

## Table of Contents

Table of Contents	1-8
GcExcel .NET Overview	9
Key Features	10-11
Getting Started	12-15
Quick Start	15-18
License Information	18-21
Upgrade to Latest Version	21-22
Technical Support	22
Redistribution	22
End User License Agreement	22
Features	23-24
Worksheet	24
Work with Worksheets	24-29
Range Operations	29-30
Access a Range	30-31
Access Areas in a Range	31
Get Special Cell Ranges	31-36
Access Cells, Rows and Columns in a Range	36-37
Get Address of Cell Range	37-38
Cut or Copy Cell Ranges	38-41
Cut or Copy Shape, Slicer, Chart and Picture	41-42
Find and Replace Data	42-44
Get Row and Column Count	44-45
Hide Rows and Columns	45
Insert And Delete Cell Ranges	45-47
Insert and Delete Rows and Columns	47-48
Merge Cells	48
Set Values to a Range	48-49
Set Row Height and Column Width	49
Auto Fit Row Height and Column Width	49-51
Work with Used Range	51-52
Measure Digital Width	52-53

Freeze Panes in a Worksheet	53-54
Freeze Trailing Panes in a Worksheet	54-55
Customize Worksheets	55-57
Worksheet Views	57-59
Cell Types	59-62
Range Template Cell	62-66
Quote Prefix	66-67
Tags	67-70
Rich Text	70-74
Workbook	74
Create Workbook	74-75
Open and Save Workbook	75-76
Protect Workbook	76-78
Cut or Copy Across Sheets	78-79
Enable or Disable Calculation Engine	79-80
Workbook Views	80
Comments	80-83
Hyperlinks	83-85
Sort	85-88
Filter	89-91
Group	91
Create Row or Column Group	91-92
Remove a Group	92-93
Summary Row	93-94
Outline Subtotals	94-96
Outline Column	96-100
Conditional Formatting	100-101
Cell Value Rule	101
Date Occurring Rule	101-102
Average Rule	102
Color Scale Rule	102-103
Data Bar Rule	103
Top Bottom Rule	103-104

Unique Rule	104-105
Icon Sets Rule	105
Expression Rule	105-106
Data Validations	106
Add Validations	106-109
Delete Validation	109
Modify Validation	109-110
Data Binding	110-114
Digital Signatures	114-124
Formulas	124-125
Formula Parser	125-131
Formula Functions	131-149
Set Formula to Range	149-151
Set Table Formula	151-153
Set Array Formula	153-154
Precedents and Dependents	154-157
Iterative Calculation	157-158
Cross Workbook Formula	158-159
Custom Functions	159-168
Shapes and Pictures	168-171
Customize Shape Format and Shape Text	171-176
Hyperlink on Shape	176-178
Group or Ungroup Shapes	178-180
Shape Adjustment	180-181
Background Image	181-182
Size and Position of Image	182-183
Image Transparency	183
Control Position of Overlapping Shapes	183-184
Styles	184-185
Set Sheet Styling	185-189
Create and Set Custom Named Style	189-191
Barcodes	191
QRCode	191-194

EAN-13	194-196
EAN-8	196-198
Codabar	198-201
Code39	201-203
Code93	203-205
Code128	206-208
GS1-128	208-210
Code49	210-212
PDF417	212-214
Data Matrix	214-217
Theme	217-218
Chart	218-219
Create and Delete Chart	219-220
Configure Chart	220
Chart Title	220-221
Chart Area	221-222
Plot Area	222-223
Customize Chart Objects	223-224
Series	224-228
Configure Chart Series	228-238
Error Bars	238-243
Walls	243
Axis and Other Lines	243-245
Configure Chart Axis	245-248
Floor	248-249
Data Label	249-251
Legends	251-252
Chart Types	252-254
Area Chart	254-256
Bar Chart	256-258
Column Chart	258-260
Combo Chart	260-262
Line Chart	262-264

Pie Chart	264-266
Stock Chart	266-269
Surface Chart	269-270
XY (Scatter) Chart	270-273
Radar Chart	273-274
Statistical Chart	274-275
Box Whisker	275-276
Histogram	276-277
Waterfall Chart	277-279
Pareto Chart	279-280
Specialized Chart	280
Sunburst	280-281
TreeMap	281-283
Funnel	283-284
Chart Sheet	284-286
Table	286-287
Create and Delete Tables	287
Modify Tables	287-289
Table Sort	289
Table Filters	289-290
Add and Delete Table Columns and Rows	290-292
Table Style	292
Modify Table with Custom Style	292-293
Modify Table Layout	293-294
Pivot Table	294-295
Create Pivot Table	295-296
Pivot Table Settings	296-302
Pivot Table Style	302-307
Pivot Chart	307-310
Sparkline	310-315
Slicer	315
Add Slicer in Table	315-316
Add Slicer in Pivot Table	316-318

Use Do Filter Operation	318-319
Slicer Style	319-320
Modify Slicer with Custom Style	320
Modify Table Layout for Slicer Style	320-321
Print Settings	321
Configure Page Header and Footer	321-322
Configure Page Settings	322-324
Configure Page Breaks	324-325
Configure Paper Settings	325-326
Configure Print Area	326-327
Configure Columns to Repeat at Left and Right	327-328
Configure Rows to Repeat at Top and Bottom	328
Configure Sheet Print Settings	328-329
Configure Paper Source	329-330
Logging	330-336
Templates	337-340
Template Configuration	340-342
Template Fields	342-345
Template Properties	345-352
Cell Expansion	352
Cell Context	352-356
Conditional Formatting	356-358
Global Settings	358-361
Fixed Layout	362-365
Default Values in Template Cells	365-366
PDF Form Builder	366-379
Custom Form Input Types	379-384
Charts	384-389
Tables	389-391
Sparklines	391-393
Data Source Binding	393-397
Create Excel Report using Template	397-402
File Operations	403

Import and Export .xlsx Document	403-405
Export to PDF	405-407
Configure Fonts and Set Style	407-408
Export Pivot Table Styles And Format	408-410
Export Shapes	410-411
Export Vertical Text	411-412
Shrink To Fit With Text Wrap	412-413
Control Pagination	413-414
Render Excel Range Inside PDF	414-417
Export Multiple Sheets To One Page	417-418
Keep Rows Together Over Page Breaks	418-419
Delete Blank Pages From Middle	419-420
Export Different Headers On Different Pages	420-421
Export Last Page Without Headers	421-422
Export Custom Page Information	422-423
Export Specