, 1).ToString();
worksheet.Range["D20:F29"].AutoFilter(2, ">=" + criterial, AutoFilterOperator.And, "<="
+ criteria2);
```

Apply dynamic date filters

Refer to the following example code to see how dynamic date filters can be applied to display results that match the specified date criteria taking into account the current system date that automatically gets updated everyday.

C#

```
//filter in yesterday.
worksheet.Range["D7:F18"].AutoFilter(2, DynamicFilterType.Yesterday,
AutoFilterOperator.Dynamic);
```

Apply filters by cell colors

Refer to the following example code to see how you can apply filters by cell colors on a column to display results containing cells with distinct fill shades.

C#

```
worksheet.Range["A1:A6"].AutoFilter(0, Color.FromArgb(255, 255, 0),
AutoFilterOperator.CellColor);
```

Apply filters by no fill

Refer to the following example code to see how you can apply filters by no fill on a column to display results containing cells with no fill color.

C#

```
worksheet.Range["A1:A6"].AutoFilter(0, null, AutoFilterOperator.NoFill);
```

Apply filters by icon

Refer to the following example code to see how you can apply filters by icon to display results that contain a specific icon in the cells.

C#


```
worksheet.Range["A1:A10"].AutoFilter(0, workbook.IconSets[IconSetType.Icon5ArrowsGray][0], AutoFilterOperator.Icon);
```

Apply filters by no icon

Refer to the following example code to see how you can apply filters by no icon to display results where cells do not possess an icon.

C#

```
worksheet.Range["A1:A10"].AutoFilter(0, null, AutoFilterOperator.NoIcon);
```

Manage Hyperlinks

In GcExcel, hyperlinks can be created and inserted in cells to allow users to quickly access related information present in another file or on a webpage by clicking on the link.

Hyperlinks are stored in a specific worksheet or in a range by accessing the **Hyperlinks** collection of the **IWorksheet interface (on-line documentation)** and the **IRange interface (on-line documentation)** respectively.

In GcExcel, you can perform the following tasks to manage hyperlinks.

- **Add Hyperlinks**
- **Configure Hyperlinks**
- **Delete Hyperlinks**

Add hyperlinks

Hyperlinks can be created and inserted through linking to an external file, linking to a webpage, linking to an email address and also linking to a range within the worksheet. You can add hyperlinks for a range of cells in a worksheet using the **Add method (on-line documentation)** of the **IHyperLinks interface (on-line documentation)**.

Refer to the following example code to insert hyperlinks to an external file, to a webpage, to a range within the worksheet and to an email address.

C#

```
// Add a hyperlink link to external file
worksheet.Range["A1:B2"].Hyperlinks.Add(worksheet.Range["A1"],
@"C:\Documents\GcExcel\GrapeCityDocumentsExcel\Project\Hyperlink\SampleFile.xlsx",
null,
"link to SampleFile.xlsx file.",
"SampleFile.xlsx");
```

C#

```
// Add a hyperlink link to web page
worksheet.Range["A1:B2"].Hyperlinks.Add(worksheet.Range["A1"],
"http://www.grapecity.com/",
null,
"open Grapecity web site.",
"Grapecity");
```

C#

```
//Add a hyperlink link to a range in this document.
worksheet.Range["A1:B2"].Hyperlinks.Add(worksheet.Range["A1"],
null,
"Sheet1!$C$3:$E$4",
"Go To sheet1 C3:E4");
```

C#

```
//Add a hyperlink link to email address.
worksheet.Range["A1:B2"].Hyperlinks.Add(worksheet.Range["A1"],
                                         "mailto:abc.xyz@grapecity.com",
                                         null,
                                         "Send an email to ABC",
                                         "Send To ABC");
```

Configure Hyperlinks

Hyperlinks can be configured using the following properties of the **IHyperlink interface (on-line documentation)**.

1. You can use the Address and SubAddress properties of the IHyperlink interface to configure the hyperlink references. The table shown below illustrates both of these properties with examples:

Link To	Address	SubAddress
External File	Example: "C:\Users\Desktop\test.xlsx"	null
Webpage	Example: "http://www.grapecity.com/"	null
A range in this document	Example: null	"Sheet1!\$C\$3:\$E\$4"
Email Address	Example: "mailto: abc.xyz@grapecity.com"	null

2. You can use the EmailSubject property to set the text of hyperlink's email subject line.
3. You can use the ScreenTip property to set the tip text for the specified hyperlink.
4. You can use the TextToDisplay property to set the text to be displayed for the specified hyperlink.

Delete Hyperlinks

The hyperlinks inserted in the cells can be removed from the hyperlinks collection in a specific worksheet or in a specific range using the Delete method.

Refer to the following example code to delete hyperlinks.

C#

```
//Delete hyperlinks.
worksheet.Range["A1:B2"].Hyperlinks.Add(worksheet.Range["A1:A2"],
                                         null,
                                         "Sheet1!$C$3:$E$4",
                                         "Go To sheet1 C3:E4");

worksheet.Range["H5"].Hyperlinks.Add(worksheet.Range["A1"],
                                         "http://www.grapecity.com/");
worksheet.Range["J6"].Hyperlinks.Add(worksheet.Range["A1"],
                                         "http://www.grapecity.com/");

//delete hyperlinks in range A1:A2.
worksheet.Range["A1:A2"].Hyperlinks.Delete();

//delete all hyperlinks in this worksheet.
worksheet.Hyperlinks.Delete();
```

Apply Grouping

You can group rows or columns of data to organize information and create custom views in a spreadsheet.

Each group in GcExcel .NET is distinguished with a group header row with collapse and expand icons next to it that can be used for displaying or hiding information as and when required. You can set the **Show Detail property**

('ShowDetail Property' in the on-line documentation) of the **IRange Interface (on-line documentation)** to boolean true to expand a group to display rows and columns that have been hidden and false to collapse the expanded rows or columns.

Applying grouping in a spreadsheet involves the following tasks:

- [Create Row or Column Group](#)
- [Remove a Group](#)
- [Set Summary Row](#)



Note : When grouping is applied, rows of data are automatically sorted in ascending order against the grouped columns.

Create Row or Column Group

You can apply grouping on rows and columns of a spreadsheet.

- **Apply row grouping**
- **Apply column grouping**
- **Set outline level for rows and columns**

Apply row grouping

You can apply row grouping by using the **Group method (on-line documentation)** of the **IRange interface (on-line documentation)** and specifying the rows you want to apply grouping on.

Refer to the following example code to apply row grouping in a worksheet.

C#

```
//1:20 rows' outline level will be 2.  
worksheet.Range["1:20"].Group();
```

Apply column grouping

You can apply column grouping by using the Group method of the IRange interface and specifying the columns you want to apply grouping on.

Refer to the following example code to apply column grouping in a worksheet.

C#

```
//A:I columns' outline level will be 2.  
worksheet.Range["A:I"].Group();
```

Set outline level for rows and columns

When the data is grouped for the first time, it displays only the rows arranged into the first level group on the basis of the values of the cells in that particular column. After the first-level grouping, when the view is grouped by any column other than the one used previously, the rows will be arranged in the second level group, third level group and so on.

In case you want to set the specific outline level for grouping of rows or columns, you can use the **OutlineLevel property (on-line documentation)** of the IRange interface. You can also choose to display specified levels of row or column groups using the **ShowLevels method (on-line documentation)** of the **IOutline interface (on-line documentation)**.

Refer to the following example code to set the Outline level for rows and columns.

C#

```
//1:10 rows' outline level will be 4.  
worksheet.Range["1:10"].Group();  
worksheet.Range["1:10"].OutlineLevel = 4;  
  
//A:E columns' outline level will be 4.
```

```
worksheet.Range["A:E"].Group();  
worksheet.Range["A:E"].OutlineLevel = 4;
```

You can use **SummaryColumn property (on-line documentation)** or **SummaryRow property (on-line documentation)** of the IOutline interface to set whether summary column is in left or right of column groups or summary row is above or below the row groups, respectively.

Remove a Group

You can remove a group by implementing the following tasks in your worksheet.

- **Ungroup rows and columns**
- **Clear Outline**
- **Collapse a Group**

Ungroup rows and columns

The grouped rows or columns can be ungrouped if you no longer want the information to be organized in clusters. You can increment or decrement the outline level for the specified rows or columns using the **Group method (on-line documentation)** and **Ungroup method (on-line documentation)** of the **IRange interface (on-line documentation)** respectively.

Refer to the following example code to ungroup row and column in a worksheet.

C#

```
// Row Ungrouping  
//1:5 rows' outline level will be 1.  
worksheet.Range["1:5"].Ungroup();  
  
// Column Ungrouping  
//A:I columns outline level will be 2.  
worksheet.Range["A:I"].Group();  
//A:D columns outline level will be 1.  
worksheet.Range["A:D"].Ungroup();
```

Clear outline

You can clear the outline level of the specified rows or columns using the **ClearOutline method (on-line documentation)** of the IRange interface.

Refer to the following example code to clear outline in a worksheet.

C#

```
//1:20 rows' outline level will be 2.  
worksheet.Range["1:20"].Group();  
//1:10 rows' outline level will be 3.  
worksheet.Range["1:10"].Group();  
  
//ClearOutline  
//12:20 rows' outline level will be 1.  
worksheet.Range["12:20"].ClearOutline();
```

Collapse a group

You can collapse a group by setting the **ShowDetail property (on-line documentation)** of the IRange interface to boolean false.

Refer to the following example code to collapse a group in a worksheet.

C#

```
//1:20 rows' outline level will be 2.
worksheet.Range["1:20"].Group();
//1:10 rows' outline level will be 3.
worksheet.Range["1:10"].Group();
//collapse
//1:10 rows will be collapsed.
worksheet.Range["11:11"].ShowDetail = false;
```

Set Summary Row

When grouping is performed in a spreadsheet, a summary row is automatically created corresponding to each group. Summary rows are group header rows that display the group name with the information about the group that is being created.

While working with GcExcel .NET, you modify and customize the summary row as per the requirement using the **SummaryRow property (on-line documentation)** of the **IOutline interface (on-line documentation)**.

Refer to the following example code to set summary row.

```
C#
//summary
worksheet.Outline.SummaryRow = SummaryRow.Above;

//Summary row will be row 4.
worksheet.Range["5:20"].Group();
//Summary row will be row 14.
worksheet.Range["15:20"].Group();
```

Apply Style

In GcExcel .NET, a workbook possesses a set of styles that can be utilized to format cell appearance in individual worksheets for enhanced clarity and increased readability.

Applying style in a worksheet involves following tasks.

- [Set Sheet Styling](#)
- [Create and Set Custom Named Style](#)

Some of the built-in styles in GcExcel .NET are listed below:

Category	Description	Properties
Number Format	Cell number format.	IRange.NumberFormat ('NumberFormat Property' in the on-line documentation)
Alignment	Horizontal and vertical alignment of cell content, indentation, text wrap, text rotation and text shrinking.	IRange.AddIndent ('AddIndent Property' in the on-line documentation) IRange.IndentLevel ('IndentLevel Property' in the on-line documentation) IRange.WrapText ('WrapText Property' in the on-line documentation) IRange.ShrinkToFit ('ShrinkToFit Property' in the on-line documentation) IRange.MergeCells

		('MergeCells Property' in the on-line documentation) IRange.ReadingOrder ('ReadingOrder Property' in the on-line documentation) IRange.Orientation ('Orientation Property' in the on-line documentation)
Font	IRange.Font(IFont)	IRange.Font ('Font Property' in the on-line documentation)(IFont)
Borders	Cell border line styles and colors.	IRange.Borders ('Borders Property' in the on-line documentation)(IBorders)
Fill	Cell pattern fill or gradient fill.	IRange.Interior ('Interior Property' in the on-line documentation)(IInterior)
Protection	Cell protection options (Locked and Hidden)	IRange.Locked ('Locked Property' in the on-line documentation) IRange.FormulaHidden ('FormulaHidden Property' in the on-line documentation)

Apart from the built-in styles, you can also create custom styles with description for individual cells or a range of cells in a worksheet where you can define all the style attributes and properties including font, font size, number format, alignment etc.

Set Sheet Styling

You can apply styling to your worksheets by performing actions like setting different fill styles for a cell, customizing the cell border and configuring the fonts for the spreadsheets etc.

- **Set fill**
 - **Solid fill**
 - **Pattern fill**
 - **Gradient fill**
 - **Linear gradient fill**
 - **Rectangular gradient fill**
- **Set font**
- **Set border**
- **Set number format**
- **Set alignment**
- **Set protection**

Set fill

You can set the fill style for a cell by using the **Interior property (on-line documentation)** of the **IRange interface (on-line documentation)**. A cell interior can be of three types, namely, solid fill, pattern fill and gradient fill.

Solid fill

You can specify the fill style for the cell as solid by setting the **Pattern property (on-line documentation)** of the **IInterior interface (on-line documentation)**.

Refer to the following example code to set solid fill.

C#

```
// Solid Fill for B5
worksheet.Range["B5"].Interior.Pattern = Pattern.Solid;
worksheet.Range["B5"].Interior.Color = Color.FromArgb(255, 0, 255);
```

After you set the fill style for the cells, if you also want to modify the background color of the cells, refer to [Set Color](#) in this documentation.

Pattern fill

You can integrate pattern fill in cells using the **Pattern** property of the **IInterior** interface to one of the valid pattern types. Pattern fill consists of two parts - background Color and foreground Color.

In order to set the background color, you can use the **Color** ('Color Property' in the on-line documentation), **ColorIndex** ('ColorIndex Property' in the on-line documentation), **ThemeColor** ('ThemeColor Property' in the on-line documentation) and **TintAndShade** ('TintAndShade Property' in the on-line documentation) properties of the **IInterior** interface (on-line documentation). In order to set the foreground color, you can use the **PatternColor** ('PatternColor Property' in the on-line documentation), **PatternColorIndex** ('PatternColorIndex Property' in the on-line documentation), **PatternThemeColor** ('PatternThemeColor Property' in the on-line documentation), **PatternTintAndShade** ('PatternTintAndShade Property' in the on-line documentation) properties of the **IInterior** interface (on-line documentation).



Note: For the **TintAndShade** ('TintAndShade Property' in the on-line documentation) property, it is important to enter a number only from -1(darkest) to 1(lightest). If any value less than -1 or greater than 1 is provided, it will be treated as invalid and an exception will be thrown at runtime. The value zero (0) refers to neutral. Also, the TintAndShade property works only with the **ThemeColor** ('ThemeColor Property' in the on-line documentation) property.

Refer to the following example code to set pattern fill.

C#

```
// Pattern Fill for A1
worksheet.Range["A1"].Interior.Pattern = Pattern.LightDown;
worksheet.Range["A1"].Interior.Color = Color.FromArgb(255, 0, 255);
worksheet.Range["A1"].Interior.PatternColorIndex = 5;
```

Gradient Fill

You can integrate gradient fill in cells using the **Gradient** property (on-line documentation) of the **IInterior** interface (on-line documentation).

Gradient fill can be of two types - Linear Gradient Fill and Rectangle Gradient Fill.

Linear gradient fill

You can set the linear gradient fill using the properties and methods of the **ILinearGradient** interface (on-line documentation).

Refer to the following example code to set linear gradient fill.

C#

```
// Gradient Fill for C1
worksheet.Range["C1"].Interior.Pattern = Pattern.LinearGradient;
(worksheet.Range["C1"].Interior.Gradient as ILinearGradient).ColorStops[0].Color =
Color.FromArgb(255, 0, 0);
(worksheet.Range["C1"].Interior.Gradient as ILinearGradient).ColorStops[1].Color =
Color.FromArgb(255, 255, 0);

(worksheet.Range["C1"].Interior.Gradient as ILinearGradient).Degree = 90;
```

Rectangular gradient fill

You can also set the rectangular gradient fill using the properties and methods of the **IRectangularGradient interface (on-line documentation)**.

Refer to the following example code to set rectangular gradient fill.

C#

```
// Rectangular Gradient Fill for E1
worksheet.Range["E1"].Interior.Pattern = Pattern.RectangularGradient;
(worksheet.Range["E1"].Interior.Gradient as IRectangularGradient).ColorStops[0].Color =
Color.FromArgb(255, 0, 0);
(worksheet.Range["E1"].Interior.Gradient as IRectangularGradient).ColorStops[1].Color =
Color.FromArgb(0, 255, 0);

(worksheet.Range["E1"].Interior.Gradient as IRectangularGradient).Bottom = 0.2;
(worksheet.Range["E1"].Interior.Gradient as IRectangularGradient).Right = 0.3;
(worksheet.Range["E1"].Interior.Gradient as IRectangularGradient).Top = 0.4;
(worksheet.Range["E1"].Interior.Gradient as IRectangularGradient).Left = 0.5;
```

Set font

You can customize the font of a worksheet using the **Font property (on-line documentation)** of IRange interface.

Refer to the following example code to set font style in your worksheet.

C#

```
// Set Font
worksheet.Range["A1"].Value = "GcExcel";
worksheet.Range["A1"].Font.ThemeColor = ThemeColor.Accent1;
worksheet.Range["A1"].Font.TintAndShade = -0.5;
worksheet.Range["A1"].Font.ThemeFont = ThemeFont.Major;
worksheet.Range["A1"].Font.Bold = true;
worksheet.Range["A1"].Font.Size = 20;
worksheet.Range["A1"].Font.Strikethrough = true;
```

Set border

You can customize the border of a worksheet using the **Borders property (on-line documentation)** of the IRange interface.

Refer to the following example code to set border in your worksheet.

C#

```
// Set Border
worksheet.Range["A1:B5"].Borders.LineStyle = BorderLineStyle.DashDot;
worksheet.Range["A1:B5"].Borders.ThemeColor = ThemeColor.Accent1;

worksheet.Range["A1:B5"].Borders[BordersIndex.EdgeRight].LineStyle =
BorderLineStyle.Double;
worksheet.Range["A1:B5"].Borders[BordersIndex.EdgeRight].ThemeColor =
ThemeColor.Accent2;
worksheet.Range["A1:B5"].Borders[BordersIndex.DiagonalDown].LineStyle =
BorderLineStyle.Double;
worksheet.Range["A1:B5"].Borders[BordersIndex.DiagonalDown].ThemeColor =
ThemeColor.Accent5;
```

Set number format

You can set the number format in a worksheet using the **NumberFormat property (on-line documentation)** of the IRange interface.

Refer to the following example code to set number format in your worksheet.

C#

```
// Set Number format
worksheet.Range["A5"].Value = 12;
worksheet.Range["A5"].NumberFormat = "$#,##0.00";
```

Set alignment

You can customize the alignment of a worksheet using any of the properties : **HorizontalAlignment property (on-line documentation)**, **VerticalAlignment property (on-line documentation)**, **AddIndent property (on-line documentation)** and **ReadingOrder property (on-line documentation)** of the IRange interface.

Refer to the following example code to set alignment in your worksheet.

C#

```
// Set Alignment
worksheet.Range["B8"].HorizontalAlignment = HorizontalAlignment.Distributed;
worksheet.Range["B8"].AddIndent = true;
worksheet.Range["B8"].VerticalAlignment = VerticalAlignment.Top;
worksheet.Range["B8"].ReadingOrder = ReadingOrder.RightToLeft;
```

Set protection

You can set protection for your worksheet using the **FormulaHidden property (on-line documentation)** and **Locked property (on-line documentation)** of the IRange interface.

Refer to the following example code to set protection for your worksheet.

C#

```
//Set Protection
worksheet.Range["C4"].Locked = true;
worksheet.Range["C4"].FormulaHidden = true;
```

Create and Set Custom Named Style

Named style is a custom cell style that you apply to your workbook or worksheet with a unique name, which is different from the already existing built-in style names defined for a spreadsheet.

You can create and set custom named styles as and when required. You can also modify an existing style and save it as another workbook style. In GcExcel .NET, **Styles** refers to the named style collection that stores both the built-in and custom named styles.

While working with styles in the spreadsheets, you can use any of the following ways -

- **Create and Set a Custom Named Style**
- **Modify an Existing Style and Save it as a New Workbook Style**

Create and Set a Custom Named Style

GcExcel .NET enables you to define custom named styles for your worksheet, configure it as per your preferences and store them in the collection so that they can be accessed later.

You can add a custom named style to your worksheet using the **Add method (on-line documentation)** of **IStyleCollection interface (on-line documentation)**. This method can also be used to return an IStyle instance. If you want to configure the named style settings in your spreadsheet, you can use the properties of the **IStyle interface (on-line documentation)**.

Refer to the following example code to create a custom name style and configure its settings.

C#

```
//Add custom name style.
IStyle style = workbook.Styles.Add("SampleStyle");
```

```
//Config custom name style settings begin.
//Border
style.Borders[BordersIndex.EdgeLeft].LineStyle = BorderLineStyle.Thin;
style.Borders[BordersIndex.EdgeTop].LineStyle = BorderLineStyle.Thick;
style.Borders[BordersIndex.EdgeRight].LineStyle = BorderLineStyle.Double;
style.Borders[BordersIndex.EdgeBottom].LineStyle = BorderLineStyle.Double;
style.Borders.Color = Color.FromArgb(0, 255, 0);

//Protection
style.FormulaHidden = true;
style.Locked = false;

//Number
style.NumberFormat = "#,##0_);[Red](#,##0)";

//Alignment
style.HorizontalAlignment = HorizontalAlignment.Right;
style.VerticalAlignment = VerticalAlignment.Bottom;
style.WrapText = true;
style.IndentLevel = 5;
style.Orientation = 45;

//Fill
style.Interior.ColorIndex = 5;
style.Interior.Pattern = GrapeCity.Documents.Excel.Pattern.Down;
style.Interior.PatternColor = Color.FromArgb(0, 0, 255);
style.IncludeAlignment = false;
style.IncludeBorder = true;
style.IncludeFont = false;
style.IncludeNumber = true;
style.IncludePatterns = false;
style.IncludeProtection = true;
//Config custom name style settings end.
```

You can also get or set named style in a worksheet using the **Style property (on-line documentation)** of the **IRange interface (on-line documentation)**. The **Styles** collection stores both built-in and custom named styles in GcExcel .NET.

Refer to the following example code to get or set named style in your worksheet.

C#

```
//Set range's style to custom name style.
worksheet.Range["A1"].Style = worksheet.Workbook.Styles["SampleStyle"];
```

Modify an Existing Style and Save it as a New Workbook Style

With GcExcel.NET, you don't always need to create a custom named style right from the scratch. Instead, you can modify an existing style (via getting the existing style from the Styles collection) as per your specific preferences and save the new style as another workbook style that can be used as and when required.

Users can use the **Add method (on-line documentation)** in order to add the new style. The newly created custom style will be based on the existing workbook style and will be stored in the **IStyleCollection interface (on-line documentation)** so that it can be used as another workbook style in the future.

Refer to the following example code in order to modify an existing style and save it as a new workbook style in the Styles collection.

C#

```
// Create workbook
Workbook workbook = new Workbook();

// Fetch the default worksheet
IWorksheet worksheet = workbook.Worksheets[0];
```

```
// Fetch existing Style "Good" and set to Range A1's Style
worksheet.Range["A1"].Style = workbook.Styles["Good"];

// Setting Cell Text
worksheet.Range["A1"].Value = "Good";

// Create and modify a style based on current existing style "Good" and name it as "MyGood"
IStyle myGood = workbook.Styles.Add("MyGood", workbook.Styles["Good"]);
myGood.Font.Bold = true;
myGood.Font.Italic = true;

// Set new style "MyGood" to Range B1's Style
worksheet.Range["B1"].Style = workbook.Styles["MyGood"];

// Setting Cell Text
worksheet.Range["B1"].Value = "MyGood";

// Saving the workbook
workbook.Save(@"6 - AddWorkbookStyles.xlsx");
```

Apply Comments

You can add comments in cells to annotate a worksheet with additional data related to the information entered in that particular cell. The inserted comments can be edited, formatted, resized, moved and deleted.

The following tasks can be performed while applying comments in cells of a spreadsheet:

- **Add comment to a cell**
- **Set comment layout**
- **Show/Hide comment**
- **Author comments**
- **Set rich text for comment**
- **Delete comment**

Add comment to a cell

In GcExcel .NET, a cell comment instance is represented by the **IComment interface (on-line documentation)**. You can insert comment to a cell or a range of cells using the **AddComment method (on-line documentation)** of the **IRange interface (on-line documentation)**. You can also set the text of the comment using the **Text property (on-line documentation)** of the **IComment interface (on-line documentation)**.

Refer to the following example code to add comment to a cell.

```
C#
//Change the text of the comment.
commentC3.Text = "Range C3's new comment.";
```

Set comment layout

You can configure the layout of the comment added to an individual cell or a range of cells using **Shape property (on-line documentation)** of the **IComment interface (on-line documentation)**.

Refer to the following example code to set comment layout.

```
C#
//Configure comment layout
```

```
commentC3.Shape.Line.Color.RGB = Color.Green;
commentC3.Shape.Line.Weight = 7;

commentC3.Shape.Fill.Color.RGB = Color.Gray;
commentC3.Shape.Width = 100;
commentC3.Shape.Height = 200;
commentC3.Shape.TextFrame.TextRange.Font.Bold = true;
commentC3.Visible = true;
```

Show/Hide comment

You can choose to keep comments hidden or visible by using the **Visible property (on-line documentation)** of the **IComment interface (on-line documentation)**.

Refer to the following example code to show/hide comment added to a cell.

C#

```
//Hide Comment
worksheet.Range["C3"].Comment.Visible = true;
worksheet.Range["C:C"].EntireColumn.Hidden = true;

//Show Comment
var commentVisible = worksheet.Range["C3"].Comment.Visible;
//Range C3's comment container shape's visible is false, because column hidden.
var shapeVisible = worksheet.Range["C3"].Comment.Shape.Visible;
```

Author comments

You can represent the author of the comment by using the **Author property (on-line documentation)** of the **IComment interface (on-line documentation)**. Also, you can use this property to change the author of an existing comment.

Refer to the following example code to set comment author for a cell.

C#

```
// Set comment author
workbook.Author = "joneshan";
worksheet.Range["H6"].AddComment("H6's comment.");
//H6's comment author is "joneshan".
var authorH6 = worksheet.Range["H6"].Comment.Author;
```

Set rich text for comment

You can set the rich text for the comment using the properties and methods of the **ITextFrame Interface (on-line documentation)** that control the text style.

Refer to the following example code to set rich text for the comment.

C#

```
//Rich Text and TextFrame
commentC3.Shape.TextFrame.TextRange.Paragraphs.Add("aaa");
commentC3.Shape.TextFrame.TextRange.Paragraphs.Add("bbb", 0);

commentC3.Shape.TextFrame.TextRange.Paragraphs[0].Font.Bold = true;
commentC3.Shape.TextFrame.TextRange.Paragraphs[1].Font.Size = 30;

commentC3.Shape.TextFrame.TextRange.Paragraphs[1].Runs.Add("ccc", 0);
commentC3.Shape.TextFrame.TextRange.Paragraphs[1].Runs.Add("ddd");
commentC3.Shape.TextFrame.TextRange.Paragraphs[1].Runs.Add("eee", 1);

commentC3.Shape.TextFrame.TextRange.Paragraphs[1].Runs[2].Font.Italic = true;
```

```
commentC3.Visible = true;

var text = commentC3.Text;
```

Delete comment

You can delete the comment added to a cell or to a cell range using the **Delete method (on-line documentation)** of the **IComment interface (on-line documentation)** and the **IRange interface (on-line documentation)** respectively.

Refer to the following example code to delete comment from a cell.

```
C#
// Delete Comment instance
commentC3.Delete();
```

Set Rich Text in a Cell

GcExcel.NET provides support for applying rich text formatting in the cells of the worksheet. By default, when textual information is entered in a cell, the alphabets are displayed without any formatting style. Rich text feature is useful for incorporating multiple styles to the textual information in a cell as per the custom requirements and preferences.

Let's say you have a worksheet wherein the cells contain some characters that need to be highlighted to a greater extent in order to emphasize on important information like the name of an organization, company's flagship product, a number, or any other sensitive data. In such a scenario, rich text feature comes in handy while setting multiple styles in a cell.

In the following example, cell A1 contains a string where rich text formatting has been applied. The word "Documents" is formatted with a custom font size, underline style and blue color. Similarly, the text "GrapeCity" and "Excel" has been formatted using multiple styles.

	A	B	C
1	GrapeCity Documents for Excel		
2			
3			

You can set the rich text in the cells of a worksheet by using any of the following ways -

- Using the **IRichText Interface**.
- Using the **IRange.Characters()**.
- Using the **IRange.Characters() to Configure Font Across Several Runs**.
- Using the **ITextRun.InsertAfter()** and **ITextRun.InsertBefore()**.

Using the IRichText Interface.

The **Add ('Add Method' in the on-line documentation)** method of the **IRichText ('IRichText Interface' in the on-line documentation)** interface can be used to add specific ranges of text to the RichText collection of IText runs.

Using Code

Refer to the following example code in order to set rich text in the cells of a worksheet using the IRichText

interface.

C#

```
// Setting column "A" width
worksheet.Range["A1"].ColumnWidth = 70;

// Using IRichText interface to add rich text in cell range A1

// Fetch the IRichText object associated with the cell range
IRichText richText = worksheet.Range["A1"].RichText;

// Add string "GrapeCity " to IRichText object and apply formatting
ITextRun run1 = richText.Add("GrapeCity ");
run1.Font.Color = Color.Red;
run1.Font.Bold = true;
run1.Font.Size = 20;

// Append string "Documents" to IRichText object and apply formatting
ITextRun run2 = richText.Add("Documents");
run2.Font.ThemeFont = ThemeFont.Major;
run2.Font.ThemeColor = ThemeColor.Accent1;
run2.Font.Size = 30;
run2.Font.Underline = UnderlineType.Single;

// Append string " for " to IRichText object
richText.Add(" for ");

// Append string "Excel" to IRichText object and apply formatting
ITextRun run3 = richText.Add("Excel");
run3.Font.Name = "Arial Black";
run3.Font.Color = Color.LightGreen;
run3.Font.Size = 36;
run3.Font.Italic = true;
```

Using the IRange.Characters()

The **Characters()** ('Characters Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface can be used to represent a range of characters within the text entered in the cell. This method will be called only when the value of the cell is in the string format.

Using Code

Refer to the following example code in order to set rich text in the cells of a worksheet.

C#

```
// Setting column "A" width
worksheet.Range["A1"].ColumnWidth = 70;

// Use IRange.Characters() to add rich text

// Setting Cell Text
worksheet.Range["A1"].Value = "GrapeCity Documents for Excel";

// Extracting character ranges from cell text and applying different formatting rules
to each range

// Formatting string "Grapecity"
ITextRun run1 = worksheet.Range["A1"].Characters(0, 9);
run1.Font.Color = Color.Red;
```

```
run1.Font.Bold = true;
run1.Font.Size = 20;

// Formatting string "Documents"
ITextRun run2 = worksheet.Range["A1"].Characters(10, 9);
run2.Font.ThemeFont = ThemeFont.Major;
run2.Font.ThemeColor = ThemeColor.Accent1;
run2.Font.Size = 30;
run2.Font.Underline = UnderlineType.Single;

// Formatting string "Excel"
ITextRun run3 = worksheet.Range["A1"].Characters(24, 5);
run3.Font.Name = "Arial Black";
run3.Font.Color = Color.LightGreen;
run3.Font.Size = 36;
run3.Font.Italic = true;
```

Using the `IRange.Characters()` to Configure Font Across Several Runs

You can also insert rich text in the cells of a worksheet via using the **Characters()** ('**Characters Method**' in the **on-line documentation**) method of the **IRange** ('**IRange Interface**' in the **on-line documentation**) interface in order to configure the font across several runs and then consolidate them into a single entity.

Using Code

Refer to the following example code in order to set rich text in the cells of a worksheet.

C#

```
// Setting column "A" width
worksheet.Range["A1"].ColumnWidth = 75;

// Use IRange.Characters() to configure font across several runs

// Fetch the IRichText object associated with the cell range
IRichText richText = worksheet.Range["A1"].RichText;

// Add string "GrapeCity " to IRichText object and apply formatting
ITextRun run1 = richText.Add("GrapeCity ");
run1.Font.Color = Color.Red;
run1.Font.Bold = true;
run1.Font.Size = 20;

// Append string "Documents" to IRichText object and apply formatting
ITextRun run2 = richText.Add("Documents");
run2.Font.ThemeFont = ThemeFont.Major;
run2.Font.ThemeColor = ThemeColor.Accent1;
run2.Font.Size = 30;
run2.Font.Underline = UnderlineType.Single;

// Append string " for " to IRichText object
richText.Add(" for ");

// Append string "Excel" to IRichText object and apply formatting
ITextRun run3 = richText.Add("Excel");
run3.Font.Name = "Arial Black";
run3.Font.Color = Color.LightGreen;
run3.Font.Color = Color.LightGreen;
run3.Font.Size = 36;
run3.Font.Italic = true;
```

```
// Create composite run
// Extract character range composed of "City" word from run1 and " for" word and apply
formatting
ITextRun compositeRun = worksheet.Range["A1"].Characters(5, 18);
compositeRun.Font.Bold = true;
compositeRun.Font.Italic = true;
compositeRun.Font.ThemeColor = ThemeColor.Accent1;
```

Using the ITextRun.InsertAfter() and ITextRun.InsertBefore

The **ITextRun** (**'ITextRun Interface' in the on-line documentation**) interface provides the properties and methods for adding and customizing the rich text entered in the cells of the worksheet. The **InsertAfter()** (**'InsertAfter Method' in the on-line documentation**) and **InsertBefore()** (**'InsertBefore Method' in the on-line documentation**) methods of the ITextRun interface can be used to insert rich text after and before a range of characters respectively. Also, you can use the **Delete method (on-line documentation)** of the ITextRun interface in order to delete the inserted rich text in the cells.

Using Code

Refer to the following example code in order to set rich text in the cells of a worksheet.

C#

```
// Setting column "A" width
worksheet.Range["A1"].ColumnWidth = 70;

// Using ITextRun.InsertAfter() and InsertBefore() to add rich text

// Fetch the IRichText object associated with the cell range
IRichText richText = worksheet.Range["A1"].RichText;

// Add string " for " to IRichText object
ITextRun run1 = richText.Add(" for ");

// Use InsertBefore() to add string "Documents" to run1 and apply formatting
ITextRun run2 = run1.InsertBefore("Documents");
run2.Font.ThemeFont = ThemeFont.Major;
run2.Font.ThemeColor = ThemeColor.Accent1;
run2.Font.Size = 30;
run2.Font.Underline = UnderlineType.Single;

// Use InsertBefore() to add string "GrapeCity " to run2 and apply formatting
ITextRun run3 = run2.InsertBefore("GrapeCity ");
run3.Font.Color = Color.Red;
run3.Font.Bold = true;
run3.Font.Size = 20;

// Use InsertAfter() to add string "Excel" to run1 and apply formatting
ITextRun run4 = run1.InsertAfter("Excel");
run4.Font.Name = "Arial Black";
run4.Font.Color = Color.LightGreen;
run4.Font.Size = 36;
run4.Font.Italic = true;
```

Manage Formulas

You can add formulas in a worksheets to carry out complex calculations on numerical data residing in cells or a range of cells. Formulas are written as algebraic expressions, statements, or equations that start with an "=" (equal to) sign. The computation of a formula always begins from left and extends towards the right as per the operator precedence. In case you want to modify the order of computation, you can enclose some specific portions within the formula in parentheses.

Shared below is the descending order of operations for GcExcel .NET formulas with the first one holding the maximum precedence and last one holding the minimum precedence.

1. Parentheses evaluation of expressions
2. Range evaluation
3. Evaluation of spaces within the expression.
4. Evaluation of commas within the expression
5. Evaluation of variables with negation sign (-)
6. Conversion of percentages(%)
7. Evaluation of exponents (with ^ sign)
8. Multiplication and Division operators (hold equal precedence).
9. Addition and Subtraction operators (hold equal precedence).
10. Evaluation of text operators
11. Evaluation of comparison operators (=,<>,<=,>=)

In GcExcel .NET, managing formulas involves the following tasks.

- [Set Formula to Range](#)
- [Set Table Formula](#)
- [Set Array Formula](#)

Set Formula to Range

You can set formula to a cell range using the **Formula property (on-line documentation)** of the **IRange interface (on-line documentation)**.

Refer to the following example code to add custom names and set formula to a range in a worksheet.

C#

```
// Add custom name and set formula to range
worksheet.Names.Add("test1", "=Sheet1!$A$1");
worksheet.Names.Add("test2", "=Sheet1!test1*2");

worksheet.Range["A1"].Value = 1;
//C6's value is 1.
worksheet.Range["C6"].Formula = "=test1";
//C7's value is 3.
worksheet.Range["C7"].Formula = "=test1 + test2";
//C8's value is 6.283185307
worksheet.Range["C8"].Formula = "=test2*PI()";
```



Note: The value calculated by the formula is stored in a cache. Users can verify the cached value by invoking the **Dirty method (on-line documentation)** of the **IRange interface**. This method clears the cached value of the specified range and all the ranges dependent on it, or the entire workbook.

Reference style

GcExcel .NET supports the RIC1 reference style to allow users to perform calculations in a much easier and quicker way. To set reference style, you can use the **Reference Style ('ReferenceStyle Property' in the on-line documentation)** property of the **IWorkbook interface (on-line documentation)**.

Refer to the following example code to see how reference style can be set in a workbook.

C#

```
//set workbook's reference style to R1C1.
workbook.ReferenceStyle = ReferenceStyle.R1C1;
```

Set Table Formula

Table formula refers to a formula that is used as a structured reference in the worksheet instead of using it as an explicit cell reference. Structured reference in a table formula is the combination of table and column names in a spreadsheet with syntax rules that must be applied while creating a table formula.

For instance, let us consider an example of a table formula in a spreadsheet.

	A	B	C	D	E	F	G	H
1	SalesPerson	Region	SalesAmount	ComPct	ComAmt			
2	Joe	North	260	10%				
3	Robert	South	660	15%				
4	Michelle	East	940	15%				
5	Erich	West	410	12%				
6	Dafna	North	800	15%				
7	Rob	South	900	15%				
8	Total				0			
9								
10								
11								
12								
13				=SUM(DeptSales[[#Totals],[SalesAmount]],DeptSales[[#Data],[ComAmt]])				
14				SUM(number1, [number2], [number3], ...)				
15								
16								

The structured reference components in the above table formula are described below.

Components**Description****Table Name**

References the table data, without any header or total rows. You can use a default table name, such as Table1, or change it to use a custom name. Example: DeptSales is a custom table name in the table formula.

Column Specifier

Column specifiers use the names of the columns they represent. They reference column data without any column header or total row. Column specifiers must be enclosed in [] square brackets when they are written in the table formula. Example: [SalesAmount] and [ComAmt]

Item Specifier

Refers to a specific portions of the table such as total row. Example: [#Totals] and [#Data]

Table Specifier

Represents the outer portions of the structured reference. Outer references follow table names and are enclosed within the square brackets. Example: [[#Totals],[SalesAmount]], [[#Data],[ComAmt]]

Structures Reference

Represented by a string that begins with the table name and ends with the column specifier. Example: DeptSales[[#Totals],[SalesAmount]] and DeptSales[[#Data],[ComAmt]]

Reference operators

In GcExcel .NET, reference operators are used to combine column specifiers in a table formula.

Shared below is a table that describes the reference operators along with structured reference components and cell range corresponding to the table formula.

Operators**Description****Example**

:(colon) range operator

All of the cells in two or more adjacent columns.

=DeptSales[[SalesPerson]:[Region]]

,(comma) union operator	A combination of two or more columns.	=DeptSales[SalesAmount],DeptSales[ComAmt]
(space) intersection operator	The intersection of two or more columns.	=DeptSales[[SalesPerson]:[SalesAmount]]DeptSales[[Region]:[ComPct]]

Special item specifier

Special item specifier refers to a particular area in a table formula which is identified either with a # prefix or with an @ prefix.

GcExcel .NET supports the following types of special item specifiers:

Special Item Specifier	Description
#All	To the entire table including column headers, data and totals (if any).
#Data	Only the data rows
#Headers	Only the header rows
#Totals	Only the total row. If there is none, it returns null.
#This Row	Cells in the same row as the formula
@	Cells in the same row as the formula

Refer to the following example code to set table formula in your spreadsheets.

C#

```
// Define Data
worksheet.Range["A1:E3"].Value = new object[,]
{
    { "SalesPerson", "Region", "SalesAmount", "ComPct", "ComAmt" },
    { "Joe", "North", 260, 0.10, null },
    { "Robert", "South", 660, 0.15, null },
};

worksheet.Tables.Add(worksheet.Range["A1:E3"], true);
worksheet.Tables[0].Name = "DeptSales";
worksheet.Tables[0].Columns["ComPct"].DataBodyRange.NumberFormat = "0%";

//Use table formula in table range.
worksheet.Tables[0].Columns["ComAmt"].DataBodyRange.Formula = "=[@ComPct]*[@SalesAmount]";

//Use table formula out of table range.
worksheet.Range["F2"].Formula = "=SUM(DeptSales[@SalesAmount])";
worksheet.Range["G2"].Formula = "=SUM(DeptSales[#Data],[SalesAmount])";
worksheet.Range["H2"].Formula = "=SUM(DeptSales[SalesAmount])";
worksheet.Range["I2"].Formula = "=SUM(DeptSales[@ComPct], DeptSales[@ComAmt])";
```

Set Array Formula

Array formula is a formula that can execute multiple calculations on individual cells or a range of cells to display a column or a row of subtotals. The array formula can consist of array of row of values, column of values or simply a combination of rows and columns of values that may return either multiple results or a single result.

Array formulas can be used to simplify the following tasks in a worksheet:

1. You can count the number of characters in a range of cells.
2. You can sum numeric values in cells that meet a specified criteria. For instance, the highest value in a range or values that fall between an upper and lower boundary.
3. You can sum every nth value in a range of cell values in a spreadsheet.

In GcExcel .NET, you can use **FormulaArray property (on-line documentation)** of the **IRange interface (on-line documentation)** to set array formula for a range. In case, you want to find out whether a range has array formula or not, you can use the **HasArray property (on-line documentation)** of the **IRange interface**. In order to get an entire array if specified range is part of an array, you can use **CurrentArray property (on-line**

documentation).

Refer to the following example code to set array formula and get entire array:

C#

```
// Setting cell value using arrays
worksheet.Range["E4:J5"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6}
};

worksheet.Range["I6:J8"].Value = new object[,]
{
    {2, 2},
    {3, 3},
    {4, 4}
};

// To set array formula for range.
//O      P      Q
//2      4      #N/A
//12     15     #N/A
//#N/A   #N/A   #N/A
worksheet.Range["O9:Q11"].FormulaArray = "=E4:G5*I6:J8";

//O9's current array is "O9:Q11". Current array gets the entire array.
var currentarray = worksheet.Range["O9"].CurrentArray.ToString();
```

Manage Custom Functions

GcExcel .NET provides support for adding custom functions, thus enabling users to implement custom arithmetic logic to spreadsheets. These functions run extremely fast, can make web service calls, look similar to the native Excel functions, and can be used across all Excel platforms including major operating systems (Windows, Mac, Mobile OS and Office: both online and offline).

For instance, you can use company's proprietary functions, apply a nested formula with custom functions, or use a combination of standard built-in functions in order to handle complex spreadsheet calculations.

To implement custom functions in GcExcel .NET, you need to create a derived class from the **CustomFunction ('CustomFunction Class' in the on-line documentation)** class and declare the custom function in the new class along with the function name, return type, and parameters. For more details on the steps to add custom functions in your applications, refer to [Work With Custom Functions](#).

Work With Custom Functions

For specific calculation requirements, the available built-in functions may not suffice. In order to perform a particular task, you may want to create a custom function that can be used as desired.

GcExcel .NET provides support for defining and using custom functions in the worksheet.

Using Code

- Step 1: Define a custom function
- Step 2: Register the custom function in your worksheet using the **AddCustomFunction()** ('AddCustomFunction Method' in the on-line documentation) method
- Step 3: Implement the custom function

Shared below are some examples of custom functions that can be created and used to perform complex calculation tasks:

- **Example 1: Conditional Sum Function**
- **Example 2: Custom Concatenation Function**
- **Example 3: Merged Range Function**
- **Example 4: Error Detection Function**

[Example 1: Conditional Sum Function](#)

Refer to the following example code to create and use custom conditional sum function in your spreadsheet. This function can sum cell values based on specific display format or style (like cells with interior color as red).

C#

```
// Step 1- Defining custom function: MyConditionalSum
// Creating a new class MyConditionalSumFunctionX by inheriting the CustomFunction
class
public class MyConditionalSumFunctionX : CustomFunction
{
    public MyConditionalSumFunctionX() : base("MyConditionalSum",
FunctionValueType.Number, CreateParameters())
    {
    }
    private static Parameter[] CreateParameters()
    {
        Parameter[] parameters = new Parameter[254];
        for (int i = 0; i < 254; i++)
        {
            parameters[i] = new Parameter(FunctionValueType.Object, true);
        }
        return parameters;
    }
    public override object Evaluate(object[] arguments, ICalcContext context)
    {

```

```
double sum = 0d;
foreach (var argument in arguments)
{
    foreach (var item in Enumerate(argument))
    {
        if (item is CalcError)
        {
            return item;
        }
        if (item is double)
        {
            sum += (double)item;
        }
    }
}
return sum;
}

private static IEnumerable<object> Enumerate(object obj)
{
    if (obj is IEnumerable<object>)
    {
        foreach (var item in obj as IEnumerable<object>)
        {
            foreach (var item2 in Enumerate(item))
            {
                yield return item2;
            }
        }
    }
    else if (obj is object[,])
    {
        var array = obj as object[,];
        int rowCount = array.GetLength(0);
        int colCount = array.GetLength(1);
        for (int i = 0; i < rowCount; i++)
        {
            for (int j = 0; j < colCount; j++)
            {
                yield return array[i, j];
            }
        }
    }
    else if (obj is CalcReference)
    {
        foreach (var item in Enumerate(obj as CalcReference))
        {
            yield return item;
        }
    }
    yield return obj;
}

private static IEnumerable<object> Enumerate(CalcReference reference)
{
    foreach (var range in reference.GetRanges())
    {
        int rowCount = range.Rows.Count;
        int colCount = range.Columns.Count;
        for (int i = 0; i < rowCount; i++)
        {
            for (int j = 0; j < colCount; j++)
            {

```



```

        {
            return argument;
        }
        if (argument is string || argument is double)
        {
            sb.Append(argument);
        }
    }
    return sb.ToString();
}
}

```

C#

```

// Step2: Register the custom function using AddCustomFunction() method
var workbook = new GrapeCity.Documents.Excel.Workbook();
GrapeCity.Documents.Excel.Workbook.AddCustomFunction(new MyConcatenateFunctionX());
IWorksheet worksheet = workbook.Worksheets[0];

// Step3- Implement the Custom Function
worksheet.Range["A1"].Formula = "=MyConcatenate(\"I\", \" \", \"work\", \" \", \"with\", \" \", \"GcExcel\", \".\")";
worksheet.Range["A2"].Formula = "=MyConcatenate(A1, \"Documents.\")";
// Value of cell A1 is "I work with GcExcel."
var resultA1 = worksheet.Range["A1"].Value;
// Display result in cell C1
worksheet.Range["C1"].Value = resultA1;
// Value of cell A2 is "I work with GcExcel Documents."
var resultA2 = worksheet.Range["A2"].Value;
// Display result in cell C2
worksheet.Range["C2"].Value = resultA2;

```

Example 3: Merged Range Function

Refer to the following example code to create and use custom merged range function in your spreadsheet.

C#

```

// Step 1- Defining custom function: MyIsMergedRange
// Creating a new class MyIsMergedRangeFunctionX by inheriting the CustomFunction class
public class MyIsMergedRangeFunctionX : CustomFunction
{
    public MyIsMergedRangeFunctionX()
        : base("MyIsMergedRange", FunctionValueType.Boolean, new Parameter[] { new
Parameter(FunctionValueType.Object, true) })
    {
    }
    public override object Evaluate(object[] arguments, ICalcContext context)
    {
        if (arguments[0] is CalcReference)
        {
            IEnumerable<IRange> ranges = (arguments[0] as CalcReference).GetRanges();

            foreach (var range in ranges)
            {
                return range.MergeCells;
            }
        }
        return false;
    }
}

```


C#

```
// Step2: Register the custom function using AddCustomFunction() method
var workbook = new GrapeCity.Documents.Excel.Workbook();
GrapeCity.Documents.Excel.Workbook.AddCustomFunction(new MyIsMergedRangeFunctionX());
IWorksheet worksheet = workbook.Worksheets[0];

// Step3- Implement the Custom Function
worksheet.Range["A1:B2"].Merge();
worksheet.Range["C1"].Formula = "=MyIsMergedRange(A1)";
worksheet.Range["C2"].Formula = "=MyIsMergedRange(H2)";
//A1 is a merged cell, Range["C1"]'s value is true.
var resultC1 = worksheet.Range["C1"].Value;
// Display result in cell D1
worksheet.Range["D1"].Value = resultC1;
//H2 is not a merged cell, Range["C2"]'s value is false.
var resultC2 = worksheet.Range["C2"].Value;
// Display result in cell D2
worksheet.Range["D2"].Value = resultC2;
```

Example 4: Error Detection Function

Refer to the following example code to create and use custom error detection function in your spreadsheet.

C#

```
// Step 1- Defining custom function: MyIsError
// Creating a new class MyIsErrorFunctionX by inheriting the CustomFunction class
public class MyIsErrorFunctionX : CustomFunction
{
    public MyIsErrorFunctionX()
        : base("MyIsError", FunctionValueType.Boolean, new Parameter[] { new
Parameter(FunctionValueType.Variant) })
    {
    }
    public override object Evaluate(object[] arguments, ICalcContext context)
    {
        if (arguments[0] is CalcError)
        {
            if ((CalcError)arguments[0] != CalcError.None &&
(CalcError)arguments[0] != CalcError.GettingData)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        return false;
    }
}
```

C#

```
// Step2: Register the custom function using AddCustomFunction() method
var workbook = new Workbook();
Workbook.AddCustomFunction(new MyIsErrorFunctionX());
IWorksheet worksheet = workbook.Worksheets[0];

// Step3: Implement the custom function
worksheet.Range["A1"].Value = CalcError.Num;
worksheet.Range["A2"].Value = 100;
```

```
worksheet.Range["B1"].Formula = "=MyIsError(A1)";
worksheet.Range["B2"].Formula = "=MyIsError(A2)";
// Range["B1"]'s value is true.
var resultB1 = worksheet.Range["B1"].Value;
// Display Result in cell C1
worksheet.Range["C1"].Value = resultB1;
// Range["B2"]'s value is false.
var resultB2 = worksheet.Range["B2"].Value;
// Display Result in cell C2
worksheet.Range["C2"].Value = resultB2;
```

Manage Data

The tasks associated with handling data in GcExcel .NET component includes the following:

- [Use Chart](#)
- [Use Sparkline](#)
- [Use Table](#)
- [Use Pivot Table](#)
- [Use Slicer](#)

Use Chart

You can use charts in spreadsheets to graphically interpret data and visualize large volumes of information quickly and efficiently.

Working with charts involves the following tasks:

- [Create and Delete Chart](#)
- [Configure Chart](#)
- [Customize Chart Objects](#)

Create and Delete Chart

GcExcel .NET allows users to create and delete chart in spreadsheets as per their requirements.

You can create and delete chart using the properties and methods of the **IShapes Interface (on-line documentation)** and the **IChart interface (on-line documentation)**

Create Chart

You can create chart in a worksheet by using the **AddChart method (on-line documentation)** of the IShapes interface.

Refer to the following example code to create a chart.

C#

```
//Add Chart
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "Revenue", "Profit", "Sales"},
    {"North", 10, 25, 25},
    {"East", 51, 36, 27},
    {"South", 52, 85, 30},
    {"West", 22, 65, 65}
};
```

Delete Chart

You can delete an existing chart by using **Delete method (on-line documentation)** of the IChart interface.

Refer to the following example code to delete a chart from your spreadsheet.

C#

```
// Delete Chart
shape.Chart.Delete();
```

Configure Chart

In GcExcel .NET, you can configure a chart added to a spreadsheet in order to set up its display as per your preferences.

Following tasks can be performed while configuring a chart:

- [Chart Title](#)
- [Chart Area](#)
- [Plot Area](#)

Chart Title

In GcExcel .NET, you can use the properties of the **IChart Interface (on-line documentation)** to set up the chart title as per your choice. When working with chart title, you can perform the following tasks:

- **Set formula for chart title**
- **Set format for chart title and font style**

Set formula for chart title

Refer to the following example code to set formula for chart title.

C#

```
//Set formula for chart title.  
shape.Chart.HasTitle = true;  
shape.Chart.ChartTitle.Formula = "=Sheet1!$E$1";  
worksheet.Range["E1"].Value = "Sample Chart";
```

Set format for chart title and font style

Refer to the following example code to set format for chart title and font style.

C#

```
//Set chart title's format and font style.  
shape.Chart.HasTitle = true;  
//shape.Chart.ChartTitle.Text = "aaaaa";  
shape.Chart.ChartTitle.Font.Bold = true;  
shape.Chart.ChartTitle.Format.Fill.Color.RGB = Color.Red;  
shape.Chart.ChartTitle.Format.Line.Color.RGB = Color.Blue;
```

Chart Area

In GcExcel .NET, you can use the properties of the **IChartArea interface (on-line documentation)** to set up the chart area as per your preferences.

This topic includes the following tasks:

- **Configure chart area style**
- **Set chart area format**

Configure chart area style

You can configure the chart area style by changing its font, format and other attributes using the **Font property (on-line documentation)**, **Format property (on-line documentation)** and **RoundedCorners property (on-line documentation)** of the IChartArea interface.

Refer to the following example code to configure chart area style in your worksheet.

C#

```
//Configure chart area style
```

```

IShape shape = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IChartArea chartarea = shape.Chart.ChartArea;
//Format.
chartarea.Format.Fill.Color.RGB = Color.Gray;
chartarea.Format.Line.Color.RGB = Color.Gold;
chartarea.Format.ThreeD.RotationX = 60;
chartarea.Format.ThreeD.RotationY = 20;
chartarea.Format.ThreeD.RotationZ = 100;
chartarea.Format.ThreeD.Z = 20;
chartarea.Format.ThreeD.Perspective = 20;
chartarea.Format.ThreeD.Depth = 5;
//Font
chartarea.Font.Bold = true;
chartarea.Font.Italic = true;
chartarea.Font.Color.RGB = Color.Red;
//Rounded corners.
chartarea.RoundedCorners = true;

```

Set chart area format

Refer to the following example code to set chart area format in your worksheet.

C#

```

IShape shape = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IChartArea chartarea = shape.Chart.ChartArea;
//Format.
chartarea.Format.Fill.Color.RGB = Color.Gray;
chartarea.Format.Line.Color.RGB = Color.Gold;
chartarea.Format.ThreeD.RotationX = 60;
chartarea.Format.ThreeD.RotationY = 20;
chartarea.Format.ThreeD.RotationZ = 100;
chartarea.Format.ThreeD.Z = 20;
chartarea.Format.ThreeD.Perspective = 20;
chartarea.Format.ThreeD.Depth = 5;
//Font
chartarea.Font.Bold = true;
chartarea.Font.Italic = true;
chartarea.Font.Color.RGB = Color.Red;

```

```
//rounded corners.  
chartarea.RoundedCorners = true;
```

Plot Area

In GcExcel .NET, you can use the properties of the **IPlotArea Interface (on-line documentation)** to set up the plot area in a chart as per your preferences.

Configure plot area format

You can configure the plot area format by changing its fill color, line color and other attributes using the **Format property (on-line documentation)** of the IPlotArea interface.

Refer to the following example code to configure plot area format for a chart inserted in your worksheet.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 100, 300, 300);  
worksheet.Range["A1:D6"].Value = new object[,]  
{  
    {null, "S1", "S2", "S3"},  
    {"Item1", 10, 25, 25},  
    {"Item2", -51, -36, 27},  
    {"Item3", 52, -85, -30},  
    {"Item4", 22, 65, 65},  
    {"Item5", 23, 69, 69}  
};  
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);  
  
IPlotArea plotarea = shape.Chart.PlotArea;  
//Format.  
plotarea.Format.Fill.Color.RGB = Color.Pink;  
plotarea.Format.Line.Color.RGB = Color.Green;
```

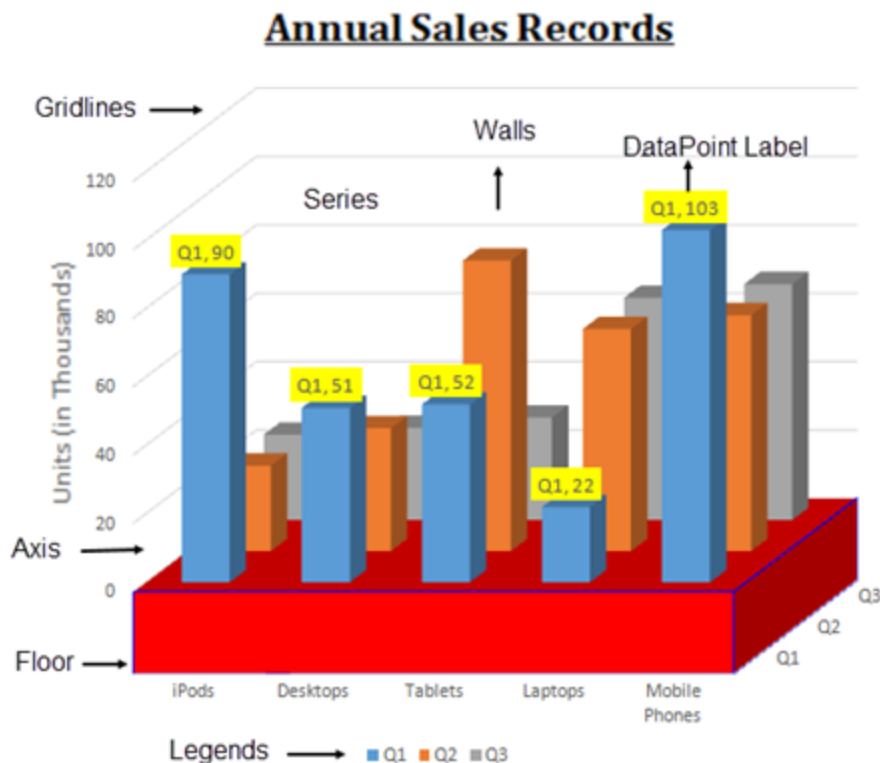
Customize Chart Objects

In GcExcel .NET, the chart feature provides extensive support for creating various types of charts including both 2-D and 3-D views.

Chart objects are fully customizable. Shared below is a list of charting objects that can be modified in charts created using GcExcel .NET:

1. [Series](#)
2. [Walls](#)
3. [Axis and other Lines](#)
4. [Floor](#)
5. [Data Label](#)
6. [Legends](#)

The following diagram displays a sample chart depicting the annual sales records of different electronic gadgets per quarter along with the chart objects that can be customized in a worksheet.



Series

Series refers to a set of data points, or simply a list of values that are plotted in a chart.

In a spreadsheet, you can plot one or more data series while creating a chart. Each series is represented by an item on the legend and provides access to the chart control's collection of series objects.

In GcExcel .NET, the `SeriesCollection` can be used to create chart series. The properties and methods of the **`ISeries interface (on-line documentation)`** and the **`ISeriesCollection interface (on-line documentation)`** allows users to add individual series, access it, delete it and perform other useful operations on it as per the requirements.

Refer to the following example code to add series in your chart.

C#

```
// Adding charts
IShape shapel = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 50, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", 51, 36, 27},
    {"Item3", 52, 85, 30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};

//Detects three series, B2:B6, C2:C6, D2:D6.
//Does not detect out series labels and category labels, auto generated.
shapel.Chart.SeriesCollection.Add(worksheet.Range["B2:D6"]);

IShape shape2 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 550, 50, 300,
```

```

300);
//Detects three series, B2:B6, C2:C6, D2:D6.
//Detects out series labels and category labels.
//Series labels are "S1", "S2", "S3".
//Category labels are "Item1", "Item2", "Item3", "Item4", "Item5".
shape2.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"]);

IShape shape3 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 450, 300,
300);
//Detects five series, B2:D2, B3:C3, B4:C4, B5:C5, B6:C6.
//Does not detects out series labels and category labels, auto generated.
shape3.Chart.SeriesCollection.Add(worksheet.Range["B2:D6"], RowCol.Rows);

IShape shape4 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 550, 450, 300,
300);
//Detects three series, B2:B6, C2:C6, D2:D6
//Does not detects out series labels and category labels, auto generated.
shape4.Chart.SeriesCollection.Add(worksheet.Range["B2:D6"], RowCol.Columns);

IShape shape5 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 850, 450, 300,
300);
//Detects three series, B2:B6, C2:C6, D2:D6
//Detects out series labels and category labels.
//Series labels are "S1", "S2", "S3".
//Category labels are "Item1", "Item2", "Item3", "Item4", "Item5".
shape5.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns);

IShape shape6 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 750, 300,
300);
//Detects three series, B2:B6, C2:C6, D2:D6
//Detects out series labels and category labels.
//Series labels are "S1", "S2", "S3".
//Category labels are "Item1", "Item2", "Item3", "Item4", "Item5".
shape6.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);

workbook.Worksheets.Add();
IWorksheet worksheet1 = workbook.Worksheets[1];
worksheet1.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};

//Use ISeriesCollection.NewSeries() to add series
IShape shape7 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 50, 300,
300);
ISeries series1 = shape7.Chart.SeriesCollection.NewSeries();
ISeries series2 = shape7.Chart.SeriesCollection.NewSeries();
ISeries series3 = shape7.Chart.SeriesCollection.NewSeries();
series1.Formula = "=SERIES(Sheet1!$B$1,Sheet1!$A$2:$A$6,Sheet1!$B$2:$B$6,1)";
series2.Formula = "=SERIES(Sheet1!$C$1,Sheet1!$A$2:$A$6,Sheet1!$C$2:$C$6,2)";
series3.Formula = "=SERIES(Sheet1!$D$1,Sheet1!$A$2:$A$6,Sheet1!$D$2:$D$6,3)";

//Use ISeriesCollection.Extend(IRange source, RowCol rowcol, bool categoryLabels) to
add new data points to existing series
IShape shape8 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 450, 300,

```



```

300);
    shape8.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
    worksheet1.Range["A12:D14"].Value = new object[,]
    {
        {"Item6", 50, 20, -30},
        {"Item7", 60, 50, 50},
        {"Item8", 35, 80, 60}
    };
    shape8.Chart.SeriesCollection.Extend(worksheet1.Range["A12:D14"], RowCol.Columns,
true);

    workbook.Worksheets.Add();
    IWorksheet worksheet2 = workbook.Worksheets[2];
    worksheet2.Range["A1:D6"].Value = new object[,]
    {
        {null, "S1", "S2", "S3"},
        {"Item1", 10, 25, 25},
        {"Item2", -51, -36, 27},
        {"Item3", 52, -85, -30},
        {"Item4", 22, 65, 65},
        {"Item5", 23, 69, 69}
    };

    //Create a line chart, change one series's AxisGroup, change another one series's
chart type.
    IShape shape9 = worksheet2.Shapes.AddChart(ChartType.Line, 200, 50, 300, 300);
    shape9.Chart.SeriesCollection.Add(worksheet2.Range["A1:D6"], RowCol.Columns, true,
true);
    ISeries series4 = shape9.Chart.SeriesCollection[0];
    ISeries series5 = shape9.Chart.SeriesCollection[1];
    series4.AxisGroup = AxisGroup.Secondary;
    series5.ChartType = ChartType.ColumnClustered;

    //Set 3D column chart's bar shape.
    IShape shape10 = worksheet2.Shapes.AddChart(ChartType.Column3D, 200, 450, 300, 300);
    shape10.Chart.SeriesCollection.Add(worksheet2.Range["A1:D6"], RowCol.Columns, true,
true);
    ISeries series6 = shape10.Chart.SeriesCollection[0];
    ISeries series7 = shape10.Chart.SeriesCollection[1];
    ISeries series8 = shape10.Chart.SeriesCollection[2];
    series6.BarShape = BarShape.ConeToMax;
    series7.BarShape = BarShape.Cylinder;
    series8.BarShape = BarShape.PyramidToPoint;

    //Set negative point's fill color.
    IShape shape11 = worksheet2.Shapes.AddChart(ChartType.Column3D, 200, 800, 300, 300);
    shape11.Chart.SeriesCollection.Add(worksheet2.Range["A1:D6"], RowCol.Columns, true,
true);
    ISeries series9 = shape11.Chart.SeriesCollection[0];
    series9.InvertIfNegative = true;
    //Issue to be escalated
    series9.InvertColor.RGB = Color.DarkOrange;

    //Set series' plot order.6
    IShape shape12 = worksheet2.Shapes.AddChart(ChartType.ColumnClustered, 200, 1100,
300, 300);
    worksheet.Range["A1:E6"].Value = new object[,]
    {
        {null, "S1", "S2", "S3", "S4"},
        {"Item1", 10, 25, 25, 30},

```

```

{"Item2", -51, -36, 27, 35},
{"Item3", 52, -85, -30, 40},
{"Item4", 22, 65, 65, 45},
{"Item5", 23, 69, 69, 50}
};
shape12.Chart.SeriesCollection.Add(worksheet2.Range["A1:E6"], RowCol.Columns, true,
true);

ISeries series10 = shape12.Chart.SeriesCollection[0];
ISeries series11 = shape12.Chart.SeriesCollection[1];
ISeries series12 = shape12.Chart.SeriesCollection[2];
ISeries series13 = shape12.Chart.SeriesCollection[3];

//series11 and series13 plot on secondary axis.
series11.AxisGroup = AxisGroup.Secondary;
series13.AxisGroup = AxisGroup.Secondary;

//series10 and series12 are in one chart group.
series12.PlotOrder = 1;
series10.PlotOrder = 2;

//series4 and series2 are in one chart group.
series13.PlotOrder = 1;
series11.PlotOrder = 2;

//Config series' marker.
IShape shape13 = worksheet2.Shapes.AddChart(ChartType.Line, 200, 1450, 300, 300);
shape13.Chart.SeriesCollection.Add(worksheet2.Range["A1:D6"], RowCol.Columns, true,
true);

ISeries series14 = shape13.Chart.SeriesCollection[0];

series14.MarkerStyle = MarkerStyle.Diamond;
series14.MarkerSize = 10;
series14.MarkerFormat.Fill.Color.RGB = Color.Red;
series14.MarkerFormat.Line.Style = LineStyle.ThickThin;
series14.MarkerFormat.Line.Color.RGB = Color.Green;
series14.MarkerFormat.Line.Weight = 3;

```

Configure Chart Series

In GcExcel .NET, you can configure chart series using the following in your spreadsheet:

- **DataPoint**
- **DataLabel**
- **Trendline**
- **ChartGroup**
- **DropLine, HiLoLine and SeriesLine**
- **Up-Down Bars**

DataPoint

The Points collection in GcExcel .NET is used to represent all the points in a specific series and the indexer notation of the **IPoints interface (on-line documentation)** to get a specific point in the series. Also, you can use the **DataLabel property (on-line documentation)** of the **IPoint interface (on-line documentation)** to get data label of a specific point.

Set the format of DataPoint

Refer to the following example code to set data point format for the chart inserted in your worksheet.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
ISeries series2 = shape.Chart.SeriesCollection[1];
ISeries series3 = shape.Chart.SeriesCollection[2];

series1.Format.Fill.Color.RGB = Color.Blue;
series1.Points[2].Format.Fill.Color.RGB = Color.Green;
```

Configure secondary section for pie of a pie chart

You can use the **SecondaryPlot property (on-line documentation)** of the IPoint interface to set if the point lies in the secondary section of either a pie of pie chart or a bar of pie chart.

Refer to the following example code to configure secondary section for pie of a pie chart.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.PieOfPie, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

shape.Chart.ChartGroups[0].SplitType = ChartSplitType.SplitByCustomSplit;
series1.Points[0].SecondaryPlot = true;
series1.Points[1].SecondaryPlot = false;
series1.Points[2].SecondaryPlot = true;
series1.Points[3].SecondaryPlot = false;
series1.Points[4].SecondaryPlot = true;
```

DataLabel

The DataLabels collection in GcExcel .NET is used to represent the collection of all the data labels for the specified series.

You can use the **Font property (on-line documentation)** and **Format property (on-line documentation)** of the **IDataLabel interface (on-line documentation)** to set font style, fill, line and 3-D formatting for all the data labels of the specified series. You can also configure the layout of the data labels using other properties of the IDataLabel interface.

Set all data labels and specific data label format for series

Refer to the following example code to set series' all data labels and specific data label format.

C#

```

IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

//set series1's all data label's format.
series1.DataLabels.Format.Fill.Color.RGB = Color.Green;
series1.DataLabels.Format.Line.Color.RGB = Color.Red;
series1.DataLabels.Format.Line.Weight = 3;

//set series1's specific data label's format.
series1.DataLabels[2].Format.Fill.Color.RGB = Color.Yellow;
series1.Points[2].DataLabel.Format.Line.Color.RGB = Color.Blue;
series1.Points[2].DataLabel.Format.Line.Weight = 5;

```

Customize data label text

Refer to the following example code to customize the text of the data label.

C#

```

IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

//customize data labels' text.
series1.DataLabels.ShowCategoryName = true;
series1.DataLabels.ShowSeriesName = true;
series1.DataLabels.ShowLegendKey = true;

```

Trendline

The Trendlines collection in GcExcel .NET is used to represent a collection of trend lines for a specific series. You can use the **Add Method (on-line documentation)** of the **ITrendlines interface (on-line documentation)** to create a new trendline for a specific series. Also, you can use the indexer notation of the ITrendlines interface to

get a specific trend line.

Add trendline for series and configure its style

Refer to the following example code to add trendline for series and configure its style.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.Trendlines.Add();
series1.Trendlines[0].Type = TrendlineType.Linear;
series1.Trendlines[0].Forward = 5;
series1.Trendlines[0].Backward = 0.5;
series1.Trendlines[0].Intercept = 2.5;
series1.Trendlines[0].DisplayEquation = true;
series1.Trendlines[0].DisplayRSquared = true;
```

Add two trendlines for one series

Refer to the following example code to add two trendlines for one series.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.Trendlines.Add();
series1.Trendlines[0].Type = TrendlineType.Linear;
series1.Trendlines[0].Forward = 5;
series1.Trendlines[0].Backward = 0.5;
series1.Trendlines[0].Intercept = 2.5;
series1.Trendlines[0].DisplayEquation = true;
series1.Trendlines[0].DisplayRSquared = true;

series1.Trendlines.Add();
series1.Trendlines[1].Type = TrendlineType.Polynomial;
series1.Trendlines[1].Order = 3;
```

Chart Group

Chart Group contains common settings for one or more series. Typically, it is a group of specific featured series.

Set varied colors for column chart with one series

Refer to the following example code to set different colors for a column chart which has only one series.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.SeriesCollection[2].Delete();
shape.Chart.SeriesCollection[1].Delete();
//Chart's series count is 1.
var count = shape.Chart.SeriesCollection.Count;
//set vary colors for column chart which only has one series.
shape.Chart.ColumnGroups[0].VaryByCategories = true;
```

Set split setting and gap width for pie of a pie chart

Refer to the following example code to set split setting and gap width for pie of a pie chart.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.PieOfPie, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

shape.Chart.PieGroups[0].SplitType = ChartSplitType.SplitByValue;
shape.Chart.PieGroups[0].SplitValue = 20;
shape.Chart.PieGroups[0].GapWidth = 350;
```

Set gap width of column chart and overlap

Refer to the following example code in order to set the gap width of the column chart along with overlap.

C#

```
//Set column chart's gap width and overlap
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
```

```

        {"Item2", -51, -36, 27},
        {"Item3", 52, -85, -30},
        {"Item4", 22, 65, 65},
        {"Item5", 23, 69, 69}
    };
    shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

    ISeries series1 = shape.Chart.SeriesCollection[0];
    series1.HasDataLabels = true;

    shape.Chart.ColumnGroups[0].GapWidth = 120;
    shape.Chart.ColumnGroups[0].Overlap = -20;

```

Configure the layout of the bubble chart

Refer to the following example code to configure the layout of the bubble chart as per your preferences.

C#

```

//Configure bubble chart's layout
IShape shape = worksheet.Shapes.AddChart(ChartType.Bubble, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

shape.Chart.XYGroups[0].BubbleScale = 150;
shape.Chart.XYGroups[0].SizeRepresents = SizeRepresents.SizeIsArea;
shape.Chart.XYGroups[0].ShowNegativeBubbles = true;

```

Configure the layout of the doughnut chart

Refer to the following example code to configure the layout of the doughnut chart as per your preferences.

C#

```

IShape shape = worksheet.Shapes.AddChart(ChartType.Doughnut, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

shape.Chart.DoughnutGroups[0].FirstSliceAngle = 50;
shape.Chart.DoughnutGroups[0].DoughnutHoleSize = 20;

```

Dropline, HiLoLine and SeriesLine

You can use the **HasDropLines** property ([on-line documentation](#)), **HasHiLoLines** property ([on-line documentation](#)), **HasSeriesLines** property ([on-line documentation](#)), **DropLines** property ([on-line documentation](#)), **HiLoLines** property ([on-line documentation](#)), **SeriesLines** property ([on-line documentation](#)) of the **IShapeGroup** interface ([on-line documentation](#)) to configure Dropline, HiLoLine and Series lines in a chart.

Configure the drop lines of the line chart

Refer to the following example code to configure the drop lines of the line chart as per your preferences.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.Line, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.LineGroups[0].HasDropLines = true;
shape.Chart.LineGroups[0].DropLines.Format.Line.Color.RGB = Color.Red;
```

Configure the high-low lines of the line chart

Refer to the following example code to configure the high-low lines of the line chart as per your preferences.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.Line, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.LineGroups[0].HasHiLoLines = true;
shape.Chart.LineGroups[0].HiLoLines.Format.Line.Color.RGB = Color.Red;
```

Configure the series lines for column chart

Refer to the following example code to configure the column chart's series lines as per your preferences.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnStacked, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);
```



```
shape.Chart.ColumnGroups[0].HasSeriesLines = true;
shape.Chart.ColumnGroups[0].SeriesLines.Format.Line.Color.RGB = Color.Red;
```

Configure the connector lines for pie of a pie chart

Refer to the following example code to configure the connector lines for pie of a pie chart as per your preferences.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.PieOfPie, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.PieGroups[0].HasSeriesLines = true;
shape.Chart.PieGroups[0].SeriesLines.Format.Line.Color.RGB = Color.Red;
```

Up-Down Bars

You can use the **HasUpDownBars property (on-line documentation)**, **DownBars property (on-line documentation)** and **UpBars property (on-line documentation)** of the IChartGroup interface up-down bars in a chart to configure the style of the up bars and the down bars as per your preferences.

Configure the up-down bars for the line chart

Refer to the following example code to configure the up-down bars for the line chart as per your preferences.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.Line, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.LineGroups[0].HasUpDownBars = true;
shape.Chart.LineGroups[0].UpBars.Format.Fill.Color.RGB = Color.Green;
shape.Chart.LineGroups[0].DownBars.Format.Fill.Color.RGB = Color.Red;
```

Walls

A wall refers to an area or a plane which is present behind, below or beside a chart.

GcExcel .NET enables users to set up a chart as per their custom preferences by defining the thickness, fill color, line color and format of the back wall as well as the side wall, using the properties of the **IWall Interface (on-line documentation)** and the **IChart Interface (on-line documentation)**.

Refer to the following example code to configure the walls of the chart inserted in a worksheet.

C#

```
//Config back wall and side wall's format together.
IShape shape1 = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 50, 300, 300);
shape1.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);
shape1.Chart.Walls.Thickness = 20;
shape1.Chart.Walls.Format.Fill.Color.RGB = Color.Red;
shape1.Chart.Walls.Format.Line.Color.RGB = Color.Blue;

// Config back wall's format individually.
IShape shape2 = worksheet.Shapes.AddChart(ChartType.Column3D, 550, 50, 300, 300);
shape2.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);
shape2.Chart.BackWall.Thickness = 20;
shape2.Chart.BackWall.Format.Fill.Color.RGB = Color.Red;
shape2.Chart.BackWall.Format.Line.Color.RGB = Color.Blue;
```

Axis and Other Lines

Axis is one of the charting elements meant for displaying the scale for a single dimension of a plot area. In GcExcel .NET, an axis in a chart can have a title, major tick mark, minor tick mark, tick mark labels, major gridlines and minor gridlines.

There are three types of axes in charts:

1. Category axis - Displays categories generally in the horizontal axis for all types of charts. An exception to this is the bar chart, where categories are shown along the y-axis that is, the vertical axis.
2. Value axis - Displays series values in vertical axis. An exception to this is the bar chart, where series values are shown along the x-axis that is, the horizontal axis.
3. Series axis - Displays data series for 3-dimensional charts including 3-D column chart, 3-D area chart, 3-D line chart, and surface charts.

Typically, a two-dimensional chart is comprised of two axes - category axis and value axis. While the category axis is also known as horizontal axis (x-axis) and is used to represent arguments, the value axis is also known as vertical axis (y-axis) and it represents the data values for rows and columns in a worksheet. However, in a three-dimensional chart, there is one more axis apart from the horizontal and vertical axis. This axis is known as the series axis.

You can use the properties of the **IAxis Interface (on-line documentation)** to configure category axis, value axis and series axis in a chart.

Refer to the following example code to configure axis in your chart.

C#

```
//Use IAxis.CategoryType to set category axis's scale type
IShape shape1 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 50, 300,
300);

worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {new DateTime(2015, 10, 21), 10, 25, 25},
    {new DateTime(2016, 10, 25), -51, -36, 27},
    {new DateTime(2017, 12, 20), 52, -85, -30},
    {new DateTime(2018, 5, 5), 22, 65, 65},
    {new DateTime(2019, 10, 12), 23, 69, 69}
};

shape1.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);
worksheet.Range["A2:A6"].NumberFormat = "m/d/yyyy";
IAxis category_axis = shape1.Chart.Axes.Item(AxisType.Category);
```

```

category_axis.CategoryType = CategoryTypeAutomaticScale;
//Category axis's category type is automatic scale.
var categorytype = category_axis.CategoryType;
//Category axis's actual category type is time scale.
var actualcategorytype = category_axis.ActualCategoryType;

workbook.Worksheets.Add();
IWorksheet worksheet1 = workbook.Worksheets[1];
worksheet1.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
//Set Category axis and Value axis's format.
IShape shape2 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 50, 300,
300);
shape2.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis category_axis1 = shape2.Chart.Axes.Item(AxisType.Category);
IAxis value_axis = shape2.Chart.Axes.Item(AxisType.Value);
//set category axis's format.
category_axis1.Format.Line.Color.RGB = Color.Green;
category_axis1.Format.Line.Weight = 3;
category_axis1.Format.Line.Style = LineStyle.ThickBetweenThin;
//set value axis's format.
value_axis.Format.Line.Color.RGB = Color.Red;
value_axis.Format.Line.Weight = 8;
value_axis.Format.Line.Style = LineStyle.ThinThin;

//Config time scale category axis's units.
worksheet1.Range["A8:A12"].NumberFormat = "m/d/yyyy";
worksheet1.Range["A7:D12"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {new DateTime(2015, 10, 21), 10, 25, 25},
    {new DateTime(2016, 10, 25), -51, -36, 27},
    {new DateTime(2017, 12, 20), 52, -85, -30},
    {new DateTime(2018, 5, 5), 22, 65, 65},
    {new DateTime(2019, 10, 12), 23, 69, 69}
};
IShape shape3 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 450, 300,
300);
shape3.Chart.SeriesCollection.Add(worksheet1.Range["A7:D12"], RowCol.Columns, true,
true);
IAxis category_axis2 = shape3.Chart.Axes.Item(AxisType.Category);
category_axis2.MaximumScale = new DateTime(2019, 10, 1).ToOADate();
category_axis2.MinimumScale = new DateTime(2015, 10, 1).ToOADate();
category_axis2.BaseUnit = TimeUnit.Years;
category_axis2.MajorUnitScale = TimeUnit.Months;
category_axis2.MajorUnit = 4;
category_axis2.MinorUnitScale = TimeUnit.Days;
category_axis2.MinorUnit = 60;

//Config value axis's units.
IShape shape4 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 800, 300,
300);

```

```

shape4.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis category_axis3 = shape4.Chart.Axes.Item(AxisType.Category);
IAxis value_axis1 = shape4.Chart.Axes.Item(AxisType.Value);
value_axis1.MaximumScale = 150;
value_axis1.MinimumScale = 50;
value_axis1.MajorUnit = 20;
value_axis1.MinorUnit = 5;

//Set axis crosses at.
IShape shape5 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 1150, 300,
300);
shape5.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis value_axis2 = shape5.Chart.Axes.Item(AxisType.Value);
value_axis2.Crosses = AxisCrosses.Maximum;

//Set axis's scale type.
IShape shape6 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 1500, 300,
300);
shape6.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis value_axis3 = shape6.Chart.Axes.Item(AxisType.Value);
value_axis3.ScaleType = ScaleType.Logarithmic;
value_axis3.LogBase = 5;

//Set axis's tick mark.
IShape shape7 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 1850, 300,
300);
shape7.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis category_axis4 = shape7.Chart.Axes.Item(AxisType.Category);
category_axis4.Format.Line.Color.RGB = Color.Green;
category_axis4.MajorTickMark = TickMark.Inside;
category_axis4.MinorTickMark = TickMark.Cross;
category_axis4.TickMarkSpacing = 2;

```

Configure Chart Axis

In GcExcel .NET, you can configure chart axis using the following elements in your spreadsheet:

- **Axis title**
- **Gridlines**
- **Display unit label**
- **Tick labels**

Axis title

While configuring chart axis, you can set the style for the axis title as per your preferences by using the **AxisTitle property (on-line documentation)** of the **IAxis interface (on-line documentation)**.

Refer to the following example code to configure axis title's layout.

C#

```

IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},

```

```

        {"Item1", 10, 25, 25},
        {"Item2", -51, -36, 27},
        {"Item3", 52, -85, -30},
        {"Item4", 22, 65, 65},
        {"Item5", 23, 69, 69}
    };
    shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

    IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);
    IAxis value_axis = shape.Chart.Axes.Item(AxisType.Value);
    category_axis.HasTitle = true;
    category_axis.AxisTitle.Format.Fill.Color.RGB = Color.Pink;
    category_axis.AxisTitle.Text = "aaaaaaaaaa";
    category_axis.AxisTitle.Font.Size = 20;
    category_axis.AxisTitle.Font.Color.RGB = Color.Green;
    category_axis.AxisTitle.Font.Strikethrough = true;

```

Gridlines

While configuring the axis of a chart, you can also set the style of major and minor gridlines as per your choice using the **HasMajorGridlines property (on-line documentation)**, **HasMinorGridlines property (on-line documentation)**, **MajorGridlines property (on-line documentation)** and **MinorGridlines property (on-line documentation)** of the IAxis interface.

Refer to the following example code to set major and minor gridlines' style.

C#

```

IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IAxis value_axis = shape.Chart.Axes.Item(AxisType.Value);
IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);
category_axis.HasMajorGridlines = true;
category_axis.HasMinorGridlines = true;
category_axis.MajorGridlines.Format.Line.Color.RGB = Color.Red;
category_axis.MajorGridlines.Format.Line.Weight = 3;
category_axis.MinorGridlines.Format.Line.Color.RGB = Color.Green;
category_axis.MinorGridlines.Format.Line.Weight = 1;
category_axis.MinorGridlines.Format.Line.Style = LineStyle.ThickThin;

```

Display unit label

While configuring the chart axis in your worksheet, you can also set the display unit for the axis and configure its label style using the **DisplayUnit property (on-line documentation)**, **DisplayUnitLabel property (on-line documentation)** and **HasDisplayUnitLabel property (on-line documentation)** of the IAxis interface.

Refer to the following example code to set display unit for the axis and configure its label style.

C#

```

IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]

```

```

{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);
IAxis value_axis = shape.Chart.Axes.Item(AxisType.Value);
value_axis.DisplayUnit = DisplayUnit.Hundreds;
value_axis.HasDisplayUnitLabel = true;
value_axis.DisplayUnitLabel.Font.Color.RGB = Color.Green;
value_axis.DisplayUnitLabel.Font.Italic = true;
value_axis.DisplayUnitLabel.Format.Fill.Color.RGB = Color.Pink;
value_axis.DisplayUnitLabel.Format.Line.Color.RGB = Color.Red;

```

Tick labels

While configuring the axis of a chart, you can also set the position and layout of the tick-mark labels as per your choice using the **TickLabelPosition property (on-line documentation)**, **TickLabels property (on-line documentation)**, **TickLabelSpacing property (on-line documentation)**, **TickLabelSpacingIsAuto property (on-line documentation)** and **TickMarkSpacing property (on-line documentation)** of the IAxis interface.

Refer to the following example code to configure the tick mark label's position and layout.

C#

```

IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);
IAxis value_axis = shape.Chart.Axes.Item(AxisType.Value);

//tick-mark labels' fill will be green according to axis's format.
category_axis.Format.Fill.Color.RGB = Color.Green;

category_axis.TickLabelPosition = TickLabelPosition.NextToAxis;
category_axis.TickLabelSpacing = 2;
category_axis.TickLabels.Font.Color.RGB = Color.Red;
category_axis.TickLabels.Font.Italic = true;
category_axis.TickLabels.NumberFormat = "#,##0.00";
category_axis.TickLabels.Offset = 100;

```

Floor

Floor represents the floor of a three-dimensional chart. The area of a 3-D chart can be formatted using floor as the charting object.

In GcExcel .NET, you can use the properties and methods of the **IFloor interface (on-line documentation)** to set the line and fill format of the floor along with its thickness.

Refer to the following example code to configure the format of floor in a chart.

C#

```
//Configure floor's format.
IShape shape = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 50, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.Floor.Thickness = 20;
shape.Chart.Floor.Format.Fill.Color.RGB = Color.Red;
shape.Chart.Floor.Format.Line.Color.RGB = Color.Blue;
```

Data Label

GcExcel .NET allows you to insert data labels in a chart to ensure the information depicted in it can be easily interpreted and visualized. You can add data labels in a chart using the properties and methods of the **IPoint interface (on-line documentation)** and the **ISeries interface (on-line documentation)**.

Refer to the following example code to set data labels in a chart and customize the data label text.

C#

```
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};

//Set Series' all data labels and specific data label's format.
IShape shapel = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 50, 300, 300);
shapel.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);
ISeries series1 = shapel.Chart.SeriesCollection[0];
series1.HasDataLabels = true;
//set series1's all data label's format.
series1.DataLabels.Format.Fill.Color.RGB = Color.Green;
series1.DataLabels.Format.Line.Color.RGB = Color.Red;
series1.DataLabels.Format.Line.Weight = 3;
//set series1's specific data label's format.
series1.DataLabels[2].Format.Fill.Color.RGB = Color.Yellow;
series1.Points[2].DataLabel.Format.Line.Color.RGB = Color.Blue;
series1.Points[2].DataLabel.Format.Line.Weight = 5;

//Customize data label's text.
IShape shape2 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 550, 50, 300, 300);
```

```

shape2.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);
ISeries series2 = shape2.Chart.SeriesCollection[0];
series2.HasDataLabels = true;
//customize data lables' text.
series2.DataLabels.ShowCategoryName = true;
series2.DataLabels.ShowSeriesName = true;
series2.DataLabels.ShowLegendKey = true;

```

Legends

In order to enable users to quickly interpret and understand the charted data, Legends (visual charting elements) automatically appear in spreadsheets when you finish creating a chart.

Legends are also known as keys and are associated with the graphic data plotted on the chart. Usually, they are located at the right side of the chart. From a wider perspective, they facilitate end users to determine series and series points representing distinct data groups in a spreadsheet.

Typically, legends depict series names by listing and identifying the data points that belong to a particular series. Corresponding to the data, each legend entry appearing on the worksheet can be shown with the help of a legend marker along with the legend text that identifies it.

In GcExcel .NET, you can even customize the legend text, configure the position and layout of the legend, reset the font style for the legend entries, delete legend and its entries as and when you want using the properties and methods of the **ILegend interface (on-line documentation)** and the **IChart interface (on-line documentation)**.

Refer to the following example code to configure some useful legend settings in your chart.

C#

```

worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};

//Config legend's position and layout.
IShape shape = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 50, 300, 300);
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);
shape.Chart.HasLegend = true;
ILegend legend = shape.Chart.Legend;
//position.
legend.Position = LegendPosition.Left;
//font.
legend.Font.Color.RGB = Color.Red;
legend.Font.Italic = true;
//format.
legend.Format.Fill.Color.RGB = Color.Pink;
legend.Format.Line.Color.RGB = Color.Blue;

//Config legend entry's font style.
ILegendEntry legendentry = legend.LegendEntries[0];
legendentry.Font.Size = 20;
legendentry.Font.Italic = true;

```

Refer to the following example code if you want to delete the legend or a specific legend entry from your chart.

C#


```
//Delete legend.
IShape shapel = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 450, 300, 300);
shapel.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);
shapel.Chart.HasLegend = true;
ILegend legend1 = shapel.Chart.Legend;
legend1.Delete();

//Delete legend entry.
IShape shape2 = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 800, 300, 300);
shape2.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);
shape2.Chart.HasLegend = true;
ILegend legend2 = shape2.Chart.Legend;
ILegendEntry legendentry2 = legend2.LegendEntries[0];
legendentry2.Delete();
```

Use Sparkline

You can use sparklines in cells to insert graphical illustration of trends in data. Sparklines are particularly useful for analytical dashboards, presentations, business reports etc. The sparkline displays the most recent value as the rightmost data point and compares it with earlier values on a scale, allowing you to view general changes in data over time.

Using sparklines includes the following tasks.

- **Add a group of new sparklines**
- **Clear sparkline**
- **Clear sparkline groups**
- **Create a group of existing sparklines**
- **Add group of new sparklines with Date Axis**
- **Configure layout of sparkline**

Add a group of new sparklines

You can add a group of new sparklines for each row or column of data in your worksheet by first specifying the data range and then using the **Add method (on-line documentation)** of the **ISparklineGroups interface (on-line documentation)**.

Refer to the following example code to add a group of new sparklines.

C#

```
//Create workbook and access its first worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
// Add a group of new sparklines
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
```

Clear sparkline

You can remove a sparkline from your worksheet by first specifying the data range and then using the **Clear**

method (on-line documentation) of the ISparklineGroups interface.

Refer to the following example code to clear sparkline.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};

worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
// Defining data in the range
worksheet.Range["F1:H4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
// Add a group of new sparklines
worksheet.Range["J1:J4"].SparklineGroups.Add(SparkType.Line, "F1:H4");

//Clear D2 and J1 cell's sparkline.
worksheet.Range["D2, J1"].SparklineGroups.Clear();
```

Clear sparkline groups

You can remove a group of sparklines (added for a row or column) from your worksheet by specifying the data range and then using the **ClearGroups method (on-line documentation)** of the ISparklineGroups interface.

Refer to the following example code to clear sparkline groups.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
// Add a group of new sparklines
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
// Defining data in the range
worksheet.Range["F1:H4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
// Add a group of new sparklines
worksheet.Range["J1:J4"].SparklineGroups.Add(SparkType.Line, "F1:H4");

//Clear sparkline groups.
worksheet.Range["D2, J1"].SparklineGroups.ClearGroups();
```

Create a group of existing sparklines

You can create a group of existing sparklines by specifying the data range and then using the **Group() method ('Group Method' in the on-line documentation)** of the **ISparklineGroups** interface.

Refer to the following example code to create a group of existing sparklines.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};

// Add a group of new sparklines
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
// Defining data in the range
worksheet.Range["F1:H4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};

// Add a group of new sparklines
worksheet.Range["J1:J4"].SparklineGroups.Add(SparkType.Column, "F1:H4");

//Create a new group, according to Range["J2"]'s sparkline group setting.
worksheet.Range["A1:J4"].SparklineGroups.Group(worksheet.Range["J2"]);
```

Add group of new sparklines with Date Axis

You can add a group of new sparklines with date axis by first specifying the data range and then using the **DateRange property (on-line documentation)** of the **ISparklineGroup interface (on-line documentation)**.

Refer to the following example code to add group of new sparkline with date axis.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};

// Add a group of new sparklines
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
worksheet.Range["A7:C7"].Value = new object[] { new DateTime(2011, 12, 16), new
DateTime(2011, 12, 17), new DateTime(2011, 12, 18) };

//Set horizontal axis's Date range.
worksheet.Range["D1"].SparklineGroups[0].DateRange = "A7:C7";

worksheet.Range["D1"].SparklineGroups[0].Axes.Horizontal.Axis.Visible = true;
worksheet.Range["D1"].SparklineGroups[0].Axes.Horizontal.Axis.Color.Color =
Color.Green;
worksheet.Range["D1"].SparklineGroups[0].Axes.Vertical.MinScaleType =
SparkScale.SparkScaleCustom;
worksheet.Range["D1"].SparklineGroups[0].Axes.Vertical.MaxScaleType =
```

```
SparkScale.SparkScaleCustom;  
worksheet.Range["D1"].SparklineGroups[0].Axes.Vertical.CustomMinScaleValue = -2;  
worksheet.Range["D1"].SparklineGroups[0].Axes.Vertical.CustomMaxScaleValue = 8;
```

Configure layout of sparkline

You can configure the layout of the sparkline by using the properties of the `ISparklineGroup` interface.

Refer to the following example code to configure the layout of the sparkline.

C#

```
// Defining data in the range  
worksheet.Range["A1:C4"].Value = new object[,]  
{  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};  
// Adding sparkline  
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");  
// Configuring the layout  
var sparklinegroup = worksheet.Range["D1"].SparklineGroups[0];  
sparklinegroup.LineWeight = 2.5;  
sparklinegroup.Points.Markers.Color.Color = Color.Red;  
sparklinegroup.Points.Markers.Visible = true;  
sparklinegroup.SeriesColor.Color = Color.Purple;
```

Use Table

Tabular data is easy to read, interpret, visualize and manage.

GcExcel .NET supports the use of tables in worksheets by enabling users to perform different tasks on a table that help them in handling large chunks of data quickly and efficiently.

In GcExcel .NET, you can use table in the following ways:

- [Create and Delete Tables](#)
- [Modify Tables](#)
- [Apply Table Sort](#)
- [Set Table Filters](#)
- [Add and Delete Table Columns and Rows](#)
- [Apply Table Style](#)

Create and Delete Tables

In GcExcel .NET, you can create and delete tables in spreadsheets using the **Add method (on-line documentation)** of the **ITables interface (on-line documentation)** and the **Delete Method (on-line documentation)** of the **ITable Interface (on-line documentation)**, or simply transform a cell range into a table by specifying the existing data lying in a worksheet.

Refer to the following example code to create and delete tables in a worksheet.

C#

```
//Create workbook and access its first worksheet  
Workbook workbook = new Workbook();  
IWorksheet worksheet = workbook.Worksheets[0];  
//Add first table  
ITable table1 = worksheet.Tables.Add(worksheet.Range["A1:E5"], true);  
//Add second table
```

```
ITable table2 = worksheet.Tables.Add(worksheet.Range["F1:G5"], true);  
//Delete second Table  
worksheet.Tables[1].Delete();
```

Modify Tables

While working with tables in GcExcel .NET, you can configure it as per your spreadsheet requirements by modifying the table using the properties and methods of the **ITable interface (on-line documentation)**.

- **Modify table range**
- **Modify table areas**
- **Modify totals row of table column**

Modify table range

GcExcel .NET allows you to modify the table range of your worksheet using the **Resize method (on-line documentation)** of the ITable interface.

Refer to the following example code to modify table range.

```
C#  
//Modify table range  
table.Resize(worksheet.Range["B1:E4"]);
```

Modify table areas

You can modify the value of specific table areas by accessing its header range, data range and total range using the **HeaderRange property (on-line documentation)**, **DataRange property (on-line documentation)** and **TotalsRange property (on-line documentation)** of the ITable interface.

Refer to the following example code to modify table areas in your worksheet.

```
C#  
ITable table = worksheet.Tables.Add(worksheet.Range["A1:E5"], true);  
table.ShowTotals = true;  
  
//Populate table values  
worksheet.Range["A2"].Value = 3;  
worksheet.Range["A3"].Value = 4;  
worksheet.Range["A4"].Value = 2;  
worksheet.Range["A5"].Value = 1;  
worksheet.Range["B2"].Value = 32;  
worksheet.Range["B3"].Value = 41;  
worksheet.Range["B4"].Value = 12;  
worksheet.Range["B5"].Value = 16;  
worksheet.Range["C2"].Value = 3;  
worksheet.Range["C3"].Value = 4;  
worksheet.Range["C4"].Value = 15;  
worksheet.Range["C5"].Value = 18;  
  
//Table second column name set to "Age".  
worksheet.Tables[0].HeaderRange[0, 1].Value = "Age";  
  
// "Age" Column's second row's value set to 23.  
worksheet.Tables[0].DataRange[1, 1].Value = 23;  
  
// "Age" column's total row function set to average.  
worksheet.Tables[0].TotalsRange[0, 1].Formula = "=SUBTOTAL(101, [Age])";
```

Modify totals row of table column

When you need to make changes to the total row's calculation function of a specific table column, you can use the **TotalsCalculation property (on-line documentation)** of the **ITableColumn interface (on-line documentation)**.

Refer to the following example code to modify column total row's calculation function.

C#

```
worksheet.Tables.Add(worksheet.Range["A1:C5"], true);
worksheet.Tables[0].ShowTotals = true;

//Populate table values
worksheet.Range["A2"].Value = 3;
worksheet.Range["A3"].Value = 4;
worksheet.Range["A4"].Value = 2;
worksheet.Range["A5"].Value = 1;
worksheet.Range["B1"].Value = 13;
worksheet.Range["B2"].Value = 32;
worksheet.Range["B3"].Value = 41;
worksheet.Range["B4"].Value = 12;
worksheet.Range["B5"].Value = 16;
worksheet.Range["C1"].Value = 1;
worksheet.Range["C2"].Value = 3;
worksheet.Range["C3"].Value = 4;
worksheet.Range["C4"].Value = 15;
worksheet.Range["C5"].Value = 18;

//First table column's total row calculation fuction will be "=SUBTOTAL(101,[Column1])"
worksheet.Tables[0].Columns[1].TotalsCalculation = TotalsCalculation.Count;
```

Apply Table Sort

GcExcel .NET provides an option to apply sorting on a specific table in the worksheet. To accomplish this, you can use the **Sort property (on-line documentation)** of the **ITable interface (on-line documentation)**.

The **Apply ('Apply Method' in the on-line documentation)** method is used to apply the selected sort state and display the results.

Refer to the following example code to apply table sorting in a worksheet.

C#

```
// Assigning Value to the range
worksheet.Range["A2"].Value = 3;
worksheet.Range["A3"].Value = 4;
worksheet.Range["A4"].Value = 2;
worksheet.Range["A5"].Value = 1;

worksheet.Range["B2"].Value = 1;
worksheet.Range["B3"].Value = 2;
worksheet.Range["B4"].Value = 3;
worksheet.Range["B5"].Value = 4;

worksheet.Range["F2"].Value = "aaa";
worksheet.Range["F3"].Value = "bbb";
worksheet.Range["F4"].Value = "ccc";
worksheet.Range["F5"].Value = "ddd";

worksheet.Range["B2:B5"].FormatConditions.AddIconSetCondition();

//Sort by column A firstly, then by column B.
ValueSortField key1 = new ValueSortField(worksheet.Range["A1:A2"],
SortOrder.Ascending);
```

```
IconSortField key2 = new IconSortField(worksheet.Range["B1:B2"],
workbook.IconSets[IconSetType.Icon3Arrows][1], SortOrder.Descending);

table.Sort.SortFields.Add(key1);
table.Sort.SortFields.Add(key2);
table.Sort.Apply();
```

Set Table Filters

When you have a lot of data to handle, you can create as many tables on a spreadsheet as you want and apply separate filters on columns of each of the table to manage information in an effective manner.

GcExcel .NET provides users with the ability to set table filters while setting up worksheets for ensuring improved data analysis.

When applying filters on tables in worksheets created, you need to first get the table range and then use the **AutoFilter method (on-line documentation)** of the **IRange interface (on-line documentation)** to filter the table.

Refer to the following example code to set table filters in a worksheet.

C#

```
//Add Table
ITable table = worksheet.Tables.Add(worksheet.Range["A1:E5"], true);

//Populate table values
worksheet.Range["A2"].Value = 3;
worksheet.Range["A3"].Value = 4;
worksheet.Range["A4"].Value = 2;
worksheet.Range["A5"].Value = 1;

//Apply table filter
worksheet.Tables[0].Range.AutoFilter(0, ">2");
```

Add and Delete Table Columns and Rows

You can add and delete columns and rows of a table using the methods and properties of the following interfaces:

- **ITableColumns Interface (on-line documentation)** - Represents the table columns collection.
- **ITableRows Interface (on-line documentation)** - Represents the table rows collection.
- **ITableColumn Interface (on-line documentation)** - Represents an individual table column.
- **ITableRow Interface (on-line documentation)** - Represents an individual table row.

Add and delete table columns

To add and delete table columns, you can use the **Add method (on-line documentation)** of the **ITableColumns** interface and the **Delete method (on-line documentation)** of the **ITableColumn** interface respectively.

Refer to the following example code in order to add and delete table columns.

C#

```
//Create first table
ITable table1 = worksheet.Tables.Add(worksheet.Range["D3:I6"], true);

//Create second table
ITable table2 = worksheet.Tables.Add(worksheet.Range["A1:C6"], true);

//Insert a table column before first column in first table
```

```
table1.Columns.Add(0);

//Insert a table column before first column in second table
table2.Columns.Add(0);

//Delete the first table column from the first table.
worksheet.Tables[0].Columns[0].Delete();
```

Add and delete table rows

To add and delete table rows, you can use the **Add method (on-line documentation)** of the `ITableRows` interface and the **Delete method (on-line documentation)** of the `ITableRow` interface respectively.

Refer to the following example code in order to add and delete table rows.

C#

```
//insert a new row at the end of the first table.
table1.Rows.Add();

//insert a new row at the end of the second table.
table2.Rows.Add();

//Delete the second row in the second table.
table2.Rows[1].Delete();
```

Apply Table Style

In GcExcel .NET, you can create custom table style elements and apply them to your worksheet using the **ITableStyle Interface (on-line documentation)**. Also, you can format a table using any of the predefined table styles provided by GcExcel .NET.

Typically, each workbook possesses an **ITableStyle collection ('ITableStyleCollection Interface' in the on-line documentation)** that is used to store both built-in and custom table styles. If you want to insert a custom table style, you use the **Add method (on-line documentation)** of the **ITables interface (on-line documentation)**, which returns the `IStyle` object representing the corresponding table style instance.

C#

```
//Use table style name get one build in table style.
ITableStyle tableStyle = workbook.TableStyles["TableStyleLight11"];
worksheet.Tables.Add(worksheet.Range[0, 0, 2, 2], true);

//set build in table style to table.
worksheet.Tables[0].TableStyle = tableStyle;
```

Modify Table with Custom Style

In order to manage the collection of table styles in your workbook, you can modify the existing table style with your own custom table style that you have created. Each table style element represents the formatting for a particular element of the table. When you define a custom style for your table, you need to first access the existing table style element to customize table borders, set custom fill for your table, style row stripes or column stripes etc.

As a default characteristic, you will find your workbook possessing a collection of table style for you to apply formatting to tables. These default table styles are built-in table styles which represent no formatting is applied to the tables. However, when you create a custom table style, it automatically gets added to the table style collection of your workbook and can be reused as and when you require.

If you want to change the table style, you can use the `TableStyle` property. For accomplishing this task, you will

first need to use the indexer notation of `ITableStyleCollection` to set the table style instance.

In case you want to delete the applied table style, you can use the `Delete` method.

C#

```
//Add one custom table style.
ITableStyle style = workbook.TableStyles.Add("test");

//Set WholeTable element style.
style.TableStyleElements[TableStyleElementType.WholeTable].Font.Italic = true;
style.TableStyleElements[TableStyleElementType.WholeTable].Font.ThemeColor =
ThemeColor.Accent6;
style.TableStyleElements[TableStyleElementType.WholeTable].Font.Strikethrough = true;
style.TableStyleElements[TableStyleElementType.WholeTable].Borders.LineStyle =
BorderStyle.Dotted;
style.TableStyleElements[TableStyleElementType.WholeTable].Borders.ThemeColor =
ThemeColor.Accent2;
style.TableStyleElements[TableStyleElementType.WholeTable].Interior.Color =
Color.FromArgb(24, 232, 192);

//Set FirstColumnStripe element style.
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].Font.Bold = true;
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].Font.Color =
Color.FromArgb(255, 0, 0);
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].Borders.LineStyle =
BorderStyle.Thick;
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].Borders.ThemeColor =
ThemeColor.Accent5;
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].Interior.Color =
Color.FromArgb(255, 255, 0);
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].StripeSize = 2;

//Set SecondColumnStripe element style.
style.TableStyleElements[TableStyleElementType.SecondColumnStripe].Font.Color =
Color.FromArgb(255, 0, 255);
style.TableStyleElements[TableStyleElementType.SecondColumnStripe].Borders.LineStyle =
BorderStyle.DashDot;
style.TableStyleElements[TableStyleElementType.SecondColumnStripe].Borders.Color =
Color.FromArgb(42, 105, 162);
style.TableStyleElements[TableStyleElementType.SecondColumnStripe].Interior.Color =
Color.FromArgb(204, 204, 255);

ITable table = worksheet.Tables.Add(worksheet.Range["A1:C3"], true);

//Set custom table style to table.
table.TableStyle = style;

table.ShowTableStyleColumnStripes = true;
```

Modify Table Layout

In GcExcel .NET, Table Layout mode allows users to divide an area of a group into several rows and columns and then place controls into the created cells by specifying the indexes and span values for rows and columns. This functionality is similar to the one which is used while creating a table in HTML.

C#

```
ITable table = worksheet.Tables.Add(worksheet.Range["A1:B2"]);

//Show table header row.
```

```
table.ShowHeaders = true;

//To make "first row stripe" and "second row stripe" table style element's style
effective.
table.ShowTableStyleRowStripes = false;

//Hide auto filter drop down button.
table.ShowAutoFilterDropDown = false;

//To make "first column" table style element's style effective.
table.ShowTableStyleFirstColumn = true;

//Show table total row.
table.ShowTotals = true;

//To make "last column" table style element's style effective.
table.ShowTableStyleLastColumn = true;

//To make "first column stripe" and "second column stripe" table style element's style
effective.
table.ShowTableStyleColumnStripes = true;

//Unfilter table column filters, and hide auto filter drop down button.
table.ShowAutoFilter = false;
```

Use Pivot Table

GcExcel .NET provides users with the ability to display aggregated data in a spreadsheet using pivot tables - a data summarization tool that can perform complex analysis of information stored in cells for exploring, analyzing and manipulating bulk data in a worksheet.

Pivot tables not only help in categorizing data but they also help in computing the totals and average of the values in the cells as per the summary functions defined in the built-in functions list.

For incorporating and using pivot tables in worksheets, you can perform the following tasks:

- [Create Pivot Table](#)
- [Pivot Table Settings](#)
- [Apply Style To Pivot Table](#)

Create Pivot Table

GcExcel .NET allows you to create pivot tables in a spreadsheet. But, before generating a pivot table, you first need to create the pivot cache using the PivotCaches collection to stores all the pivot caches in the workbook.

After you accomplish this, you need to call the **Create method (on-line documentation)** of the **IPivotCaches interface (on-line documentation)** to create a new pivot cache. After creating pivot cache, the next step is to create the new pivot table using **CreatePivotTable ('CreatePivotTable Method' in the on-line documentation)** method of the **IPivotCache interface (on-line documentation)**.

Refer to the following example code to create pivot table in a worksheet.

C#

```
//Source data for PivotCache
object[,] sourceData = new object[,] {
    { "Order ID", "Product", "Category", "Amount", "Date",
    "Country" },
    { 1, "Carrots", "Vegetables", 4270, new DateTime(2012, 1, 6),
    "United States" },
```

```

        { 2,          "Broccoli", "Vegetables", 8239,    new DateTime(2012, 1, 7),
        "United Kingdom" },
        { 3,          "Banana",   "Fruit",    617,      new DateTime(2012, 1, 8),
        "United States" },
        { 4,          "Banana",   "Fruit",    8384,   new DateTime(2012, 1, 10),
        "Canada" },
        { 5,          "Beans",    "Vegetables", 2626,   new DateTime(2012, 1, 10),
        "Germany" },
        { 6,          "Orange",   "Fruit",    3610,   new DateTime(2012, 1, 11),
        "United States" },
        { 7,          "Broccoli", "Vegetables", 9062,   new DateTime(2012, 1, 11),
        "Australia" },
        { 8,          "Banana",   "Fruit",    6906,   new DateTime(2012, 1, 16), "New
Zealand" },
        { 9,          "Apple",    "Fruit",    2417,   new DateTime(2012, 1, 16),
        "France" },
        { 10,         "Apple",    "Fruit",    7431,   new DateTime(2012, 1, 16),
        "Canada" },
        { 11,         "Banana",   "Fruit",    8250,   new DateTime(2012, 1, 16),
        "Germany" },
        { 12,         "Broccoli", "Vegetables", 7012,   new DateTime(2012, 1, 18),
        "United States" },
        { 13,         "Carrots",  "Vegetables", 1903,   new DateTime(2012, 1, 20),
        "Germany" },
        { 14,         "Broccoli", "Vegetables", 2824,   new DateTime(2012, 1, 22),
        "Canada" },
        { 15,         "Apple",    "Fruit",    6946,   new DateTime(2012, 1, 24),
        "France" },
    };

    //Initialize the Workbook and fetch the default WorkSheet
    Workbook workbook = new Workbook();
    IWorksheet worksheet = workbook.Worksheets[0];
    // Assigning data to the range
    worksheet.Range["A1:F16"].Value = sourceData;
    // Creating pivot
    var pivotcache = workbook.PivotCaches.Create(worksheet.Range["A1:F16"]);
    var pivottable = worksheet.PivotTables.Add(pivotcache,
    worksheet.Range["L7"], "pivottable1");

```

Pivot Table Settings

You can modify the setting of the pivot table created in a spreadsheet by performing the following tasks:

- **Configure pivot table fields**
- **Add field function**
- **Filter pivot table**
- **Managing pivot field level**
- **Refresh pivot table**

Configure pivot table fields

You can configure the fields of your pivot table using the properties and methods of the **IPivotCaches interface** ([on-line documentation](#)) and **IPivotTables interface** ([on-line documentation](#)).

Refer to the following example code to configure the pivot table fields in a worksheet.

C#

```
// Configuring pivot table fields
var field_Category = pivottable.PivotFields["Category"];
field_Category.Orientation = PivotFieldOrientation.RowField;

var field_Product = pivottable.PivotFields["Product"];
field_Product.Orientation = PivotFieldOrientation.ColumnField;

var field_Amount = pivottable.PivotFields["Amount"];
field_Amount.Orientation = PivotFieldOrientation.DataField;

var field_Country = pivottable.PivotFields["Country"];
field_Country.Orientation = PivotFieldOrientation.PageField;
```

Add field function

Refer to the following example code to add field function in a pivot table.

C#

```
//Set field amount function
field_Amount.Function = ConsolidationFunction.Average;
```

Filter pivot table

Refer to the following example code to filter a pivot table.

C#

```
var field_product = pivottable.PivotFields[1];
field_product.Orientation = PivotFieldOrientation.RowField;

var field_Amount = pivottable.PivotFields[3];
field_Amount.Orientation = PivotFieldOrientation.DataField;

var field_Country = pivottable.PivotFields[5];
field_Country.Orientation = PivotFieldOrientation.PageField;

//row field filter.
field_product.PivotItems["Apple"].Visible = false;
field_product.PivotItems["Beans"].Visible = false;
field_product.PivotItems["Orange"].Visible = false;

//page filter.
field_Country.PivotItems["United States"].Visible = false;
field_Country.PivotItems["Canada"].Visible = false;
```

Managing pivot field level

Refer to the following example code to manage the field level of a pivot table.

C#

```
//product in level 1.
var field_product = pivottable.PivotFields["Product"];
field_product.Orientation = PivotFieldOrientation.RowField;

//category in level 2.
var field_category = pivottable.PivotFields["Category"];
field_category.Orientation = PivotFieldOrientation.RowField;

var field_Amount = pivottable.PivotFields[3];
field_Amount.Orientation = PivotFieldOrientation.DataField;
```

```
//category will in level 1 and product in level 2.  
field_product.Position = 1;  
field_category.Position = 0;
```

Refresh pivot table

Refer to the following example code to refresh a pivot table.

C#

```
var field_product = pivottable.PivotFields["Product"];  
field_product.Orientation = PivotFieldOrientation.RowField;  
  
var field_Amount = pivottable.PivotFields[3];  
field_Amount.Orientation = PivotFieldOrientation.DataField;  
  
//change pivot cache's source data.  
worksheet.Range["D8"].Value = 3000;  
  
//sync cache's data to pivot table.  
worksheet.PivotTables[0].Refresh();
```

Apply Style To Pivot Table

GcExcel .NET allows users to apply built-in and custom styles to the pivot table.

With the help of this feature, users will be able to save pivot tables with different styles (with respect to the pivot table layout and pivot table fields). Users can customize how their pivot table is displayed including the pivot table's orientation, page size, pivot table fields and many other characteristics as per their custom display preferences. Further, users can also refer to the topic [Export Pivot Table Styles and Format](#) in order export spreadsheets with different pivot table styles in PDF format.

Usually, when users add a pivot table to the worksheet, a default pivot table style is applied automatically. Users can modify the default style of the pivot table added to the worksheet by either copying an existing style (also called built-in style) or creating a custom pivot table style right from the scratch.

In order to apply style to the pivot table, you can refer to the following sections:

- **Apply Built-In Pivot Table Style**
- **Apply Custom Style**

Apply Built-In Pivot Table Style

You can change the default appearance of the pivot table by applying any of the built-in styles. In order to apply built-in style to the pivot table, users can either use the **Style ('Style Property' in the on-line documentation)** property or use the **TableStyle ('TableStyle Property' in the on-line documentation)** property of the **IPivotTable ('IPivotTable Interface' in the on-line documentation)** interface.

The image shared below depicts a pivot table with built-in style.

Date	(All)						
Sum of Amount	Column Labels	Banana	Beans	Broccoli	Carrots	Orange	Grand Total
Row Labels	Apple						
Fruit	\$9,848.00	\$24,157.00				\$3,610.00	\$37,615.00
Canada	\$7,431.00	\$8,384.00					\$15,815.00
France	\$2,417.00						\$2,417.00
Germany		\$8,250.00					\$8,250.00
New Zealand		\$6,906.00					\$6,906.00
United States		\$617.00				\$3,610.00	\$4,227.00
Vegetables			\$2,626.00	\$24,313.00	\$6,173.00		\$33,112.00
Australia				\$9,062.00			\$9,062.00
Germany			\$2,626.00		\$1,903.00		\$4,529.00
United Kingdom				\$8,239.00			\$8,239.00
United States				\$7,012.00	\$4,270.00		\$11,282.00
Grand Total	\$9,848.00	\$24,157.00	\$2,626.00	\$24,313.00	\$6,173.00	\$3,610.00	\$70,727.00

Refer to the following example code in order to apply built-in style to the pivot table.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Create PivotTable
object[,] sourceData = new object[,] {
{ "Order ID", "Product", "Category", "Amount", "Date", "Country" },
{ 1, "Carrots", "Vegetables", 4270, new DateTime(2018, 1, 6), "United States" },
{ 2, "Broccoli", "Vegetables", 8239, new DateTime(2018, 1, 7), "United Kingdom" },
{ 3, "Banana", "Fruit", 617, new DateTime(2018, 1, 8), "United States" },
{ 4, "Banana", "Fruit", 8384, new DateTime(2018, 1, 10), "Canada" },
{ 5, "Beans", "Vegetables", 2626, new DateTime(2018, 1, 10), "Germany" },
{ 6, "Orange", "Fruit", 3610, new DateTime(2018, 1, 11), "United States" },
{ 7, "Broccoli", "Vegetables", 9062, new DateTime(2018, 1, 11), "Australia" },
{ 8, "Banana", "Fruit", 6906, new DateTime(2018, 1, 16), "New Zealand" },
{ 9, "Apple", "Fruit", 2417, new DateTime(2018, 1, 16), "France" },
{ 10, "Apple", "Fruit", 7431, new DateTime(2018, 1, 16), "Canada" },
{ 11, "Banana", "Fruit", 8250, new DateTime(2018, 1, 16), "Germany" },
{ 12, "Broccoli", "Vegetables", 7012, new DateTime(2018, 1, 18), "United States" },
{ 13, "Carrots", "Vegetables", 1903, new DateTime(2018, 1, 20), "Germany" },
{ 14, "Broccoli", "Vegetables", 2824, new DateTime(2018, 1, 22), "Canada" },
{ 15, "Apple", "Fruit", 6946, new DateTime(2018, 1, 24), "France" },
};

worksheet.Range["A20:F33"].Value = sourceData;
worksheet.Range["A:F"].ColumnWidth = 10;

// Add pivot table
var pivotcache = workbook.PivotCaches.Create(worksheet.Range["A20:F33"]);
var pivottable = worksheet.PivotTables.Add(pivotcache, worksheet.Range["A1"], "pivottable1");

// Setting number format for a field
worksheet.Range["D21:D35"].NumberFormat = "$#,##0.00";

// Configure pivot table's fields
var field_Date = pivottable.PivotFields["Date"];
field_Date.Orientation = PivotFieldOrientation.PageField;
```

```

var field_Category = pivottable.PivotFields["Category"];
field_Category.Orientation = PivotFieldOrientation.RowField;

var field_Product = pivottable.PivotFields["Product"];
field_Product.Orientation = PivotFieldOrientation.ColumnField;

var field_Amount = pivottable.PivotFields["Amount"];
field_Amount.Orientation = PivotFieldOrientation.DataField;

field_Amount.NumberFormat = "$#,##0.00";

var field_Country = pivottable.PivotFields["Country"];
field_Country.Orientation = PivotFieldOrientation.RowField;

// Set pivot style
pivottable.TableStyle = "PivotStyleMedium20";

worksheet.PageSetup.TopMargin = 30;
worksheet.PageSetup.LeftMargin = 30;

worksheet.Range["A1:H16"].Columns.AutoFit();

// Saving workbook to PDF
workbook.Save(@"81-PivotTableBuiltInStyle.pdf", SaveFileFormat.Pdf);

```



Note: While applying built-in styles to the pivot table, it is important to note that if users apply a TableStyle whose **ShowAsAvailableTableStyle** ('**ShowAsAvailableTableStyle** Property' in the on-line documentation) property is set to **true**, then the InvalidOperationException is thrown.

Apply Custom Style

If you don't want to apply any of the built-in styles, you can also create and apply your own custom style to the pivot table. This can be done using the **Style** ('**Style** Property' in the on-line documentation) property of the **IPivotTable** ('**IPivotTable** Interface' in the on-line documentation) interface.

The image shared below depicts a pivot table with custom style.

Date (All)							
Sum of Amount	Column Labels						
Row Labels	Apple	Banana	Beans	Broccoli	Carrots	Orange	Grand Total
Fruit	\$9,848.00	\$24,157.00				\$3,610.00	\$37,615.00
Canada	\$7,431.00	\$8,384.00					\$15,815.00
France	\$2,417.00						\$2,417.00
Germany		\$8,250.00					\$8,250.00
New Zealand		\$6,906.00					\$6,906.00
United States		\$617.00				\$3,610.00	\$4,227.00
Vegetables			\$2,626.00	\$24,313.00	\$6,173.00		\$33,112.00
Australia				\$9,062.00			\$9,062.00
Germany			\$2,626.00		\$1,903.00		\$4,529.00
United Kingdom				\$8,239.00			\$8,239.00
United States				\$7,012.00	\$4,270.00		\$11,282.00
Grand Total	\$9,848.00	\$24,157.00	\$2,626.00	\$24,313.00	\$6,173.00	\$3,610.00	\$70,727.00

Refer to the following example code in order to apply custom style to the pivot table.

C#

```

// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

object[,] sourceData = new object[,] {
{ "Order ID", "Product", "Category", "Amount", "Date", "Country" },
{ 1, "Carrots", "Vegetables", 4270, new DateTime(2018, 1, 6), "United States" },
{ 2, "Broccoli", "Vegetables", 8239, new DateTime(2018, 1, 7), "United Kingdom" },
{ 3, "Banana", "Fruit", 617, new DateTime(2018, 1, 8), "United States" },
{ 4, "Banana", "Fruit", 8384, new DateTime(2018, 1, 10), "Canada" },
{ 5, "Beans", "Vegetables", 2626, new DateTime(2018, 1, 10), "Germany" },
{ 6, "Orange", "Fruit", 3610, new DateTime(2018, 1, 11), "United States" },
{ 7, "Broccoli", "Vegetables", 9062, new DateTime(2018, 1, 11), "Australia" },
{ 8, "Banana", "Fruit", 6906, new DateTime(2018, 1, 16), "New Zealand" },
{ 9, "Apple", "Fruit", 2417, new DateTime(2018, 1, 16), "France" },
{ 10, "Apple", "Fruit", 7431, new DateTime(2018, 1, 16), "Canada" },
{ 11, "Banana", "Fruit", 8250, new DateTime(2018, 1, 16), "Germany" },
{ 12, "Broccoli", "Vegetables", 7012, new DateTime(2018, 1, 18), "United States" },
{ 13, "Carrots", "Vegetables", 1903, new DateTime(2018, 1, 20), "Germany" },
{ 14, "Broccoli", "Vegetables", 2824, new DateTime(2018, 1, 22), "Canada" },
{ 15, "Apple", "Fruit", 6946, new DateTime(2018, 1, 24), "France" },
};

// Set source data
worksheet.Range["A20:F33"].Value = sourceData;
worksheet.Range["A:F"].ColumnWidth = 10;

// Add pivot table
var pivotcache = workbook.PivotCaches.Create(worksheet.Range["A20:F33"]);
var pivottable = worksheet.PivotTables.Add(pivotcache, worksheet.Range["A1"], "pivottable1");

// Setting number format for a field
worksheet.Range["D21:D35"].NumberFormat = "$#,##0.00";

// Configure pivot table's fields
var field_Date = pivottable.PivotFields["Date"];
field_Date.Orientation = PivotFieldOrientation.PageField;

var field_Category = pivottable.PivotFields["Category"];
field_Category.Orientation = PivotFieldOrientation.RowField;

var field_Product = pivottable.PivotFields["Product"];
field_Product.Orientation = PivotFieldOrientation.ColumnField;

var field_Amount = pivottable.PivotFields["Amount"];
field_Amount.Orientation = PivotFieldOrientation.DataField;
field_Amount.NumberFormat = "$#,##0.00";

var field_Country = pivottable.PivotFields["Country"];
field_Country.Orientation = PivotFieldOrientation.RowField;

// Create pivot style with name "CustomPivotstyle"
ITableStyle pivotStyle = workbook.TableStyles.Add("CustomPivotstyle");

// Set table style as pivot table style
pivotStyle.ShowAsAvailablePivotStyle = true;

pivotStyle.TableStyleElements[TableStyleElementType.PageFieldLabels].Interior.Color =
System.Drawing.Color.LightGreen;
pivotStyle.TableStyleElements[TableStyleElementType.PageFieldValues].Interior.Color =
System.Drawing.Color.LightGreen;

pivotStyle.TableStyleElements[TableStyleElementType.GrandTotalColumn].Interior.Color =
System.Drawing.Color.PowderBlue;

```



```

pivotStyle.TableStyleElements[TableStyleElementType.GrandTotalRow].Interior.Color =
System.Drawing.Color.PowderBlue;

pivotStyle.TableStyleElements[TableStyleElementType.HeaderRow].Interior.Color =
System.Drawing.Color.MistyRose;
pivotStyle.TableStyleElements[TableStyleElementType.FirstColumn].Interior.Color =
System.Drawing.Color.LightPink;

pivotStyle.TableStyleElements[TableStyleElementType.FirstRowStripe].Interior.Color =
System.Drawing.Color.SteelBlue;
pivotStyle.TableStyleElements[TableStyleElementType.SecondRowStripe].Interior.Color =
System.Drawing.Color.NavajoWhite;

// Set ShowTableStyleRowStripes as true
pivottable.ShowTableStyleRowStripes = true;

// Set pivot table style
pivottable.Style = pivotStyle;
worksheet.Range["A1:H16"].Columns.AutoFit();
worksheet.PageSetup.TopMargin = 30;
worksheet.PageSetup.LeftMargin = 30;

// Saving workbook to PDF
workbook.Save(@"82-PivotTableCustomStyle.pdf", SaveFileFormat.Pdf);

```



Note: While applying custom styles to the pivot table, it is important to note that if users apply a TableStyle whose **ShowAsAvailablePivotStyle** ('ShowAsAvailablePivotStyle Property' in the on-line documentation) property is set to false, then the InvalidOperationException is thrown.

Use Slicer

GcExcel .NET allows users to add slicer in spreadsheets in order to enable them to perform quick filtration of the data in tables and pivot tables.

Using slicer in a worksheet involves the following tasks:

- [Add Slicer in Table](#)
- [Add Slicer in Pivot Table](#)
- [Use Do Filter Operation](#)
- [Apply Slicer Style](#)

Add Slicer in Table

In GcExcel .NET, you can use slicer in a table by accessing the properties and methods of the **ISlicer interface (on-line documentation)**, **ISlicerCache interface (on-line documentation)**, and **ISlicerCaches interface (on-line documentation)**.

To add slicer in your table, you need to first invoke the **Add method (on-line documentation)** of the ISlicerCaches interface to create a new slicer cache for your table.

Refer to the following example code to add slicer in table.

C#

```

// Defining source data
object[,] sourceData = new object[,] {
    { "Order ID", "Product", "Category", "Amount", "Date",
    "Country" },
    { 1, "Carrots", "Vegetables", 4270, new DateTime(2012, 1, 6),
    "United States" },

```

```

        { 2,          "Broccoli", "Vegetables", 8239,    new DateTime(2012, 1, 7),
        "United Kingdom" },
        { 3,          "Banana",   "Fruit",      617,      new DateTime(2012, 1, 8),
        "United States" },
        { 4,          "Banana",   "Fruit",      8384,    new DateTime(2012, 1, 10),
        "Canada" },
        { 5,          "Beans",    "Vegetables", 2626,    new DateTime(2012, 1, 10),
        "Germany" },
        { 6,          "Orange",   "Fruit",      3610,    new DateTime(2012, 1, 11),
        "United States" },
        { 7,          "Broccoli", "Vegetables", 9062,    new DateTime(2012, 1, 11),
        "Australia" },
        { 8,          "Banana",   "Fruit",      6906,    new DateTime(2012, 1, 16), "New
Zealand" },
        { 9,          "Apple",    "Fruit",      2417,    new DateTime(2012, 1, 16),
        "France" },
        { 10,         "Apple",    "Fruit",      7431,    new DateTime(2012, 1, 16),
        "Canada" },
        { 11,         "Banana",   "Fruit",      8250,    new DateTime(2012, 1, 16),
        "Germany" },
        { 12,         "Broccoli", "Vegetables", 7012,    new DateTime(2012, 1, 18),
        "United States" },
        { 13,         "Carrots",  "Vegetables", 1903,    new DateTime(2012, 1, 20),
        "Germany" },
        { 14,         "Broccoli", "Vegetables", 2824,    new DateTime(2012, 1, 22),
        "Canada" },
        { 15,         "Apple",    "Fruit",      6946,    new DateTime(2012, 1, 24),
        "France" },
    };
    // Initialize the workbook and fetch the default worksheet
    Workbook workbook = new Workbook();
    IWorksheet worksheet = workbook.Worksheets[0];
    // Adding data to the table
    worksheet.Range["A1:F16"].Value = sourceData;
    ITable table = worksheet.Tables.Add(worksheet.Range["A1:F16"], true);
    ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");
    // Add slicer for table
    ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "Category",
    200, 200, 100, 200);
    ISlicer slicer2 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate2", "Category",
    100, 100, 100, 200);

```

Add Slicer in Pivot Table

In GcExcel .NET, you can use slicer to organize data in pivot table and multi pivot table by accessing the properties and methods of the **IPivotCache Interface (on-line documentation)**, **IPivotCaches Interface (on-line documentation)**, **IPivotField Interface (on-line documentation)**, **IPivotFields Interface (on-line documentation)**, **IPivotTable Interface (on-line documentation)**, **IPivotTables Interface (on-line documentation)** **IPivotItem Interface (on-line documentation)**.

To add slicer in a pivot table, you need to first invoke the **Add method (on-line documentation)** of the ISlicerCaches interface to create a new slicer cache for your pivot table.

Refer to the following example code to add slicer in a pivot table.

C#

```

// Defining source data
object[,] sourceData = new object[,] {
    { "Order ID", "Product", "Category", "Amount", "Date",
    "Country" },
    { 1,          "Carrots",  "Vegetables", 4270,    new DateTime(2012, 1, 6),

```

```

"United States" },
    { 2, "Broccoli", "Vegetables", 8239, new DateTime(2012, 1, 7),
"United Kingdom" },
    { 3, "Banana", "Fruit", 617, new DateTime(2012, 1, 8),
"United States" },
    { 4, "Banana", "Fruit", 8384, new DateTime(2012, 1, 10),
"Canada" },
    { 5, "Beans", "Vegetables", 2626, new DateTime(2012, 1, 10),
"Germany" },
    { 6, "Orange", "Fruit", 3610, new DateTime(2012, 1, 11),
"United States" },
    { 7, "Broccoli", "Vegetables", 9062, new DateTime(2012, 1, 11),
"Australia" },
    { 8, "Banana", "Fruit", 6906, new DateTime(2012, 1, 16), "New
Zealand" },
    { 9, "Apple", "Fruit", 2417, new DateTime(2012, 1, 16),
"France" },
    { 10, "Apple", "Fruit", 7431, new DateTime(2012, 1, 16),
"Canada" },
    { 11, "Banana", "Fruit", 8250, new DateTime(2012, 1, 16),
"Germany" },
    { 12, "Broccoli", "Vegetables", 7012, new DateTime(2012, 1, 18),
"United States" },
    { 13, "Carrots", "Vegetables", 1903, new DateTime(2012, 1, 20),
"Germany" },
    { 14, "Broccoli", "Vegetables", 2824, new DateTime(2012, 1, 22),
"Canada" },
    { 15, "Apple", "Fruit", 6946, new DateTime(2012, 1, 24),
"France" },
};
// Initialize the workbook and fetch the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
// Adding data to the pivot table
worksheet.Range["A1:F16"].Value = sourceData;

IPivotCache pivottable1 = workbook.PivotCaches.Create(worksheet.Range["A1:F16"]);
IPivotTable pivottable1 = worksheet.PivotTables.Add(pivottable1, worksheet.Range["K5"],
"pivottable1");
IPivotTable pivottable2 = worksheet.PivotTables.Add(pivottable1, worksheet.Range["O15"],
"pivottable2");

IPivotField field_product1 = pivottable1.PivotFields[1];
field_product1.Orientation = PivotFieldOrientation.RowField;

IPivotField field_Amount1 = pivottable1.PivotFields[3];
field_Amount1.Orientation = PivotFieldOrientation.DataField;

IPivotField field_product2 = pivottable2.PivotFields[5];
field_product2.Orientation = PivotFieldOrientation.RowField;

IPivotField field_Amount2 = pivottable2.PivotFields[2];
field_Amount2.Orientation = PivotFieldOrientation.DataField;
field_Amount2.Function = ConsolidationFunction.Count;

//Slicer just control pivot table1.
ISlicerCache cache = workbook.SlicerCaches.Add(pivottable1, "Product");
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "p1", "Product", 20,
20, 100, 200);

```

Refer to the following example code to add slicer in a multi pivot table.

C#

```
ISlicerCache cache = workbook.SlicerCaches.Add(pivottable1, "Product");
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "p1", "Product", 20,
20, 100, 200);
cache.PivotTables.AddPivotTable(pivottable2);
```

Use Do Filter Operation

You can set slicer filters to analyse bulk information in a spreadsheet quickly and efficiently.

Use slicer do-filter operation

Refer to the following example code to use slicer to perform do-filter operation.

C#

```
// Adding data to the table
worksheet.Range["A1:F16"].Value = sourceData;
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F16"], true);
ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");
// Add slicer for table
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "Category",
200, 200, 100, 200);
ISlicer slicer2 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate2", "Category",
100, 100, 100, 200);

//do filter operation.
slicer1.SlicerCache.SlicerItems["Vegetables"].Selected = false;
```

Clear slicer filter

Refer to the following example code to clear slicer filter.

C#

```
// Adding data to the table
worksheet.Range["A1:F16"].Value = sourceData;
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F16"], true);
ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");
// Add slicer for table
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "Category",
200, 200, 100, 200);
ISlicer slicer2 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate2", "Category",
100, 100, 100, 200);

//do filter operation.
slicer1.SlicerCache.SlicerItems["Vegetables"].Selected = false;

//clear filter.
slicer1.SlicerCache.ClearAllFilters();
```

Apply Slicer Style

When you create a slicer, it is mandatory to create a slicer cache first and then use the slicer cache created base on the column of the table or the pivot table.

The SlicerCaches collection in GcExcel .NET holds all the slicer caches in the workbook.

Set slicer to built-in style

You can set your slicer to built-in style by using the **Style property (on-line documentation)** of the **ISlicer interface (on-line documentation)**.

Refer to the following example code to set slicer to built-in style.

C#

```
//create slicer cache for table.
ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");

//add slicer
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "catel", "Category",
200, 200, 100, 200);

//set slicer style to build in style.
slicer1.Style = workbook.TableStyles["SlicerStyleLight1"];
```

Modify Slicer with Custom Style

In GcExcel .NET, you can define your own custom style and add it in the slicer cache to modify your slicer as per your preferences.

Refer to the following example code to see how you can modify your slicer with custom style.

C#

```
//create slicer cache for table.
ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");

//add slicer
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "catel", "Category",
200, 200, 100, 200);

ITableStyle slicerStyle = workbook.TableStyles.Add("test");
slicerStyle.ShowAsAvailableSlicerStyle = true;
slicerStyle.TableStyleElements[TableStyleElementType.WholeTable].Font.Name = "Arial";
slicerStyle.TableStyleElements[TableStyleElementType.WholeTable].Font.Bold = false;
slicerStyle.TableStyleElements[TableStyleElementType.WholeTable].Font.Italic = false;
slicerStyle.TableStyleElements[TableStyleElementType.WholeTable].Font.Color =
Color.White;
slicerStyle.TableStyleElements[TableStyleElementType.WholeTable].Borders.Color =
Color.Red;
slicerStyle.TableStyleElements[TableStyleElementType.WholeTable].Interior.Color =
Color.Green;

slicer1.Style = slicerStyle;
```

Modify Table Layout for Slicer Style

You can modify the table layout for the slicer style applied in your spreadsheet by modifying some settings including the **RowHeight property (on-line documentation)** and **DisplayHeader property (on-line documentation)** of the **ISlicer interface (on-line documentation)**.

Refer to the following example code to modify table layout for slicer style.

C#

```
//create slicer cache for table.
```

```
ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");

//add slicer
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "catel", "Category",
200, 200, 100, 200);

slicer1.NumberOfColumns = 2;
//slicer1.ColumnWidth = 10;
slicer1.RowHeight = 50;
slicer1.DisplayHeader = false;
```

Manage File Operations

GcExcel .NET allows users to export (save) data from a spreadsheet into several different file types (.xlsx, .csv, .pdf and .json files) and import data (open) files from several different file types (.xlsx, .csv and .json files) into GcExcel .NET. Using code, you can save the whole component, a particular sheet, or data from a particular range of cells to several different file types or streams.

Refer to the following procedures to handle file operations for a range of file types in GcExcel.NET:

- [Import and Export .xlsx Document](#)
- [Export to a PDF File](#)
- [Import and Export CSV File](#)
- [Import and Export CSV File with Delimiters](#)
- [Import and Export JSON Stream](#)
- [Import and Export Macros](#)
- [Import and Export OLE Objects](#)

Import and Export .xlsx Document

This section summarizes how GcExcel .NET handles the spreadsheet documents(.xlsx files).

When you create a workbook using GcExcel .NET and save it, you automatically export it to an external location or folder. When bringing an Excel file into GcExcel .NET (importing a file or opening a file) and when saving GcExcel .NET files to an Excel format (exporting), most of the data can be imported or exported successfully. The intention of providing the support for import and export capability is to enable users to handle as much of the data and formatting of a spreadsheet as possible.

GcExcel .NET also provides support for preserving the Japanese Ruby characters while executing the import and export operations on an Excel file. Also, users can adjust cells containing Japanese Ruby characters with utmost accuracy after performing other spreadsheet tasks like Insert, Delete, Copy, Cut, Merge, Clear, Sort operations etc.

Working With Import Flags

While opening a workbook, GcExcel .NET also provides you with several open options that can be used during the import operation.

The **ImportFlags ('ImportFlags Enumeration' in the on-line documentation)** enumeration allows users to import the workbook with the specified open options (a total of ten options are available: NoFlag, Data and Formulas, Table, mergeArea, Style, ConditionalFormatting, DataValidation, PivotTable and Shapes) as described in the table shared below.

Import Flag Option	Description
NoFlag ('NoFlag Field' in the on-line documentation)	Refers to "No option". This option is used when you don't want to put any import flag while opening the Excel file. This means that all the data in the worksheet will be imported as it is.
Data ('Data Field' in the on-line documentation)	Refers to "Read the Data". This option is used when you want to import only the data in the worksheet while opening the Excel file.
Formulas ('Formulas Field' in the on-line documentation)	Refers to "Read the Data and Formulas". This option is used when you want to import both the data and the formulas in the worksheet while opening the Excel file.
Table ('Table Field' in the on-line documentation)	Refers to "Read the Tables". This option is used when you want to import only the tables in the worksheet while opening the Excel file.
MergeArea ('MergeArea Field' in the on-line documentation)	Refers to "Read the Merge Cells". This option is used when you want to import only the merged cells or spanned cells in the worksheet while opening the Excel file.
Style ('Style Field' in the on-line documentation)	Refers to "Read the Styles". This option is used when you want to import only the styles applied to the cells in the worksheet while opening the Excel file.
ConditionalFormatting	Refers to "Read the Conditional Formatting". This option is used when

('ConditionalFormatting Field' in the on-line documentation)	you want to import only the conditional formatting rule applied to the worksheet while opening the Excel file.
DataValidation ('DataValidation Field' in the on-line documentation)	Refers to "Read the Data Validation". This option is used when you want to import only the data validation rule applied to the worksheet while opening the Excel file.
PivotTable ('PivotTable Field' in the on-line documentation)	Refers to "Read the Pivot Tables". This option is used when you want to import only the pivot tables in the worksheet while opening the Excel file.
Shapes ('Shapes Field' in the on-line documentation)	Refers to "Read all the Shapes". This option is used when you want to import only the shapes embedded in the worksheet while opening the Excel file.

The **DoNotRecalculateAfterOpened ('DoNotRecalculateAfterOpened Property' in the on-line documentation)** property of the **XlsxOpenOptions ('XlsxOpenOptions Class' in the on-line documentation)** class allows users to set a boolean value (True or False) which specifies whether or not the formulas will be recalculated when the file is being imported.

Refer to the following example code in order to import and export .xlsx document.

C#

```
// Create workbook and access its first worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];

// Assigning value to range
worksheet.Range["A3"].Value = 5;
worksheet.Range["A2"].Value = 5;
worksheet.Range["A1"].Value = 5;
worksheet.Range["B1"].Value = 5;

// Exporting .xlsx document
workbook.Save(@"savingfile.xlsx", SaveFileFormat.Xlsx);

// Exporting .xlsx document while setting password
XlsxSaveOptions options = new XlsxSaveOptions();
options.Password = "Pwd";
workbook.Save(@"savingfile.xlsx", options);

// Importing .xlsx document
workbook.Open(@"Source.xlsx", OpenFileFormat.Xlsx);

// Importing .xlsx document with Open options

// Import only data from .xlsx document.
XlsxOpenOptions options = new XlsxOpenOptions();
options.ImportFlags = ImportFlags.Data;
workbook.Open(@"Source.xlsx", options);

// Don't recalculate after opened.
XlsxOpenOptions options1 = new XlsxOpenOptions();
options1.DoNotRecalculateAfterOpened = true;
workbook.Open(@"Source.xlsx", options1);
```

Export to a PDF File

GcExcel .NET allows users to save all visible spreadsheets in a workbook to a Portable Document File (PDF) by using the **Save() ('Save Method' in the on-line documentation)** method of the **IWorkbook ('IWorkbook Interface' in the on-line documentation)** interface. Each worksheet in a workbook is saved to a new page in the PDF file.

However, if users want to export only the current sheet (active sheet) to PDF format, they can use the **Save()** ('**Save Method**' in the on-line documentation) method of the **IWorksheet** ('**IWorksheet Interface**' in the on-line documentation) interface.

Refer to the following example code to export a spreadsheet to a PDF file.

C#

```
//create workbook and add two sheets.
Workbook workbook = new Workbook();
IWorksheet sheet1 = workbook.Worksheets[0];
IWorksheet sheet2 = workbook.Worksheets.Add();

//export workbook to pdf file, the exported file has two pages.
workbook.Save(@"D:\workbook.pdf", SaveFileFormat.Pdf);

//just export a particular sheet to pdf file
sheet1.Save(@"D:\sheet1.pdf", SaveFileFormat.Pdf);
```

While executing the export operation, you can configure fonts, set style and specify the page setup options in order to customize the PDF as per your preferences. Refer to the following topics for more details:

- [Configure Fonts and Set Style](#)
- [Export Pivot Table Styles And Format](#)
- [Export Shapes](#)
- [Export Vertical Text](#)
- [Shrink To Fit With Text Wrap](#)
- [Control Pagination](#)
- [Working With Page Setup](#)



Note: The Export to PDF feature doesn't support charts and slicers while saving a spreadsheet into PDF format. Besides this, exporting picture settings (such as LineFormat, FillFormat, Brightness, Contrast, Watermark Color Type and black and white pictures in emf format) to PDF are also not supported.

Configure Fonts and Set Style

GcExcel .NET allows users to configure fonts and set style while saving their worksheets into the PDF format.

Before performing the export operation, users need to ensure that they set the **FontsFolderPath** ('**FontsFolderPath Property**' in the on-line documentation) property of the **Workbook** ('**Workbook Class**' in the on-line documentation) class in order to specify the font that should be used while saving the PDF.

If the folder path to the font is not specified and the user is working on Windows OS, the path "C:\Windows\Fonts" will be used by default. However, if the folder path to the font is not specified and the user is working on any other operating system, it is mandatory that the user set the font folder path and copy the used font files to the folder "C:\Windows\Fonts".

You can use the **GetUsedFonts()** ('**GetUsedFonts Method**' in the on-line documentation) method of the **IWorkbook** ('**IWorkbook Interface**' in the on-line documentation) interface in order to get the collection of all the fonts that are used in the workbook.

While saving PDF, GcExcel .NET uses the fonts specified in the `Workbook.FontsFolderPath` in order to render the PDF. However, if the used font doesn't exist, it will make use of some fallback fonts. In case, fallback fonts don't exist in the file, GcExcel .NET will throw the exception : "There is no available fonts. Please set a valid path to the `FontsFolderPath` property of the `Workbook`!"

Refer to the following example code to see how you can configure fonts and set style while saving to a PDF.

C#

```
//create workbook and add two sheets.
Workbook workbook = new Workbook();
IWorksheet sheet1 = workbook.Worksheets[0];
IWorksheet sheet2 = workbook.Worksheets.Add();
```

```
//set style.
sheet1.Range["A1"].Value = "Sheet1";
sheet1.Range["A1"].Font.Name = "Wide Latin";
sheet1.Range["A1"].Font.Color = Color.Red;
sheet1.Range["A1"].Interior.Color = Color.Green;

//create a table in sheet1.
sheet1.Tables.Add(sheet1.Range["C1:E5"], true);

sheet2.Range["A1"].Value = "Sheet2";

//specify font path.
Workbook.FontsFolderPath = @"D:\Fonts";

//get the used fonts list in workbook, the list are:"Wide Latin", "Calibri"
var fonts = workbook.GetUsedFonts();

//export workbook to pdf file, the exported file has two pages.
workbook.Save(@"D:\workbook.pdf", SaveFileFormat.Pdf);

//just export sheet1 to pdf file.
sheet1.Save(@"D:\sheet1.pdf", SaveFileFormat.Pdf);
```



Note: The Export to PDF feature doesn't support the following styles:

- a) Usage of Double, Single Accounting, Double Accounting underline, Superscript effect, Subscript effect.
- b) Alignment Preferences like Center across selection, Fill alignment, Orientation, Text reading order etc.
- c) Rectangle Gradient Fill is not supported.
- d) Pivot Table Style is not supported for PDF rendering.

Export Pivot Table Styles And Format

GcExcel .NET allows users to save Excel files containing distinct pivot table styles and formats into a PDF file.

With extensive support for exporting pivot table styles and format, users can customize how the pivot table is displayed in the PDF format. This includes saving Excel files with custom pivot table layout, pivot table fields, orientation, page size etc. into PDF files as per your specific preferences.

The **Style ('Style Property' in the on-line documentation)** property of the **IPivotTable ('IPivotTable Interface' in the on-line documentation)** interface can be used to get or set the pivot table style. While exporting PDFs with pivot table styles in GcExcel .NET, the following properties can be used:

Property

IPivotTable.ShowTableStyleColumnHeaders ('ShowTableStyleColumnHeaders Property' in the on-line documentation)

IPivotTable.ShowTableStyleRowHeaders ('ShowTableStyleRowHeaders Property' in the on-line documentation)

IPivotTable.ShowTableStyleColumnStripes ('ShowTableStyleColumnStripes Property' in the on-line documentation)

IPivotTable.ShowTableStyleRowStripes ('ShowTableStyleRowStripes Property' in the on-line documentation)

IPivotTable.ShowTableStyleLastColumn ('ShowTableStyleLastColumn Property' in

Description

This property can be used to get or set whether the column headers should be displayed in the Pivot table.

This property can be used to get or set whether the row headers should be displayed in the Pivot table.

This property can be used to get or set whether the banded columns in which even columns are formatted differently from odd columns.

This property can be used to get or set whether the banded rows in which even row are formatted differently from odd rows.

This property can be used to get or set whether to display the grand total columns style.

the on-line documentation)

ITableStyle.ShowAsAvailablePivotStyle
(**'ShowAsAvailablePivotStyle Property'** in
the on-line documentation)

This property can be used to get or set whether the specified style is shown as available in the pivot styles gallery.

IPivotField.NumberFormat (**'NumberFormat Property'** in the on-line documentation)

This property can be used to get or set the current field's number format string.

Using Code

Refer to the following example code in order to export Excel files with pivot table styles and format.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Create PivotTable
object[,] sourceData = new object[,] {
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Carrots", "Vegetables", 4270, new DateTime(2012, 1, 6), "United States" },
    { 2, "Broccoli", "Vegetables", 8239, new DateTime(2012, 1, 7), "United Kingdom" },
    { 3, "Banana", "Fruit", 617, new DateTime(2012, 1, 8), "United States" },
    { 4, "Banana", "Fruit", 8384, new DateTime(2012, 1, 10), "Canada" },
    { 5, "Beans", "Vegetables", 2626, new DateTime(2012, 1, 10), "Germany" },
    { 6, "Orange", "Fruit", 3610, new DateTime(2012, 1, 11), "United States" },
    { 7, "Broccoli", "Vegetables", 9062, new DateTime(2012, 1, 11), "Australia" },
    { 8, "Banana", "Fruit", 6906, new DateTime(2012, 1, 16), "New Zealand" },
    { 9, "Apple", "Fruit", 2417, new DateTime(2012, 1, 16), "France" },
    { 10, "Apple", "Fruit", 7431, new DateTime(2012, 1, 16), "Canada" },
    { 11, "Banana", "Fruit", 8250, new DateTime(2012, 1, 16), "Germany" },
    { 12, "Broccoli", "Vegetables", 7012, new DateTime(2012, 1, 18), "United States" },
    { 13, "Carrots", "Vegetables", 1903, new DateTime(2012, 1, 20), "Germany" },
    { 14, "Broccoli", "Vegetables", 2824, new DateTime(2012, 1, 22), "Canada" },
    { 15, "Apple", "Fruit", 6946, new DateTime(2012, 1, 24), "France" },
};

worksheet.Range["A1:F16"].Value = sourceData;
var pivotcache = workbook.PivotCaches.Create(worksheet.Range["A1:F16"]);
var pivottable = worksheet.PivotTables.Add(pivotcache, worksheet.Range["H5"],
"pivottable1");

// Create PivotTable style
ITableStyle style = workbook.TableStyles.Add("pivotStyle");

// Set the table style as a pivot table style
style.ShowAsAvailablePivotStyle = true;
style.TableStyleElements[TableStyleElementType.WholeTable].Borders.LineStyle =
BorderLineStyle.DashDotDot;
style.TableStyleElements[TableStyleElementType.WholeTable].Borders.Color =
Color.FromArgb(204, 153, 255);
style.TableStyleElements[TableStyleElementType.WholeTable].Interior.Color =
Color.FromArgb(169, 208, 142);
style.TableStyleElements[TableStyleElementType.WholeTable].Font.Italic = true;
style.TableStyleElements[TableStyleElementType.WholeTable].Font.ThemeColor =
ThemeColor.Accent2;

// Apply the style to current pivot table
pivottable.Style = style;
```

```
// Configure pivot table settings for columns and rows
pivottable.ShowTableStyleColumnHeaders = true;
pivottable.ShowTableStyleRowHeaders = true;
pivottable.ShowTableStyleColumnStripes = true;
pivottable.ShowTableStyleRowStripes = true;
pivottable.ShowTableStyleLastColumn = true;

// Add pivot field and set number format code

var field_product = pivottable.PivotFields[1];
field_product.Orientation = PivotFieldOrientation.RowField;
var field_Amount = pivottable.PivotFields[3];
field_Amount.Orientation = PivotFieldOrientation.DataField;

// Set number format code
field_Amount.NumberFormat = "#,##0";

// Saving workbook to PDF
workbook.Save(@"PivotTableStyleAndNumberFormat.pdf", SaveFileFormat.Pdf);
```

Export Shapes

GcExcel .NET provides extensive support for loading, saving, printing and exporting Excel files comprising shapes and other drawing objects embedded in the worksheets.

The **IsPrintable ('IsPrintable Property' in the on-line documentation)** property of the **IShape ('IShape Interface' in the on-line documentation)** interface can be used to get or set whether the object will be printed in the PDF document. By default, this value is TRUE and hence the shapes embedded in the Excel files are printed. In case you do not want to export shapes to the PDF files, this value must be set to FALSE.

The Export Shapes to PDF feature allows users to print and export different types of shapes such as callouts, lines, rectangles, basic shapes, block arrows, flowcharts, equation shapes, stars and banners etc. This feature is useful especially when the following scenarios are encountered while working with spreadsheets:

- When users have Excel files with graphs, reports and dashboards containing various shapes that they want to export to a PDF file.
- With the help of this feature, users can export spreadsheets that contain preset shapes, basic shapes, custom shapes and grouped shapes with different operations like rotation, flipping, connector arrows and text etc. into a PDF file.
- When users need to export Excel template files and spreadsheets containing shapes with different types of fills (like Solid fill, Gradient fill etc.) while saving to a PDF file.

Using Code

Refer to the following example code in order to export shapes to PDF.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Adding Shapes
IShape ShapeBegin = worksheet.Shapes.AddShape(AutoShapeType.CloudCallout, 1, 1, 100, 100);
IShape EndBegin = worksheet.Shapes.AddShape(AutoShapeType.Wave, 200, 200, 100, 100);

// Adding Connector Shape
```

```
IShape ConnectorShape = worksheet.Shapes.AddConnector(ConnectorType.Straight, 1, 1, 101, 101);

// Connect shapes by connector shape
ConnectorShape.ConnectorFormat.BeginConnect(ShapeBegin, 3);
ConnectorShape.ConnectorFormat.EndConnect(EndBegin, 0);

// Get second shape in current worksheet(here it's a connector shape) and do not print it(default value is true)
worksheet.Shapes[2].IsPrintable = false;

// Saving workbook to PDF
workbook.Save(@"ExportingShapesToPDF.pdf", SaveFileFormat.Pdf);
```



Note: While exporting Excel files containing shapes into the PDF format, some of the exported shapes including the shapes with rectangular gradient fill and path gradient fill; shapes with multiple lines and gradient lines; shape effects like text distribution etc. may not look exactly the same as in Excel.

Export Vertical Text

GcExcel .NET allows users to export Excel files with vertical text to PDF without any issues.

While saving an Excel file with vertical text correctly to a PDF file, the following properties can be used -

- `IRange.Orientation` - The **Orientation ('Orientation Property' in the on-line documentation)** property of the **IRange ('IRange Interface' in the on-line documentation)** interface sets the orientation of the text.
- `IRange.Font.Name` - Sets the specific font name using the **Font ('Font Property' in the on-line documentation)** property of the **IRange ('IRange Interface' in the on-line documentation)** interface. If the font name starts with "@", each double-byte character in the text is rotated to 90 degrees.

Refer to the following example code in order to export Vertical Text to PDF.

C#

```
// Create workbook and a worksheet.
Workbook workbook = new Workbook();
IWorksheet sheet = workbook.Worksheets[0];

// Specify the font name
sheet.Range["A1"].Font.Name = "@Meiryo";

// Set orientation and wrap text
sheet.Range["A1"].Orientation = -90;
sheet.Range["A1"].WrapText = true;

// Set value and configure horizontal and vertical alignment
sheet.Range["A1"].Value = "日本列島で使用されてきた言語である。GrapeCity";
sheet.Range["A1"].HorizontalAlignment = HorizontalAlignment.Right;
sheet.Range["A1"].VerticalAlignment = VerticalAlignment.Top;

// Set column width and row height

sheet.Range["A1"].ColumnWidth = 27;
sheet.Range["A1"].RowHeight = 190;

// Export the worksheet with vertical text ("sheet") to pdf file.
sheet.Save(@"D:\sheet.pdf", SaveFileFormat.Pdf);
```



Note: The following limitations must be kept in mind while exporting Excel files with vertical text to PDF -

- The orientation can only be set to 0, 90, -90 and 255. Other values will be treated as 0 while rendering the PDF file.
- If the font name starts with "@" and the orientation is 255, GcExcel will ignore the "@".

Shrink To Fit With Text Wrap

GcExcel .NET enables users to implement the shrink to fit feature in a cell along with the wrapped text. The Shrink to Fit feature automatically reduces the font size of the text so that it fits inside the cells of the spreadsheet without wrapping.

Advantage of Using Shrink To Fit Feature

The Shrink to Fit feature implemented with wrapped text is useful especially when you need to deal with spreadsheets possessing tightly constrained layouts with vertical spaces and wrapped text. Also, this feature can be used when users don't want to opt for Auto fit row height and column width option to adjust the column width and row height as per their preferred worksheet layout.

The following points should be kept in mind while working with the shrink to fit feature :

- If you're exporting your Excel files to a pdf file or stream, the **PdfSaveOptions ('PdfSaveOptions Class' in the on-line documentation)** class can be used to configure the save settings.
- In order to get or set the settings about enabling the shrink to fit feature on the wrapped text, you can use the the **ShrinkToFitSettings ('ShrinkToFitSettings Property' in the on-line documentation)** property of the **PdfSaveOptions** class.
- The **CanShrinkToFitWrappedText ('CanShrinkToFitWrappedText Property' in the on-line documentation)** property of the **IShrinkToFitSettings ('IShrinkToFitSettings Interface' in the on-line documentation)** interface can be used to get or set whether to apply shrink to fit feature on the wrapped text. If the value is true, the font size of the wrapped text may be reduced so that the wrapped text can be fully displayed.
- The **MinimumFont ('MinimumFont Property' in the on-line documentation)** property of the **IShrinkToFitSettings** interface can be used to get or set the minimum font size while enabling the shrink to fit feature.
- The **Ellipsis ('Ellipsis Property' in the on-line documentation)** property of the **IShrinkToFitSettings** interface can be used to get or set the omitted string if the wrapped text is not fully displayed. This can be used with the **MinimumFont** property.

Using Code

Refer to the following example code to allow users to use the shrink to fit feature with text wrap.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Configure page settings
worksheet.PageSetup.PrintGridlines = true;
worksheet.Range["A1"].RowHeightInPixel = 10;
worksheet.Range["A1"].ColumnWidthInPixel = 70;
worksheet.Range["A1"].WrapText = true;
worksheet.Range["A1"].ShrinkToFit = true;
worksheet.Range["A1"].Value = "GrapeCity Documents For Excel";

// Setting PdfSaveOptions
PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
pdfSaveOptions.ShrinkToFitSettings.CanShrinkToFitWrappedText = true;
```

```
pdfSaveOptions.ShrinkToFitSettings.MinimumFont = 12;
pdfSaveOptions.ShrinkToFitSettings.Ellipsis = "~";

// Saving the workbook to PDF
workbook.Save("ShrinkToFitWrappedText.pdf", pdfSaveOptions);
```

Control Pagination

GcExcel .NET enables users to manage pagination while exporting to a PDF file.

The pagination settings are used to control how the data lying in the worksheets breaks across the pages in the PDF file. The **PrintManager ('PrintManager Class' in the on-line documentation)** class contains the properties and methods that help users in handling custom pagination requirements and preferences.

The custom pagination process works in four basic steps as described below.

- Step 1 - Create an instance of the **PrintManager ('PrintManager Class' in the on-line documentation)** class
- Step 2 - Get the default pagination information of the workbook using the **Paginate() ('Paginate Method' in the on-line documentation)** method.
- Step 3 - Adjust pagination using different properties and methods of the PrintManager class.
- Step 4 - Save the PDF file by using the **SavePDF() ('SavePDF Method' in the on-line documentation)** method.

While configuring the pagination options for a workbook containing multiple worksheets, users have complete control over the flow of content (both textual and graphic) across the pages in the PDF file. Further, users can customize the PDF file by adjusting the automatic page breaks while adding or deleting the pages, and modifying the printing information on each page of the PDF file. Also, users can keep some rows together in a page; save the content from more than one worksheet to a single PDF; display custom cell ranges inside the exported PDF file; configure custom page settings (like page number, page count, page content, row headers, column headers, title columns, tail columns, page margins, page header, page footer, paper width, paper height etc.); save different header information on different pages; save the last page of the PDF file without any headers, export custom page information and export only some specific pages (or worksheets) in the PDF file.

For more information on handling pagination while working with spreadsheets, refer to the following topics:

- [Render Excel Range Inside PDF](#)
- [Export Multiple Sheets To One Page](#)
- [Keep Rows Together Over Page Breaks](#)
- [Delete Blank Pages From Middle](#)
- [Export Different Headers On Different Pages](#)
- [Export Last Page Without Headers](#)
- [Export Custom Page Information](#)
- [Export Specific Pages To PDF](#)
- [Save Multiple Workbooks to Single PDF](#)

Render Excel Range Inside PDF

GcExcel .NET enables users to render Excel cell ranges inside PDF.

This feature is useful especially when you're dealing with bulk data in the spreadsheets and you want to render only specific Excel range inside an existing PDF file. For instance - let's say you have a worksheet containing large amounts of sales data with fields such as "Number of Products Sold", "Area Sales Manager", "Region" etc. but you want to export only a chunk of useful data (like only "Number of Products Sold" and "Region") at some location in a PDF file and not all the data (you don't want to include the "Area Sales Manager" information). In this scenario, the "Render Excel Range Inside PDF" feature can be used to select some specific ranges in the worksheet and render them to specific location in a PDF file to generate full PDF reports.

In order to render Excel range inside the PDF file, you need to first create an instance of the **PrintManager**

(**'PrintManager Class' in the on-line documentation**) class and then use the **Draw()** (**'Draw Method' in the on-line documentation**) method to render the Excel range on a PDF page at a location. In case, you want to add some extra information in your PDF file (data which is not present in your Excel file), you can use the **AppendPage()** (**'AppendPage Method' in the on-line documentation**) method of the PrintManager class after configuring all the pagination settings. Finally, call the **UpdatePageNumberAndPageSettings()** (**'UpdatePageNumberAndPageSettings Method' in the on-line documentation**) method in order to update the indexes of the page number and the page settings for each page. When everything is done, simply save your PDF file using the **SavePDF()** (**'SavePDF Method' in the on-line documentation**) method.



Note: In order to render Excel cell ranges inside PDF, you should have a valid license for GrapeCity Documents for PDF.

Refer to the following example code to allow users to render Excel ranges inside the PDF file .

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Set values
worksheet.Range["A4:C4"].Value = new string[]
{ "Device", "Quantity", "Unit Price" };
worksheet.Range["A5:C8"].Value = new object[,]
{
    { "T540p", 12, 9850 },
    { "T570", 5, 7460 },
    { "Y460", 6, 5400 },
    { "Y460F", 8, 6240 }
};

// Set styles
worksheet.Range["A4:C4"].Font.Bold = true;
worksheet.Range["A4:C4"].Font.Color = Color.White;
worksheet.Range["A4:C4"].Interior.Color = Color.LightBlue;
worksheet.Range["A5:C8"].Borders[BordersIndex.InsideHorizontal].Color =
Color.Orange;
worksheet.Range["A5:C8"].Borders[BordersIndex.InsideHorizontal].LineStyle =
BorderLineStyle.DashDot;

/* NOTE: To use this feature, you should have a valid license
for GrapeCity Documents for PDF.*/

// Create a PDF document
GcPdfDocument doc = new GcPdfDocument();
Page page = doc.NewPage();
GcPdfGraphics g = page.Graphics;

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Draw the Range "A4:C8" to the specified location on the page
printManager.Draw(page, new PointF(30, 100), worksheet.Range["A4:C8"]);

// Save the modified pages into PDF file
doc.Save(@"RenderExcelRangesInsidePDFBasic.pdf");
```

Refer to the following example code to allow users to render Excel ranges inside the PDF file along with some custom textual information at runtime to the specified location on the page.

C#


```
// Create a PDF file stream
FileStream outputStream =
new FileStream("RenderExcelRangesInsidePDFAdvance.pdf", FileMode.Create);

// Create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();
Stream fileStream =
GetResourceStream("xlsx\\FinancialReport.xlsx");
workbook.Open(fileStream);
IWorksheet worksheet = workbook.Worksheets[0];

/* NOTE: To use this feature, you should have a valid license
   for GrapeCity Documents for PDF.*/

// Create a PDF document
Pdf.GcPdfDocument doc = new Pdf.GcPdfDocument();
doc.Load(GetResourceStream("Acme-Financial Report 2018.pdf"));

// Create an instance of the PrintManager class
Excel.PrintManager printManager = new Excel.PrintManager();

// Draw the contents of the sheet3 to the fourth page
IRange printArea1 = workbook.Worksheets[2].Range["A3:C24"];
SizeF size1 = printManager.GetSize(printArea1);
RectangleF position1 =
doc.FindText(new GrapeCity.Documents.Pdf.FindTextParams(
"Proposition enhancements are", true, true),
new GrapeCity.Documents.Common.OutputRange(4, 4))[0].Bounds.ToRect();
printManager.Draw(doc.Pages[3],
new RectangleF(position1.X + position1.Width +
70, position1.Y, size1.Width, size1.Height), printArea1);

// Draw the contents of the sheet1 to the fifth page
IRange printArea2 = workbook.Worksheets[0].Range["A4:E29"];
SizeF size2 = printManager.GetSize(printArea2);
RectangleF position2 =
doc.FindText(new GrapeCity.Documents.Pdf.FindTextParams(
"expenditure, an improvement in working", true, true),
new GrapeCity.Documents.Common.OutputRange(5, 5))[0].Bounds.ToRect();
printManager.Draw(doc.Pages[4],
new RectangleF(position2.X, position2.Y +
position2.Height + 20, size2.Width, size2.Height), printArea2);

// Draw the contents of the sheet2 to the sixth page
IRange printArea3 = workbook.Worksheets[1].Range["A2:E28"];
SizeF size3 = printManager.GetSize(printArea3);
RectangleF position3 =
doc.FindText(new GrapeCity.Documents.Pdf.FindTextParams(
"company will be able to continue", true, true),
new GrapeCity.Documents.Common.OutputRange(6, 6))[0].Bounds.ToRect();
printManager.Draw(doc.Pages[5],
new RectangleF(position3.X, position3.Y +
position3.Height + 20, doc.Pages[5].Size.Width -
position3.X * 2 - 10, size3.Height), printArea3);

// Save the modified pages into PDF file
doc.Save(outputStream);

// Close the PDF stream
outputStream.Close();
```

```

}

static Stream GetResourceStream(string resourcePath)
{
    string resource = "RenderExcelRangesInsideAPDF.Resource." +
        resourcePath.Replace("\\", ".");
    var assembly = typeof(Program).GetTypeInfo().Assembly;
    return assembly.GetManifestResourceStream(resource);
}

```

Export Multiple Sheets To One Page

GcExcel .NET enables users to export multiple worksheets to a single page in the PDF file .

This feature is useful especially when you want to analyse all the crucial data at one place in order to facilitate the sharing, manipulation and printing of data in an efficient way. For instance - let's say you have a workbook with multiple worksheets wherein you want to export the content of some worksheets (containing similar type of data) so that all the related data appears on the same page and is saved into a specific page in the PDF file. In this scenario, you can use this feature to export and print data from more than one worksheet to a single page in the PDF file as per your custom requirements and preferences.

In order to export multiple worksheets into a single page of the PDF file, users need to create an instance of the **PrintManager ('PrintManager Class' in the on-line documentation)** class, get the default pagination settings of the workbook using the **Paginate() ('Paginate Method' in the on-line documentation)** method, use the **Draw() ('Draw Method' in the on-line documentation)** method of the PrintManager class and finally save the PDF file using the **SavePDF() ('SavePDF Method' in the on-line documentation)** method.



Note: In order to export multiple sheets to one page, you should have a valid license for GrapeCity Documents for PDF.

Refer to the following example code to allow users to export multiple worksheets to a single page in the PDF file.

C#

```

// Initialize workbook
Workbook workbook = new Workbook();

// Open Excel file
workbook.Open("MultipleSheetsOnePage.xlsx");

/* NOTE: To use this feature, you should have a valid license
for GrapeCity Documents for PDF.*/

// Create a PDF document
GcPdfDocument doc = new GcPdfDocument();

// This page will save data for multiple pages
Page page = doc.NewPage();

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Get the pagination information of the workbook
IList<PageInfo> pages = printManager.Paginate(workbook);

/* Divide the multiple pages into one row and
two columns and print them on one page */
printManager.Draw(page, pages, 1, 2);

// Save the document to PDF file

```

```
doc.Save(@"PrintMultiplePagesToOnePage.pdf");
```

Keep Rows Together Over Page Breaks

GcExcel .NET enables users to keep some rows together over page breaks while exporting to a PDF file.

This feature is useful especially when you have data lying in large number of rows in the worksheet that you want to export to a PDF file. For instance - let's say you have a spreadsheet having multiple groups of rows that are often hidden, but ultimately modify the number of pages and page breaks while printing. Now, you want to export your Excel file to PDF in such a way that it keeps some groups of rows together so that they don't split across page breaks or pages when the print operation is executed. In such a scenario, it is extremely helpful to utilize this feature to achieve flawless printing experience and accurate content publishing while exporting to the PDF file.

In order to keep some groups of rows together over page breaks, you need to first create a cell range including the rows that you want to show together in the PDF file. Next, create an instance of the **PrintManager ('PrintManager Class' in the on-line documentation)** class and use the **Paginate() ('Paginate Method' in the on-line documentation)** method to ensure the desired rows are displayed together. When you are done, simply save your PDF file using the **SavePDF() ('SavePDF Method' in the on-line documentation)** method.

Using Code

Refer to the following example code to allow users to keep some rows together over page breaks while exporting to a PDF file.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Open Excel file
workbook.Open("KeepTogether.xlsx");

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

/* The first page of the natural pagination is from row 1st
   to 36th the second page is from row 37th to 73rd */
IList<IRange> keepTogetherRanges = new List<IRange>();

/* The row 37th and 38th need to keep together.
   So the pagination results are: the first page is from row 1st
   to 35th, the second page is from row 36th to 73rd*/
keepTogetherRanges.Add(worksheet.Range["36:37"]);

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Get the pagination information of the worksheet
IList<PageInfo> pages =
printManager.Paginate(worksheet, keepTogetherRanges, null);

// Save the modified pages into PDF file
printManager.SavePDF(@"KeepTogether.pdf", pages);
```

Delete Blank Pages From Middle

While exporting a workbook to a PDF file, sometimes you may encounter a couple of extra pages that are completely blank. In a workbook with large number of worksheets, it is extremely difficult to find out which pages

are empty and even more time-consuming to delete them from the middle without impacting the pagination.

In order to avoid printing and publishing of blank pages, GcExcel .NET enables users to scan through the pages of the PDF, find out which pages are blank and then exclude the blank pages from the middle while also updating the pagination information accurately.

For removing blank pages from your PDF file, you need to first create an instance of the **PrintManager** ('PrintManager Class' in the on-line documentation) class and use the **Paginate** ('Paginate Method' in the on-line documentation) method to get the default pagination of the workbook. Now, you can use the **HasPrintContent** ('HasPrintContent Method' in the on-line documentation) method to check whether the pages have content or not. Finally, call the **UpdatePageNumberAndPageSettings** ('UpdatePageNumberAndPageSettings Method' in the on-line documentation) method in order to update the indexes of the page number and the page settings for each page. When you are done, simply save your PDF file using the **SavePDF()** ('SavePDF Method' in the on-line documentation) method.

Using Code

Refer to the following example code to allow users to delete the blank pages in the middle while exporting to a PDF file.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Open Excel file
workbook.Open("DeletingBlankPages.xlsx");

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Get the natural pagination information of the workbook
IList<PageInfo> pages = printManager.Paginate(workbook);

// Remove empty pages
IList<PageInfo> newPages = new List<PageInfo>();

foreach (PageInfo page in pages)
{
    // True if there is content in the range to print
    if (printManager.HasPrintContent(page.PageContent.Range))
    {
        newPages.Add(page);
    }
}

// Update the page number and the page settings of each page
printManager.UpdatePageNumberAndPageSettings(newPages);

// Save to PDF file
printManager.SavePDF("DeleteBlankPagesInTheMiddle.pdf", newPages);
```

Export Different Headers On Different Pages

GcExcel .NET enables users to export different headers on different pages of the PDF file. This feature is useful especially when you have different information on each page of the PDF file and you want to provide different headers to each page of the PDF.

In order to configure different headers for different pages in the PDF file, you can use the **TitleRowStart** ('TitleRowStart Property' in the on-line documentation) property,

the **TitleRowEnd** ('TitleRowEnd Property' in the on-line documentation) property, and other properties of the **RepeatSetting** ('RepeatSetting Class' in the on-line documentation) class. When you are done, simply create an instance of the **PrintManager** ('PrintManager Class' in the on-line documentation) class, get the default pagination information using the **Paginate()** ('Paginate Method' in the on-line documentation) method and finally save your PDF file using the **SavePDF()** ('SavePDF Method' in the on-line documentation) method.

Using Code

Refer to the following example code in order to export different headers on different pages while exporting to a PDF file.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Open Excel file
workbook.Open("MultipleHeaders.xlsx");

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

IList<RepeatSetting> repeatSettings = new List<RepeatSetting>();

// The title rows of the "B2:F87" is "$2:$2"
RepeatSetting repeatSetting = new RepeatSetting();
repeatSetting.TitleRowStart = 1;
repeatSetting.TitleRowEnd = 1;
repeatSetting.Range = worksheet.Range["B2:F87"];
repeatSettings.Add(repeatSetting);

// The title rows of the "B89:F146" is "$89:$89"
RepeatSetting repeatSetting2 = new RepeatSetting();
repeatSetting2.TitleRowStart = 88;
repeatSetting2.TitleRowEnd = 88;
repeatSetting2.Range = worksheet.Range["B89:F146"];
repeatSettings.Add(repeatSetting2);

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();
worksheet.PageSetup.RightMargin = 10;

// Get the pagination information of the worksheet
IList<PageInfo> pages = printManager.Paginate(worksheet, null, repeatSettings);

// Save the modified pages into PDF file
printManager.SavePDF(@"ManageHeadersOnDifferentPages.pdf", pages);
```

Export Last Page Without Headers

GcExcel .NET enables users to export the last page of a PDF file without headers while keeping the headers intact in rest of the pages across the PDF file. For instance - While saving a workbook to a PDF file, you may sometimes have data in the last page of the PDF that doesn't need any headers. In such a scenario, this feature can be helpful in order to save the last page of the PDF without displaying any header information.

In order to export the last page without headers while saving to a PDF file, you need to first get the default pagination by using the **Paginate()** ('Paginate Method' in the on-line documentation) method of the **PrintManager** ('PrintManager Class' in the on-line documentation) class. Then, you can use the **PageContent** ('PageContent Property' in the on-line documentation) property of the **PageInfo**

(**'PageInfo Class' in the on-line documentation**) class and the **TitleRowStart** (**'TitleRowStart Property' in the on-line documentation**) property of the **PageContentInfo** (**'PageContentInfo Class' in the on-line documentation**) class in order to modify the header index of the last page. When you are done, simply save your file using the **SavePDF()** (**'SavePDF Method' in the on-line documentation**) method.

Using Code

Refer to the following example code to allow users to save the last page of the PDF without any headers while exporting to a PDF file.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Open Excel file
workbook.Open("ExcelData.xlsx");

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Repeat rows at the top of each page while saving PDF
worksheet.PageSetup.PrintTitleRows = "$1:$2";

// Get the natural pagination information of the workbook
IList<PageInfo> pages = printManager.Paginate(workbook);

// Modify the print header of the last page
pages[pages.Count - 1].PageContent.TitleRowStart = -1;

// Save the modified pages into PDF file
printManager.SavePDF("98-ExportLastPageWithoutHeaders.pdf", pages);
```

Export Custom Page Information

GcExcel .NET enables users to save and print custom page information while exporting to a PDF file.

For instance - Sometimes, users may want to apply different page settings and display custom page number, page count, title rows, tail rows, column headers, row headers, title columns, tail columns, range, paper width, paper height, page margins, page headers, page footers etc. as per their own preferences while exporting to a PDF file or while printing a PDF file. In this scenario, they can use this feature to showcase the desired page information instead of the default page information in the PDF file.

Depending upon the specific requirements of the users, the custom page information can be exported using the following APIs:

- Creating and using an instance of the **PageInfo** (**'PageInfo Class' in the on-line documentation**) class - The PageInfo object represents a page containing all the information needed for printing. This includes page number, page count, page content and page settings, etc.
- Creating and using an instance of the **PageContentInfo** (**'PageContentInfo Class' in the on-line documentation**) class- The PageContentInfo object represents the data area of a page which includes row header, title rows, tail rows, column header, title columns, tail columns, range, etc.
- Creating and using an instance of the **PageSettings** (**'PageSettings Class' in the on-line documentation**) class- The PageSettings object contains all the properties effecting the page settings including the paper width, paper height, page margins, page header, page footer, etc.

Using Code

Refer to the following example code to allow users to export custom page information while saving the workbook to a PDF file.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Open Excel file
workbook.Open("KeepTogether.xlsx");

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

/* Get the natural pagination information of the worksheet.
   The first page of the natural pagination is "A1:F37",
   the second page is from row "A38:F73" */
IList<PageInfo> pages = printManager.Paginate(worksheet);

// Customize the page information. The first page is "A1:F36"
pages[0].PageContent.Range = worksheet.Range["A1:F36"];

/* The center header of the first page will
   show the text "Budget summary report" */
pages[0].PageSettings.CenterHeader =
"&KFF0000&18 Budget summary report";

/* The center footer of the first page will
   show the page number "1" */
pages[0].PageSettings.CenterFooter =
"&KFF0000&16 Page &P";

// The second page is "A37:F73"
pages[1].PageContent.Range =
worksheet.Range["A37:F73"];

// Save the modified pages into PDF file
printManager.SavePDF(@"CustomPageInfos.pdf", pages);
```

Export Specific Pages To PDF

GcExcel .NET enables users to export only some specific worksheets in the workbook (and not the entire workbook) into the pages of the PDF file.

This feature is useful especially when you have a workbook containing large number of worksheets. For instance - While saving to a PDF file, you may not want to export the entire workbook containing multiple worksheets and want only some important worksheets to be saved to the PDF file. In this scenario, you can use this feature to generate a custom PDF file as per your requirements.

In order to export specific pages to the PDF file, create an instance of the **PrintManager ('PrintManager Class' in the on-line documentation)** class and get the default pagination using the **Paginate() ('Paginate Method' in the on-line documentation)** method. Next, you need to specify the pages that you want to export or print. Finally, call the **UpdatePageNumberAndPageSettings() ('UpdatePageNumberAndPageSettings Method' in the on-line documentation)** method in order to update the indexes of the page number and the page settings for each page. When you are done, simply save your PDF file using the **SavePDF() ('SavePDF Method' in the on-line documentation)** method.

Using Code

Refer to the following example code to export some specific pages to the PDF file.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Open Excel file
workbook.Open("PrintSpecificPDFPages.xlsx");

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Get the natural pagination information of the workbook
IList<PageInfo> pages = printManager.Paginate(workbook);

// Pick some pages to print
IList<PageInfo> newPages = new List<PageInfo>();
newPages.Add(pages[0]);
newPages.Add(pages[2]);

/* Update the page number and the page settings of
   each page. The page number is continuous */
printManager.UpdatePageNumberAndPageSettings(newPages);

// Save the pages into PDF file
printManager.SavePDF(@"PrintSpecificPages.pdf", newPages);
```

Save Multiple Workbooks to Single PDF

GcExcel .NET allows users to save multiple workbooks into a single Portable Document File (PDF) by using the **SavePDF()** ('SavePDF Method' in the on-line documentation) method of the **PrintManager** ('PrintManager Class' in the on-line documentation) class. Each workbook is saved to a new page in the PDF file. The information in the PDF such as the page number, number of pages, odd and even pages, first page etc. is saved on the basis of the final pagination results.

Advantage of Saving Multiple Workbooks to Single PDF File

This feature is useful especially when you need consolidated information at one place for enhanced analysis and visualization. For instance - let's say you have sales information about different versions of a product in different workbooks. Instead of sharing multiple spreadsheets or PDF files; you can share a combined PDF (by saving all the workbooks to a single PDF file) showcasing the annual sales figures of the product. This will not only help users to analyse all the crucial information at one place but it will also facilitate them in sharing, manipulating and printing all the sales data in an efficient way.

Refer to the following example code in order to export a spreadsheet to a PDF file.

C#

```
// Initialize workbook1
Workbook workbook1 = new Workbook();

// Open an Excel file
workbook1.Open("Book1.xlsx");
workbook1.Worksheets[0].PageSetup.CenterFooter = "&P of &N";

// Set page header with some company logo
workbook1.Worksheets[0].PageSetup.CenterHeader = "&G";
workbook1.Worksheets[0].PageSetup.CenterHeaderPicture.FileName = "logo.png";
workbook1.Worksheets[0].PageSetup.CenterHeaderPicture.Width = 150;
```



```

workbook1.Worksheets[0].PageSetup.CenterHeaderPicture.Height = 50;
workbook1.Worksheets[0].PageSetup.TopMargin = 100;
workbook1.Worksheets[1].PageSetup.CenterFooter = "&P of &N";

// Set page header with some company logo
workbook1.Worksheets[1].PageSetup.CenterHeader = "&G";
workbook1.Worksheets[1].PageSetup.CenterHeaderPicture.FileName = "logo.png";
workbook1.Worksheets[1].PageSetup.CenterHeaderPicture.LockAspectRatio = false;
workbook1.Worksheets[1].PageSetup.CenterHeaderPicture.Width = 150;
workbook1.Worksheets[1].PageSetup.CenterHeaderPicture.Height = 50;
workbook1.Worksheets[1].PageSetup.TopMargin = 100;

// Initialize workbook2
Workbook workbook2 = new Workbook();

// Open an Excel file
workbook2.Open("Book2.xlsx");
workbook2.Worksheets[0].PageSetup.CenterFooter = "&P of &N";
workbook2.Worksheets[0].PageSetup.CenterHeader = "GrapeCity";
workbook2.Worksheets[0].PageSetup.TopMargin = 100;

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Save the workbook1 and workbook2 into the pdf file
printManager.SavePDF(@"SaveDiffWorkBooksToOnePDF.pdf", workbook1, workbook2);

```

Working With Page Setup

GcExcel .NET allows users to paginate each worksheet using the properties of the **IPageSetup ('IPageSetup Interface' in the on-line documentation)** interface.

You can customize the page size, print area, print title rows, print title columns; specify horizontal page breaks, vertical page breaks, the maximum number of pages for horizontal and vertical pagination etc. along with zoom and scale factors (**IWorksheet Interface' in the on-line documentation**) as per your preferences while exporting a spreadsheet to a PDF file.

In order to set pagination in a worksheet, users can explore the following properties of the **IPageSetup ('IPageSetup Interface' in the on-line documentation)** interface and the **IWorksheet ('IWorksheet Interface' in the on-line documentation)** interface:

Settings	Description
IPageSetup.PaperSize ('PaperSize Property' in the on-line documentation)	This property can be used to determine the size of each page. For more information on implementation of this property, refer to Configure Paper Settings .
IPageSetup.Orientation ('Orientation Property' in the on-line documentation)	This property can be used to specify whether the worksheet should be printed in landscape orientation or portrait orientation. For more information on implementation of this property, refer to Configure Page Settings .
IPageSetup.PrintTitleRows ('PrintTitleRows Property' in the on-line documentation)	This property can be used to specify the rows that you want to print at the top of each page. For more information on implementation of this property, refer to Configure Rows to Repeat at Top and Bottom .
IPageSetup.PrintTitleColumns ('PrintTitleColumns Property' in the on-line documentation)	This property can be used to specify the columns that you want to print at the left of each page. For more information on implementation of this property, refer to Configure Columns to Repeat at Left and Right .
IPageSetup.PrintArea	This property can be used to specify the print area in a worksheet. If the

('PrintArea Property' in the on-line documentation)

print area is not specified by the user, the used range of the sheet is printed by default. For more information on implementation of this property, refer to [Configure Print Area](#).

IPageSetup.Zoom ('Zoom Property' in the on-line documentation)

This property can be used to use the result of zoom in order to paginate a worksheet. For more information on implementation of this property, refer to [Configure Paper Settings](#).

IPageSetup.FitToPagesWide ('FitToPagesWide Property' in the on-line documentation)

This property can be used to specify the maximum number of pages for horizontal pagination. After this property is set, the worksheet can be scaled to fit all columns to the pages. For more information on implementation of this property, refer to [Configure Paper Settings](#).

IPageSetup.FitToPagesTall ('FitToPagesTall Property' in the on-line documentation)

This property can be used to specify the maximum number of pages for vertical pagination. After this property is set, the worksheet can be scaled to fit all rows to the pages. For more information on implementation of this property, refer to [Configure Paper Settings](#).

IPageSetup.IsPercentScale ('IsPercentScale Property' in the on-line documentation)

This property specifies a boolean value to control how the worksheet is scaled while exporting to PDF. If the value is set to True, you can use the Zoom property of the IPageSetup interface and if the value is set to false, you can use the FitToPagesWide and FitToPagesTall property of the IPageSetup interface. For more information on implementation of this property, refer to [Configure Paper Settings](#).

IWorksheet.HPageBreaks ('HPageBreaks Property' in the on-line documentation)

This property can be used to specify the horizontal page breaks that will split rows to multiple pages. However, this property doesn't work when the property IsPercentScale is set to false. For more information on implementation of this property, refer to [Configure Page Breaks](#).

IWorksheet.VPageBreaks ('VPageBreaks Property' in the on-line documentation)

This property can be used to specify the vertical page breaks that will split columns to multiple pages. However, this property doesn't work when the property IsPercentScale is set to false. For more information on implementation of this property, refer to [Configure Page Breaks](#).

For more information on setting pagination, refer to [Configure Print Settings via Page Setup](#).



Note: The Export to PDF feature doesn't support inserting double underline, superscripts and subscripts etc. while working with page set up settings in a spreadsheet.

Import and Export CSV File

This section summarizes how GcExcel .NET handles the spreadsheet documents(.csv files).

While importing and exporting a workbook in order to open and save a csv file or stream, you can use the following properties and methods of the **CsvOpenOptions ('CsvOpenOptions Class' in the on-line documentation)** class and the **CsvSaveOptions ('CsvSaveOptions Class' in the on-line documentation)** class in order to configure several open and save options in a workbook.

Settings

CsvOpenOptions.ConvertNumericData ('ConvertNumericData Property' in the on-line documentation)

Description

This property can be used to get or set a value that indicates whether the string in text file is converted to numeric data.

CsvOpenOptions.ConvertDateTimeData ('ConvertDateTimeData Property' in the on-line documentation)

This property can be used to get or set a value that indicates whether the string in text file is converted to date data.

CsvOpenOptions.SeparatorString ('SeparatorString Property' in the on-line documentation)

This property can be used to get or set the string value as a separator.

CsvOpenOptions.Encoding ('Encoding Property' in the on-line documentation)

This property can be used to get or set the default encoding which is UTF-8.

CsvOpenOptions.ParseStyle ('ParseStyle Property' in the on-line documentation)	This property can be used to specify whether the style for parsed values should be applied while converting the string values to number or date time.
CsvOpenOptions.HasFormula ('HasFormula Property' in the on-line documentation)	This property can be used to specify whether the text is formula if it starts with "=".
CsvSaveOptions.SeparatorString ('SeparatorString Property' in the on-line documentation)	This property can be used to get or set the string value as the separator. By default, this value is a comma separator.
CsvSaveOptions.Encoding ('Encoding Property' in the on-line documentation)	This property can be used to specify the default encoding which is UTF-8.
CsvSaveOptions.ValueQuoteType ('ValueQuoteType Property' in the on-line documentation)	This property can be used to get or set how to quote values in the exported text file.
CsvSaveOptions.TrimLeadingBlankRowAndColumn ('TrimLeadingBlankRowAndColumn Property' in the on-line documentation)	This property can be used to specify whether the leading blank rows and columns should be trimmed like in Excel.

Refer to the following example code in order to import a .csv file.

C#

```
IWorkbook workbook = new Workbook();

//Method1 - Opening a csv file
workbook.Open(@"test.csv", OpenFileFormat.Csv);

//Method2 - Opening a csv file using several open options
CsvOpenOptions options = new CsvOpenOptions();
options.ConvertNumericData = false;
options.ParseStyle = false;
workbook.Open(@"test.csv", options);
```

Refer to the following example code in order to export a .csv file from a workbook or a particular worksheet in the workbook.

C#

```
// Save a csv file from workbook

IWorkbook workbook1 = new Workbook();

// Saving to a csv file
workbook1.Save(@"test.csv", SaveFileFormat.Csv);

// Saving to a csv file with advanced settings
CsvSaveOptions options1 = new CsvSaveOptions();
options1.SeparatorString = "-";
options1.ValueQuoteType = ValueQuoteType.Always;
workbook1.Save(@"test.csv", options1);

// Save a csv file from worksheet

IWorkbook workbook2 = new Workbook();
IWorksheet worksheet = workbook2.Worksheets[0];

// Saving to a csv file
worksheet.Save(@"test.csv", SaveFileFormat.Csv);

// Saving to a csv file with advanced settings
```

```
CsvSaveOptions options2 = new CsvSaveOptions();
options2.SeparatorString = "-";
options2.ValueQuoteType = ValueQuoteType.Always;
worksheet.Save(@"test.csv", options2);
```

Import and Export CSV File with Delimiters

GcExcel .NET allows users to open and save CSV files with custom delimiters for rows, cells and columns. You can use any custom character of your choice as a delimiter. For instance - Comma (,) , Semicolon (;) , Quotes (" , ') , Braces (() , {}) , pipes (|) , slashes (/ \) , Carat (^) , Pipe (|) , Tab (\t) etc.

Users can import and export the following three types of custom delimiters in CSV files as per their custom requirements and preferences. All these types of delimiters work independently and cannot be combined with each other.

1. **Column Delimiters** - These are the delimiters that separate the columns of a worksheet. By default, a column delimiter is of string type. Users can get or set the column delimiters using the options in the table shared below.

Settings

Description

CsvOpenOptions.ColumnSeparator
(**'ColumnSeparator Property' in the on-line documentation**)

This property can be used to get or set the column delimiter while opening CSV files.

CsvSaveOptions.ColumnSeparator
(**'ColumnSeparator Property' in the on-line documentation**)

This property can be used to get or set the column delimiter while saving CSV files.

2. **Row Delimiters** - These are the delimiters that separate the rows of a worksheet. By default, a row delimiter is of string type. Users can get or set the row delimiters using the options in the table shared below.

Settings

Description

CsvOpenOptions.RowSeparator
(**'RowSeparator Property' in the on-line documentation**)

This property can be used to get or set the row delimiter while opening CSV files.

CsvSaveOptions.RowSeparator
(**'RowSeparator Property' in the on-line documentation**)

This property can be used to get or set the row delimiter while saving CSV files.

3. **Cell Delimiters** - These are the delimiters that separate the cells of a worksheet. By default, the cell delimiter is of char type. Users can get or set the cell delimiters using the options in the table shared below.

Settings

Description

CsvOpenOptions.CellSeparator
(**'CellSeparator Property' in the on-line documentation**)

This property can be used to get or set the cell delimiter while opening CSV files.

CsvSaveOptions.CellSeparator
(**'CellSeparator Property' in the on-line documentation**)

This property can be used to get or set the cell delimiter while saving CSV files.

Using Code

Refer to the following example code in order to import and export CSV files with delimiters using **CsvOpenOptions** (**'CsvOpenOptions Class' in the on-line documentation**) class.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Setting ColumnSeparator, RowSeparator & CellOperator in Open CSV options
var openOption = new CsvOpenOptions();
openOption.ColumnSeparator = ",";
openOption.RowSeparator = "\r\n";
openOption.CellSeparator = '"';

// Opening CSV in workbook
workbook.Open(@"test.csv", openOption);

// Saving workbook to CSV
workbook.Save(@"4-OpenCSVDelimiterRowColumnCell.csv");
```

Refer to the following example code in order to import and export CSV files with delimiters using **CsvSaveOptions** ('**CsvSaveOptions Class**' in the on-line documentation) class.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

object[,] data = new object[,] {
    { "Name", "City", "Birthday", "Sex", "Weight", "Height",
    { "Bob", "NewYork", new DateTime(1968, 6, 8), "male", 80, 180},
    { "Betty", "NewYork", new DateTime(1972, 7, 3), "female", 72, 168},
    { "Gary", "NewYork", new DateTime(1964, 3, 2), "male", 71, 179},
    { "Hunk", "Washington", new DateTime(1972, 8, 8), "male", 80, 171},
    { "Cherry", "Washington", new DateTime(1986, 2, 2), "female", 58, 161},
    { "Eva", "Washington", new DateTime(1993, 2, 5), "female", 71, 180}};

// Set data
worksheet.Range["A1:F5"].Value = data;
worksheet.Range["A:F"].ColumnWidth = 20;

// Setting ColumnSeparator/ RowSeparator & CellOperator in Save CSV options
var saveOption = new CsvSaveOptions();
saveOption.ColumnSeparator = ",";
saveOption.RowSeparator = "\r\n";
saveOption.CellSeparator = '"';

// Saving workbook to CSV
workbook.Save(@"SaveCSVDelimiterRowColumnCell.csv", saveOption);
```

Import and Export JSON Stream

The sole purpose of facilitating users in importing and exporting to and from json stream is to enable them to exchange and organize object data as and when required. This reference summarizes how GcExcel .NET handles the import and export of json stream using .Net core.

This topic includes the following tasks.

- **Import and Export JSON Stream**

- **SpreadJS SSJSON Support**
- **Import and Export SpreadJS JSON Files with Shapes**

Import and Export JSON Stream

You can export a workbook to a json string/stream using the **ToJson method (on-line documentation)** of the **IWorkbook interface (on-line documentation)**. You can also import a json string or stream to your workbook using the **FromJson method (on-line documentation)** of the IWorkbook interface.

Refer to the following example code to import and export json stream.

C#

```
//ToJson&FromJson can be used in combination with spread.sheets product

//GcExcel import an excel file.
//change the path to real source file path.
string source = "savingfile.xlsx";
workbook.Open(source);

//GcExcel export to a json string.
var jsonstr = workbook.ToJson();
//use the json string to initialize spread.sheets product.
//spread.sheets will show the excel file contents.

//spread.sheets product export a json string.
//GcExcel use the json string to initialize.
workbook.FromJson(jsonstr);
//GcExcel export workbook to an excel file.
//change the path to real export file path.

string export = "export.xlsx";
workbook.Save(export);
```

SpreadJS SSJSON Support

GcExcel .NET provides support for SpreadJS SSJSON. You can import a SSJSON file created with Spread.Sheets Designer and save it back after modifying it as per your preferences.

C#

```
//Create a new workbook
Workbook workbook = new Workbook();

//Load SSJSON file
var stream = new System.IO.FileStream("Chart_Spread.ssjson", System.IO.FileMode.Open);
workbook.FromJson(stream);

//Save file
workbook.Save("workbook_ssjson.xlsx");
```



Note: Upon loading the SpreadJS JSON file, if users get the **ColorIndex ('ColorIndex Property' in the on-line documentation)** property of the **IBorder ('IBorder Interface' in the on-line documentation)** interface in order to set an index color, it will return a valid value only if the **Color ('Color Property' in the on-line documentation)** property of the IBorder interface is set to any rgb color; else, it will return -2 as an invalid flag. Usually, an index color can be converted to rgb color but vice versa is not possible.

Import and Export SpreadJS JSON Files with Shapes

GcExcel .NET allows users to load and save Grapecity SpreadJS JSON files with shapes. Besides importing the

existing Spread JS JSON files with shapes, users can also modify the exported Spread JS JSON files containing shapes and save them back to the original Spread JS JSON files as and when required.

C#

```
// Create a new workbook
Workbook workbook = new Workbook();

// Initialize another workbook - workbookWithShape
Workbook workbookWithShape = new Workbook();

// Fetch default worksheet of workbookWithShape
IWorksheet worksheet = workbookWithShape.Worksheets[0];

// Add a shape in the worksheet
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Parallelogram, 1, 1, 200, 100);
shape.Line.DashStyle = LineDashStyle.Dash;
shape.Line.Style = LineStyle.Single;
shape.Line.Weight = 2;
IColorFormat color = shape.Fill.Color;
shape.Line.Transparency = 0.3;
color.ObjectThemeColor = ThemeColor.Accent6;

// Converting workbook containing shape to JsonString
String jsonString = workbookWithShape.ToJson();

// GcExcel can load json string containing shapes
workbook.FromJson(jsonString);

// Saving the workbook
workbook.Save(@"7-LoadSaveShapesSSJSON.xlsx");
```

Import and Export Macros

This section summarizes how GcExcel.NET handles the import and export of Excel files containing macros. Using GcExcel.NET, users can load and save Excel files containing macros (.xlsm files) without any hassles. Please note that GcExcel will not execute these macros.

Typically, this feature has been introduced in order to allow users to load and save macro-enabled spreadsheets. Macros help automate repetitive tasks and hence, reduce significant amount of time while working with spreadsheets. Now, users can load such spreadsheets in GcExcel directly as Xlsm files, modify them easily and quickly and then save them back.

During the execution of import and export operations on the Excel files, all the macros will also be preserved concurrently along with the data. While opening and saving the Excel workbooks or Excel macro-enabled workbooks, macros will always be imported and exported respectively. The form controls and ActiveX controls are also supported during the import and export operations.

When the **OpenFileFormat ('OpenFileFormat Enumeration' in the on-line documentation)** is Xlsm, macros will be imported. When the **SaveFileFormat ('SaveFileFormat Enumeration' in the on-line documentation)** is Xlsm, macros will be exported.



Note: While preserving the macros on import or export of Excel files, GcExcel will not execute these macros.

Refer to the following example code in order to import and export macros in spreadsheet documents.

C#

```
// Open a .xlsm file with file name
var workbook = new Workbook();
workbook.Open("testfile.xlsm");

// Save workbook as Excel macro-enabled workbook file
```

```
var workbook = new Workbook();
workbook.Save("file.xlsx");

// Save workbook as Excel macro enabled workbook into stream
var workbook = new Workbook();
var request = WebRequest.CreateHttp("https://path/to/excel/file/upload");
request.Method = "POST";
request.ContentType = "application/x-www-form-urlencoded";
var workbookContent = new MemoryStream();
workbook.Save(workbookContent, SaveFileFormat.Xlsm);
workbookContent.Seek(0, SeekOrigin.Begin);
request.ContentLength = workbookContent.Length;
using (var reqStream = request.GetRequestStream())
{
    workbookContent.CopyTo(reqStream);
}
```

Import and Export OLE Objects

GcExcel .NET allows users to preserve OLE objects while opening and saving an Excel file. This feature is extremely useful when users need to deal with import and export of linked objects and embedded objects while working with spreadsheets.

With extensive support for importing and exporting OLE Objects, users can insert linked and embedded objects in their spreadsheets and then preserve these objects while saving the files with .xlsx or .xlsm extension. This feature also facilitates users to use the object linking and embedding (OLE) in order to load and save data from other programs, such as MS Word or MS Excel.

Example

For instance, let's say you work as a business analyst who wants to visualize information using charts.

You have a source file containing some data. But, you want the chart to be displayed in another file (called a destination file) that picks up data from the source file in order to create charts in the destination file. Now, whenever any changes are done in the data in the source file, obviously you would also want the chart to be updated (or in other words, the destination file to be updated).

That's where the role of supporting the import and export of OLE objects comes into picture. In such a scenario, GcExcel will ensure that the original data remains intact in the source file and the destination file represents the updated linked information (updated charts in this example) without impacting the storage of the original data.

Refer to the following example code in order to import and export spreadsheets containing OLE objects.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Opening workbook with OLE object
workbook.Open("OleObjectExcelFile.xlsx");

// Saving workbook with OLE object
workbook.Save("OleOutExcel.xlsx");
```

Apply Conditional Formatting

You can apply conditional formatting in individual cells or a range of cells using rules or conditional operators. The set of conditional formatting rules for a range is represented with the **FormatConditions property (on-line documentation)** of the **IRange interface (on-line documentation)**.

Shared below is a list of conditional formatting rules that can be applied in a worksheet.

- [Add Cell Value Rule](#)
- [Add Date Occurring Rule](#)
- [Add Average Rule](#)
- [Add Color Scale Rule](#)
- [Add Data Bar Rule](#)
- [Add Top Bottom Rule](#)
- [Add Unique Rule](#)
- [Add Icon Sets Rule](#)
- [Add Expression Rule](#)

If you want to delete the formatting rule applied to the cell range in a worksheet, you can do it by using the **Delete method (on-line documentation)** of **IFormatCondition interface (on-line documentation)**

Add Cell Value Rule

The cell value rule compares values entered in the cells with the condition specified in the conditional formatting rule. In order to add a cell value rule, you can use the **Formula1 property (on-line documentation)** and **Formula2 property (on-line documentation)** of the **IFormatCondition interface (on-line documentation)**. You can also use the **Operator property (on-line documentation)** of the **IFormatCondition interface** to set the operator that will perform the comparison operation, like "Between", "Less Than" etc.

Refer to the following example code to add cell value rule to a range of cells in a worksheet.

C#

```
// Assigning value using object
worksheet.Range["A1:A5"].Value = new object[,]
{
    {1},{3},{5},{7},{9}
};
// Defining format rules.
IFormatCondition condition =
worksheet.Range["A1:A5"].FormatConditions.Add(FormatConditionType.CellValue,
FormatConditionOperator.Between, 1, 5) as IFormatCondition;
condition.NumberFormat = "0.000";
```

Add Date Occurring Rule

The date occurring rule in conditional formatting feature compares the values entered in date format in the cells or a range of cells. This rule can be added using the **DateOperator property (on-line documentation)** of the **IFormatCondition interface (on-line documentation)**.

Refer to the following example code to add date occurring rule to a range of cells in a worksheet.

C#

```
// Adding Date occurring rules
IFormatCondition condition =
worksheet.Range["A1:A4"].FormatConditions.Add(FormatConditionType.TimePeriod) as
IFormatCondition;
condition.DateOperator = TimePeriods.Yesterday;
condition.Interior.Color = Color.FromArgb(128, 0, 128);
```

```
DateTime now = DateTime.Today;
worksheet.Range["A1"].Value = now.AddDays(-2);
worksheet.Range["A2"].Value = now.AddDays(-1);
worksheet.Range["A3"].Value = now;
worksheet.Range["A4"].Value = now.AddDays(1);
```

Add Average Rule

The average rule in conditional formatting can be added and deleted using the properties and methods of the **IAboveAverage interface (on-line documentation)**.

Refer to the following example code to add average rule to a range of cells in a worksheet.

C#

```
// Adding average rule
worksheet.Range["A1"].Value = 1;
worksheet.Range["A2"].Value = 2;
worksheet.Range["A3"].Value = 3;
worksheet.Range["A4"].Value = 4;
worksheet.Range["A5"].Value = 60000000;

IAboveAverage averageCondition =
worksheet.Range["A1:A5"].FormatConditions.AddAboveAverage();
averageCondition.AboveBelow = AboveBelow.AboveAverage;
averageCondition.NumStdDev = 2;
averageCondition.NumberFormat = "0.00";
```

Add Color Scale Rule

The color scale rule uses a sliding color scale to format cells or a range of cells. For instance, if numeric cell value 1 is represented with color yellow and 50 with green, then 25 would be light green. This rule can be added using the properties and methods of the **IColorScale interface (on-line documentation)**.

Refer to the following example code to add color scale rule to a cell range in a worksheet.

C#

```
// Adding colorscale rule
IColorScale twoColorScaleRule =
worksheet.Range["A2:E2"].FormatConditions.AddColorScale(ColorScaleType.TwoColorScale);

worksheet.Range["A2"].Value = 1;
worksheet.Range["B2"].Value = 2;
worksheet.Range["C2"].Value = 3;
worksheet.Range["D2"].Value = 4;
worksheet.Range["E2"].Value = 5;

twoColorScaleRule.ColorScaleCriteria[0].Type = ConditionValueTypes.Number;
twoColorScaleRule.ColorScaleCriteria[0].Value = 1;
twoColorScaleRule.ColorScaleCriteria[0].FormatColor.Color = Color.FromArgb(255, 0, 0);

twoColorScaleRule.ColorScaleCriteria[1].Type = ConditionValueTypes.Number;
twoColorScaleRule.ColorScaleCriteria[1].Value = 5;
twoColorScaleRule.ColorScaleCriteria[1].FormatColor.Color = Color.FromArgb(0, 255, 0);
```

Add Data Bar Rule

The data bar rule in conditional formatting displays a bar in the cell on the basis of cell values entered in a range.

This rule can be added using the properties and methods of the **IDataBar interface (on-line documentation)**.

Refer to the following example code to add data bar rule to a range of cells in a worksheet.

C#

```
// Adding Databar rule
worksheet.Range["A1:A5"].Value = new object[,]
{
    {1},
    {2},
    {3},
    {4},
    {5}
};

IDataBar dataBar = worksheet.Range["A1:A5"].FormatConditions.AddDatabar();

dataBar.MinPoint.Type = ConditionValueTypes.LowestValue;
dataBar.MinPoint.Value = null;
dataBar.MaxPoint.Type = ConditionValueTypes.HighestValue;
dataBar.MaxPoint.Value = null;

dataBar.BarFillType = DataBarFillType.Solid;
dataBar.BarColor.Color = Color.Green;
dataBar.Direction = DataBarDirection.Context;
dataBar.AxisColor.Color = Color.Red;
dataBar.AxisPosition = DataBarAxisPositionAutomatic;
dataBar.NegativeBarFormat.BorderColorType = DataBarNegativeColorType.Color;
dataBar.NegativeBarFormat.BorderColor.Color = Color.FromArgb(128, 0, 212);
dataBar.NegativeBarFormat.ColorType = DataBarNegativeColorType.Color;
dataBar.NegativeBarFormat.Color.Color = Color.FromArgb(128, 0, 240);
dataBar.ShowValue = false;
```

Add Top Bottom Rule

The top bottom rule checks whether the values in the top or bottom of a cell range match with the required values in the cell. In case the values don't match, the data is considered as invalid. This rule can be added using the properties and methods of the **ITop10 interface (on-line documentation)**.

The following options are available while adding top bottom rule in a worksheet:

- Top 10
- Top 10%
- Bottom 10
- Bottom 10%
- Above Average
- Below Average

Refer to the following example code to add top bottom rule in a worksheet.

C#

```
// Adding ToBottom rule
worksheet.Range["A1:A5"].Value = new object[,]
{
    {1},
    {2},
    {3},
    {4},
    {5}
};
```

```
ITop10 condition = worksheet.Range["A1:A5"].FormatConditions.AddTop10();
condition.TopBottom = TopBottom.Top10Top;
condition.Rank = 50;
condition.Percent = true;
condition.Interior.Color = Color.FromArgb(128, 0, 128);
```

Add Unique Rule

The unique rule in conditional formatting is applied to check whether the value entered in a cell is a unique value in that particular range. This is possible only when the duplication option is set to false. To check for the duplicate values, the duplicate rule is applied separately.

Unique rule can be added using the properties and methods of the **IUniqueValues interface (on-line documentation)**.

Refer to the following example code to add unique rule in a worksheet.

C#

```
// Adding Unique Rule
worksheet.Range["A1:A5"].Value = new object[,]
{
    {1},
    {2},
    {1},
    {3},
    {4}
};

IUniqueValues condition2 = worksheet.Range["A1:A5"].FormatConditions.AddUniqueValues();
condition2.DupeUnique = DupeUnique.Unique;
condition2.Font.Name = "Arial";
```

Add Icon Sets Rule

The icon sets rule in conditional formatting displays the icons on the basis of values entered in the cells. Each value represents a distinct icon that appears in a cell if it matches the icon sets rule applied on it. This rule can be added using the properties and methods of the **IIconSetCondition interface (on-line documentation)**.

Refer to the following example code to add icon sets rule in a worksheet.

C#

```
// Adding IconSets rule
IIconSetCondition condition =
worksheet.Range["A1:A5"].FormatConditions.AddIconSetCondition();
condition.IconSet = workbook.IconSets[IconSetType.Icon3Symbols];
condition.IconCriteria[1].Operator = FormatConditionOperator.GreaterEqual;
condition.IconCriteria[1].Value = 50;
condition.IconCriteria[1].Type = ConditionValueTypes.Percent;
condition.IconCriteria[2].Operator = FormatConditionOperator.GreaterEqual;
condition.IconCriteria[2].Value = 70;
condition.IconCriteria[2].Type = ConditionValueTypes.Percent;

worksheet.Range["A1"].Value = 1;
worksheet.Range["A2"].Value = 2;
worksheet.Range["A3"].Value = 3;
worksheet.Range["A4"].Value = 4;
worksheet.Range["A5"].Value = 5;
```

Add Expression Rule

The expression rule in conditional formatting is used to set the expression rule's formula. This rule can be added using the properties and methods of the **IFormatCondition interface (on-line documentation)**.

Refer to the following example code to add expression rule in a worksheet.

C#

```
// Adding Expression Rule
worksheet.Range["A1:B5"].Value = new object[,]
{
    {1, 2},
    {0, 1},
    {0, 0},
    {0, 3},
    {4, 5}
};
IFormatCondition condition =
worksheet.Range["B1:B5"].FormatConditions.Add(FormatConditionType.Expression, 0, "=A1")
as IFormatCondition;
condition.Interior.Color = Color.FromArgb(255, 0, 0);
```

Apply Data Validations

You can use the data validation feature in GcExcel .NET to ensure users enter only the valid values into a cell while working in a spreadsheet.

For instance, let's say you have a worksheet where you want users to enter only whole numbers between 1 to 15. To accomplish this, you can create a data validation rule that restricts users to enter cell values other than a whole number between 1 to 15. You can even create custom dropdown lists to specify the possible values that can be entered in the cells or display messages or error alerts to validate the data and get notified if there is something wrong with the information entered in the worksheets.

Applying data validations in worksheets involves the following tasks.

- [Add Validations](#)
- [Modify Validations](#)
- [Delete Validation](#)

Add Validations

You can use the **Add method (on-line documentation)** of the **IValidation interface (on-line documentation)** to apply data validation to individual cells or a range of cells in a spreadsheet. A single cell can have only one validation rule and if you try to apply validation on a cell that already possesses a validation rule, it will throw an exception.

Validation rule instance for a range is represented with the **Validation property (on-line documentation)** of the **IRange interface (on-line documentation)**. If you want to know whether a cell range already contains the validation rule, you can use the **HasValidation property (on-line documentation)** of the IRange interface. If all the cells in a range possess the same validation rule applied to them, it is represented with the **ValidationIsSame property (on-line documentation)** of the IRange interface.

Shared below is a list of data validations operations that can be implemented in GcExcel .NET.

- **Add Whole Number Validation**
- **Add Decimal Validation**
- **Add List Validation**
- **Add Date Validation**
- **Add Time Validation**
- **Add Text Length Validation**
- **Add Custom Validation**

Add whole number validation

You can validate your data and ensure users add only whole numbers in cells or a range of cells by applying the whole number validation in a worksheet.

Refer to the following example code to add whole number validation.

C#

```
//Add whole number validation
worksheet.Range["A1:A3"].Validation.Add(ValidationType.Whole,
ValidationAlertStyle.Stop, ValidationOperator.Between, 1, 8);
IValidation validation = worksheet.Range["A1:A3"].Validation;
validation.IgnoreBlank = true;
validation.InputTitle = "Tips";
validation.InputMessage = "Input a value between 1 and 8, please";
validation.ErrorTitle = "Error";
validation.ErrorMessage = "input value does not between 1 and 8";
validation.ShowInputMessage = true;
validation.ShowError = true;
```

Add decimal validation

You can validate your data and ensure users add only decimal numbers in cells or a range of cells by applying the

decimal validation in a worksheet.

Refer to the following example code to add decimal validation.

C#

```
//Add Decimal validation
worksheet.Range["B1:B3"].Validation.Add(ValidationType.Decimal,
ValidationAlertStyle.Stop, ValidationOperator.Between, 3.4, 102.8);
```

Add list validation

You can also validate lists inserted in cells or a range of cells by applying the list validation in your worksheet .

Refer to the following example code to add list validation.

C#

```
//Add List Validation
worksheet.Range["C4"].Value = "aaa";
worksheet.Range["C5"].Value = "bbb";
worksheet.Range["C6"].Value = "ccc";

//Use cell reference.
worksheet.Range["C1:C3"].Validation.Add(ValidationType.List, ValidationAlertStyle.Stop,
ValidationOperator.Between, "=c4:c6");

//Or use string.
//this._worksheet.Range["C2:E4"].Validation.Add(ValidationType.List,
ValidationAlertStyle.Stop, ValidationOperator.Between, "aaa, bbb, ccc");

//Display list dropdown
IValidation dvalidation = worksheet.Range["C1:C3"].Validation;
dvalidation.InCellDropdown = true;
```

Add date validation

You can validate data entered in date format in cells or a range of cells by applying the date validation in a worksheet.

Refer to the following example code to add date validation.

C#

```
//Add Date validation
worksheet.Range["D1:D3"].Validation.Add(ValidationType.Date, ValidationAlertStyle.Stop,
ValidationOperator.Between, new DateTime(2015, 12, 13), new DateTime(2015, 12, 18));
```

Add time validation

You can validate the time entered in cells or a range of cells by applying the time validation in a worksheet.

Refer to the following example code to add time validation.

C#

```
//Add Time Validation
worksheet.Range["E1:E3"].Validation.Add(ValidationType.Time, ValidationAlertStyle.Stop,
ValidationOperator.Between, new TimeSpan(13, 30, 0), new TimeSpan(18, 30, 0));
```

Add text length validation

You can validate the length of the text entered in cells or a range of cells by applying the text length validation in a worksheet.

Refer to the following example code to add text length validation.

C#

```
//Add Text Length Validation
worksheet.Range["C2:E4"].Validation.Add(ValidationType.TextLength,
ValidationAlertStyle.Stop, ValidationOperator.Between, 2, 3);
```

Add custom validation

You can add a custom validation rule to validate data in a worksheet by applying custom validation.

Refer to the following example code to add custom validation.

C#

```
//Add custom validation
worksheet.Range["A2"].Value = 1;
worksheet.Range["A3"].Value = 2;
worksheet.Range["C2"].Value = 1;
//when use custom validation, validationOperator and formula2 parameters will be
ignored even if you have given.
worksheet.Range["A2:A3"].Validation.Add(ValidationType.Custom,
ValidationAlertStyle.Information, formula1: "=C2");
```

Delete Validation

You can delete the applied validation rule using the **Delete method (on-line documentation)** of the **IValidation interface (on-line documentation)**.

Refer to the following example code to know how you can delete validation rule applied to a cell or a range of cells in a worksheet.

C#

```
//Add validation
worksheet.Range["A1:A3"].Validation.Add(ValidationType.Whole,
ValidationAlertStyle.Stop, ValidationOperator.Between, 1, 8);
worksheet.Range["B1:B3"].Validation.Add(ValidationType.Whole,
ValidationAlertStyle.Stop, ValidationOperator.Between, 11, 18);

//Delete validation.
worksheet.Range["A1:A2"].Validation.Delete();
```

Modify Validation

You can change the validation rule for a range by using either of the two ways described below:

- Set properties of the **IValidation interface (on-line documentation)** (**Type property (on-line documentation)**, **Formula1 property (on-line documentation)**, **Formula2 property (on-line documentation)**, and many more).
- Use **Delete method (on-line documentation)** of the IValidation interface to first delete validation rule and then use the **Add method (on-line documentation)** to add the new rule.

Refer to the following example code to know how you can modify an existing validation rule applied to a cell or a range of cells in a worksheet.

C#

```
//Add validation
worksheet.Range["A1:A2"].Validation.Add(ValidationType.Date, ValidationAlertStyle.Stop,
ValidationOperator.Between, new TimeSpan(13, 30, 0), new TimeSpan(18, 30, 0));

//Modify validation.
```



```
worksheet.Range["A1:A2"].Validation.Type = ValidationType.Time;  
worksheet.Range["A1:A2"].Validation.AlertStyle = ValidationAlertStyle.Stop;  
worksheet.Range["A1:A2"].Validation.Operator = ValidationOperator.Between;  
worksheet.Range["A1:A2"].Validation.Formula1 = new TimeSpan(13, 30,  
0).TotalDays.ToString();  
worksheet.Range["A1:A2"].Validation.Formula2 = new TimeSpan(18, 30,  
0).TotalDays.ToString();
```

Configure Print Settings via Page Setup

GcExcel .NET allows users to configure print settings by customizing the page setup properties as per their preferences.

1. [Configure Page Header and Footer](#)
2. [Configure Page Settings](#)
3. [Configure Page Breaks](#)
4. [Configure Paper Settings](#)
5. [Configure Print Area](#)
6. [Configure Columns to Repeat at Left and Right](#)
7. [Configure Rows to Repeat at Top and Bottom](#)
8. [Configure Sheet Print Settings](#)

Configure Page Header and Footer

In GcExcel .NET, you can use the **LeftHeader property (on-line documentation)**, **LeftFooter property (on-line documentation)**, (**'LeftFooter Property' in the on-line documentation**) **CenterFooter property (on-line documentation)**, **RightHeader property (on-line documentation)**, **CenterHeader property (on-line documentation)**, and the **RightFooter property (on-line documentation)** of the **IPageSetup interface (on-line documentation)** in order to configure header and footer for a page.

C#

```
//Configure PageHeader and PageFooter
//Set header for the page
worksheet.PageSetup.LeftHeader = "&\\"Arial,Italic\\"LeftHeader";
worksheet.PageSetup.CenterHeader = "&P";

//Set footer graphic for the page
worksheet.PageSetup.CenterFooter = "&G";
worksheet.PageSetup.CenterFooterPicture.Filename = @"Resource\logo.png";
```

For special settings, you can also perform the following tasks in order to customize the configuration of the header and footer of your page:

1. **Configure first page header and footer**
2. **Configure even page header and footer**

Configure first page header and footer

If you want a different header and footer in your first page, you first need to set the **DifferentFirstPageHeaderFooter property (on-line documentation)** of the **IPageSetup interface** to true. When this is done, you can use the properties of the **IPageSetup interface** in order to configure the first page header and footer.

C#

```
//Set first page header and footer
worksheet.PageSetup.DifferentFirstPageHeaderFooter = true;

worksheet.PageSetup.FirstPage.CenterHeader.Text = "&T";
worksheet.PageSetup.FirstPage.RightFooter.Text = "&D";

//Set first page header and footer graphic
worksheet.PageSetup.FirstPage.LeftFooter.Text = "&G";
worksheet.PageSetup.FirstPage.LeftFooter.Picture.Filename = @"Resource\logo.png";
```

Configure even page header and footer

If you want a different header and footer for all the even pages, you first need to set

the **OddAndEvenPagesHeaderFooter property (on-line documentation)** to true. When this is done, you can use the properties of the **IPageSetup** interface in order to configure the even page header and footer.

C#

```
//Set even page header and footer
worksheet.PageSetup.OddAndEvenPagesHeaderFooter = true;

worksheet.PageSetup.EvenPage.CenterHeader.Text = "&T";
worksheet.PageSetup.EvenPage.RightFooter.Text = "&D";

//Set even page header and footer graphic
worksheet.PageSetup.EvenPage.LeftFooter.Text = "&G";
worksheet.PageSetup.EvenPage.LeftFooter.Picture.FileName = @"Resource\logo.png";
```

Configure Page Settings

In GcExcel .NET, you can use the properties of the **IPageSetup** interface ([on-line documentation](#)) in order to configure page settings.

Configuring page settings involves the following tasks:

1. **Configure Page Margins**
2. **Configure Page Orientation**
3. **Configure Page Order**
4. **Configure Page Center**
5. **Configure First Page Number**

Configure Page Margins

You can use the **TopMargin property (on-line documentation)**, **RightMargin property (on-line documentation)** and **BottomMargin property (on-line documentation)** of the **IPageSetup** interface in order to configure margins for a page.

C#

```
//Set page margins, in points.
worksheet.PageSetup.TopMargin = 36;
worksheet.PageSetup.BottomMargin = 36;
worksheet.PageSetup.RightMargin = 72;
```



Note: While you set margins for your page, it is necessary to ensure that it should not be less than Zero.

Configure Page Orientation

You can use the **Orientation property (on-line documentation)** of the **IPageSetup** interface in order to set the orientation for a page to Portrait or Landscape as per your preferences.

C#

```
//Set page orientation.

worksheet.PageSetup.Orientation = PageOrientation.Landscape;
```

Configure Page Order

You can use the **Order property (on-line documentation)** of the **IPageSetup** interface in order to configure the order of the page as per your choice.

C#

```
//Set page order. The default value is DownThenOver.  
  
worksheet.PageSetup.Order = Order.OverThenDown;
```

Configure Page Center

You can use the **CenterHorizontally property (on-line documentation)** and the **CenterVertically property (on-line documentation)** of the IPageSetup interface in order to configure the center of your page according to your preferences.

```
C#  
  
//Set center. The default value is false.  
  
worksheet.PageSetup.CenterHorizontally = true;  
worksheet.PageSetup.CenterVertically = true;
```

Configure First Page Number

You can use the **FirstPageNumber property (on-line documentation)** of the IPageSetup interface in order to configure the number for your first page as per your choice.

```
C#  
  
//Set first page number. The default value is p1.  
  
worksheet.PageSetup.FirstPageNumber = 3;
```

Configure Page Breaks

GcExcel .NET allows users to configure the vertical and horizontal page breaks by using the **VPageBreaks ('VPageBreaks Property' in the on-line documentation)** property and **HPageBreaks ('HPageBreaks Property' in the on-line documentation)** property of the **IWorksheet ('IWorksheet Interface' in the on-line documentation)** interface. You can also determine whether to adjust the horizontal and vertical page breaks or keep them fixed (while performing the insert and delete operations on the rows and columns) by using the **FixedPageBreaks ('FixedPageBreaks Property' in the on-line documentation)** property of the IWorksheet interface.

This feature is useful especially when users need to print different reports from Excel to a PDF file. With the option to choose whether to adjust page breaks or keep them fixed, users can specify whether each section appears on a separate page or starts from a new page whenever any rows and columns are inserted or deleted in a spreadsheet.

If the **FixedPageBreaks ('FixedPageBreaks Property' in the on-line documentation)** property is set to false (this is the default behavior), then:

- The horizontal and vertical page breaks are adjusted when the rows and columns are inserted or deleted from the worksheet.
- The row or column index of the page break is increased or decreased according to the inserted and deleted rows or columns based on the following conditions:
 - If the inserted or deleted rows or columns exist after the page break, the row or column index of the page break remains unchanged.
 - If the deleted rows or columns are present before the page break, the row or column index of the page break is decreased accordingly.
 - If the deleted rows or columns contain the page break, the page break will be removed.

If the **FixedPageBreaks ('FixedPageBreaks Property' in the on-line documentation)** property is set to true, the row or column index of page breaks are not changed even after inserting or deleting rows or columns. Further, the horizontal and vertical page breaks are considered "fixed" and the page breaks can't be adjusted in this scenario.

Refer to the following example code in order to configure page breaks for customized printing.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

object[,] data = new object[,] {
    {"Name", "City", "Birthday", "Sex", "Weight", "Height", "Age"},
    {"Bob", "NewYork", new DateTime(1968, 6, 8), "male", 80, 180, 56},
    {"Betty", "NewYork", new DateTime(1972, 7, 3), "female", 72, 168, 45},
    {"Gary", "NewYork", new DateTime(1964, 3, 2), "male", 71, 179, 50},
    {"Hunk", "Washington", new DateTime(1972, 8, 8), "male", 80, 171, 59},
    {"Cherry", "Washington", new DateTime(1986, 2, 2), "female", 58, 161, 34},
    {"Coco", "Virginia", new DateTime(1982, 12, 12), "female", 58, 181, 45},
    {"Lance", "Chicago", new DateTime(1962, 3, 12), "female", 49, 160, 57},
    {"Eva", "Washington", new DateTime(1993, 2, 5), "female", 71, 180, 81}};

// Set data
worksheet.Range["A1:G9"].Value = data;

// Add a horizontal page break before the fourth row
var hPageBreak = worksheet.HPageBreaks.Add(worksheet.Range["F4"]);

// Add a vertical page break before the third column
var vPageBreak = worksheet.VPageBreaks.Add(worksheet.Range["F3"]);

// Saving workbook to.xlsx
workbook.Save(@"PageBreaks.xlsx", SaveFileFormat.Xlsx);

// Delete rows and columns before the page breaks, the page breaks will be adjusted
worksheet.Range["1:2"].Delete(); // the hPageBreak is before the fourth row
worksheet.Range["B:C"].Delete(); // the vPageBreak is before the fourth column

// Set the page breaks are fixed, it will not be adjusted when inserting/ deleting
rows/ columns
worksheet.FixedPageBreaks = true;

// Saving the edited workbook to.xlsx
workbook.Save(@"PageBreaksAfterDeletingRows&ColumnsWithFixedPageBreaks.xlsx",
    SaveFileFormat.Xlsx);

// Delete rows and columns after the page breaks, the page breaks will not be adjusted
worksheet.Range["1:2"].Delete(); // the hPageBreak is still before the fourth row
worksheet.Range["B:C"].Delete(); // the vPageBreak is still before the fourth column

// Insert rows
worksheet.Range["A3:A5"].EntireRow.Insert(); // Inserting rows after deleting row and
column ranges

// Saving the finalized workbook to.xlsx
workbook.Save(@"PageBreakAfterDeletingRows&Columns.xlsx", SaveFileFormat.Xlsx);
```

Configure Paper Settings

In GcExcel .NET, you can use the properties of the **IPageSetup interface (on-line documentation)** in order to configure paper settings for customized printing.

Configuring paper settings involves the following tasks:

1. **Configure Paper Scaling**
2. **Configure Paper Size**

Configure Paper Scaling

You can use the **IsPercentScale property (on-line documentation)** in order to control how the spreadsheet is scaled; the **FitToPagesTall property (on-line documentation)** and the **FitToPagesWide property (on-line documentation)** in order to set its size; and the **Zoom property (on-line documentation)** in order to adjust the size of the paper that will be used for printing.

C#

```
//Set paper scaling via percent scale

worksheet.PageSetup.IsPercentScale = true;
worksheet.PageSetup.Zoom = 150;

//Set paper scaling via FitToPagesWide and FitToPagesTall

worksheet.PageSetup.IsPercentScale = false;
worksheet.PageSetup.FitToPagesWide = 3;
worksheet.PageSetup.FitToPagesTall = 4;
```

Configure Paper Size

You can use the **PaperSize property (on-line documentation)** in order to set the paper size for the paper that will be used for printing.

C#

```
//Set built-in paper size. The Default is Letter

worksheet.PageSetup.PaperSize = PaperSize.A4;
```

Configure Print Area

At times, you may want to print only a specific area in a worksheet instead of printing the whole worksheet.

GcExcel .NET supports customized printing by allowing users to select a range of cells in order to configure the desired print area in a worksheet. This can be done by using the **PrintArea property (on-line documentation)** of the **IPageSetup interface (on-line documentation)**.

C#

```
//Set print area in the worksheet
worksheet.PageSetup.PrintArea = "$D$5:$G$10";
```

Configure Columns to Repeat at Left and Right

You can configure columns in a worksheet in order to repeat them at the left by using the **PrintTitleColumns property (on-line documentation)** and at the right by using the **PrintTailColumns ('PrintTailColumns Property' in the on-line documentation)** property of the **IPageSetup interface (on-line documentation)**.

This feature is useful especially when you're using GcExcel .NET to create reports wherein you want to repeat

specific title columns and tail columns in the exported file. With support for repeating specific columns at the left and right side of the page; it becomes much easier and quicker to handle and visualize spreadsheets containing large number of columns.

While exporting a spreadsheet with repeating columns to a PDF file, the tail columns will be exported only when its index is larger than the page's last column's index. Otherwise, the tail column is ignored. For instance, if the Print Area is "A1:J200" and the right repeating column is "\$I:\$J"; it will print "\$I:\$J" repeatedly on each page. However, if users set the right repeating column to "\$K:\$L", then it will not print "\$K:\$L" (because the column index is larger than print area).

Refer to the following example code in order to configure columns to repeat at the right.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Populating cells in worksheet
var range = worksheet.Range["A1:J200"];
for (int i = 0; i < 200; i++)
    for (int j = 0; j < 8; j++)
    {
        range.Cells[i, j].Value = i.ToString();
        range.Cells[i, 8].Value = "Col I";
        range.Cells[i, 9].Value = "Col J";
    }

// Repeat Columns from I to J at the right of each page while saving PDF
worksheet.PageSetup.PrintTailColumns = "$I:$J";

// Saving workbook to PDF
workbook.Save(@"ConfigureTailColumns.pdf", SaveFileFormat.Pdf);
```

Refer to the following example code in order to configure columns to repeat at the left.

C#

```
//Set columns to repeat at left
worksheet.PageSetup.PrintTitleColumns = "$D:$G";
```

Configure Rows to Repeat at Top and Bottom

You can configure rows in a worksheet in order to repeat them at the top by using the **PrintTitleRows** ('PrintTitleRows Property' in the on-line documentation) property ('PrintTitleColumns Property' in the on-line documentation) and at the bottom using the **PrintTailRows** ('PrintTailRows Property' in the on-line documentation) property of the **IPageSetup** interface (on-line documentation).

While exporting a spreadsheet with repeating rows to a PDF file, the tail rows will be exported only when its index is larger than the page's last row's index. Otherwise, the tail row is ignored. For instance, if the Print Area is "B5:H23" and the top repeating row is "\$3:\$3"; it will print "\$3:\$3" repeatedly on each page. However, if users set the top repeating row to "\$30:\$30", then it will not print "\$30:\$30" (because the row index is larger than print area).

Refer to the following example code in order to configure rows to repeat at the bottom.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();

// Fetch default worksheet
```

```
IWorksheet worksheet = workbook.Worksheets[0];

// Populating cells in worksheet
var range = worksheet.Range["A1:J200"];
for (int i = 0; i < 200; i++)
    for (int j = 0; j < 10; j++)
    {
        range.Cells[i, j].Value = i.ToString();
        range.Cells[199, j].Value = "Row 199";
    }

//Repeat Row 200 at the bottom of each page while saving PDF
worksheet.PageSetup.PrintTailRows = "$200:$200";

// Saving workbook to PDF
workbook.Save(@"ConfigureTailRows.pdf", SaveFileFormat.Pdf);
```

Refer to the following example code in order to configure rows to repeat at the top.

C#

```
//Set rows to repeat at top
worksheet.PageSetup.PrintTitleRows = "$5:$10";
```

Configure Sheet Print Settings

You can set the **PrintGridlines property (on-line documentation)**, **PrintHeadings property (on-line documentation)**, **BlackAndWhite property (on-line documentation)**, **PrintComments property (on-line documentation)** and **PrintErrors property (on-line documentation)** of the **IPageSetup interface (on-line documentation)** in order to configure the print settings for the sheet.

C#

```
//Configure sheet print settings

worksheet.PageSetup.PrintGridlines = true;
worksheet.PageSetup.PrintHeadings = true;
worksheet.PageSetup.BlackAndWhite = true;
worksheet.PageSetup.PrintComments = PrintLocation.InPlace;
worksheet.PageSetup.PrintErrors = PrintErrors.Dash;
```

API Reference

The complete GcExcel .NET component includes the assembly listed in the table shared below. For more details, you can click on the name of the assembly to know about the namespaces defined in it.

Assembly

Grapecity.Documents.Excel
(**'GrapeCity.Documents.Excel Assembly' in the on-line documentation**)

Description

Provides the namespaces for the GcExcel .NET component functionality for .Net Core.

For help with using the product, refer to the [Key Features](#).

Index

- Access a Range, 31-32**
- Access Areas in a Range, 32**
- Access Cells, Rows and Columns in a Range, 32-33**
- Add and Delete Table Columns and Rows, 111-112**
- Add Average Rule, 154**
- Add Cell Value Rule, 153**
- Add Color Scale Rule, 154**
- Add Data Bar Rule, 154-155**
- Add Date Occurring Rule, 153-154**
- Add Expression Rule, 157**
- Add Icon Sets Rule, 156**
- Add Slicer in Pivot Table, 122-124**
- Add Slicer in Table, 121-122**
- Add Top Bottom Rule, 155-156**
- Add Unique Rule, 156**
- Add Validations, 158-160**
- Allow Sort, 52-55**
- API Reference, 169**
- Apply Comments, 67-69**
- Apply Conditional Formatting, 153**
- Apply Data Validations, 158**
- Apply Filters, 55-57**
- Apply Grouping, 58-59**
- Apply Slicer Style, 124-125**
- Apply Style, 61-62**
- Apply Style To Pivot Table, 117-121**
- Apply Table Sort, 110-111**
- Apply Table Style, 112**
- Apply Theme, 26-27**
- Auto Fit Row Height and Column Width, 42-43**
- Axis and Other Lines, 98-100**
- Chart, 19**
- Chart Area, 84-86**
- Chart Title, 84**

- Comments, 18-19**
- Conditional Format, 19**
- Configure Chart, 83-84**
- Configure Chart Axis, 100-102**
- Configure Chart Series, 90-97**
- Configure Columns to Repeat at Left and Right, 166-167**
- Configure Fonts and Set Style, 129-130**
- Configure Page Breaks, 164-165**
- Configure Page Header and Footer, 162-163**
- Configure Page Settings, 163-164**
- Configure Paper Settings, 165-166**
- Configure Print Area, 166**
- Configure Print Settings via Page Setup, 162**
- Configure Rows to Repeat at Top and Bottom, 167-168**
- Configure Sheet Print Settings, 168**
- Control Pagination, 135**
- Create and Delete Chart, 83**
- Create and Delete Tables, 108-109**
- Create and Set Custom Named Style, 65-67**
- Create Pivot Table, 114-115**
- Create Row or Column Group, 59-60**
- Create Workbook, 23**
- Customize Chart Objects, 86-87**
- Customize User Interaction, 23**
- Customize Worksheets, 49-50**
- Cut or Copy Across Sheets, 25**
- Cut or Copy Cell Ranges, 33-35**
- Cut or Copy Shape, Slicer, Chart and Picture, 35-37**
- Data Label, 103-104**
- Data Validation, 19-20**
- Delete Blank Pages From Middle, 139-140**
- Delete Validation, 160**
- Enable or Disable Calculation Engine, 25-26**
- End User License Agreement, 16-17**
- Export Custom Page Information, 142-143**
- Export Different Headers On Different Pages, 140-141**

- Export Last Page Without Headers, 141-142**
- Export Multiple Sheets To One Page, 138-139**
- Export Pivot Table Styles And Format, 130-132**
- Export Shapes, 132-133**
- Export Specific Pages To PDF, 143-144**
- Export to a PDF File, 128-129**
- Export Vertical Text, 133-134**
- Features, 18**
- Find and Replace Data, 37-38**
- Floor, 102-103**
- Formula, 20**
- Freeze Panes in a Worksheet, 44-45**
- GcExcel .NET Overview, 6**
- Get Row and Column Count, 38-39**
- Get Started, 9-13**
- Group, 20**
- Hide Rows and Columns, 39**
- Hyperlinks, 20**
- Import and Export .xlsx Document, 127-128**
- Import and Export CSV File, 146-148**
- Import and Export CSV File with Delimiters, 148-149**
- Import and Export JSON Stream, 149-151**
- Import and Export Macros, 151-152**
- Import and Export OLE Objects, 152**
- Insert And Delete Cell Ranges, 39-40**
- Insert and Delete Rows and Columns, 40-41**
- Keep Rows Together Over Page Breaks, 139**
- Key Features, 7-8**
- Legends, 104-105**
- License Information, 13-16**
- Manage Custom Functions, 77**
- Manage Data, 83**
- Manage File Operations, 127**
- Manage Formulas, 73**
- Manage Hyperlinks, 57-58**
- Manage Workbook, 23**

- Manage Worksheet, 27-28**
- Merge Cells, 41-42**
- Modify Slicer with Custom Style, 125**
- Modify Table Layout, 113-114**
- Modify Table Layout for Slicer Style, 125-126**
- Modify Table with Custom Style, 112-113**
- Modify Tables, 109-110**
- Modify Validation, 160-161**
- Open, Save and Protect Workbook, 23-25**
- Page Setup, 20**
- PDF Export, 20**
- Pivot Table Settings, 115-117**
- Plot Area, 86**
- Range Operations, 30-31**
- Redistribution, 16**
- Remove a Group, 60-61**
- Render Excel Range Inside PDF, 135-138**
- Rich Text, 21**
- Save Multiple Workbooks to Single PDF, 144-145**
- Series, 87-90**
- Set Array Formula, 75-76**
- Set Formula to Range, 73-74**
- Set Rich Text in a Cell, 69-72**
- Set Row Height and Column Width, 42**
- Set Sheet Styling, 62-65**
- Set Summary Row, 61**
- Set Table Filters, 111**
- Set Table Formula, 74-75**
- Set Values to a Range, 42**
- Shape and Picture, 21**
- Shrink To Fit With Text Wrap, 134-135**
- Sparkline, 21**
- Style, 21**
- Table, 22**
- Technical Support, 16**
- Theme, 21-22**

Upgrade to Latest Version, 16

Use Chart, 83

Use Do Filter Operation, 124

Use Pivot Table, 114

Use Slicer, 121

Use Sparkline, 105-108

Use Table, 108

Walls, 97-98

Work With Custom Functions, 77-82

Work with Shape And Picture, 45-49

Work with Used Range, 44

Work with Workbook Views, 27

Work with Worksheet Views, 50-52

Work with Worksheets, 28-30

Working With Page Setup, 145-146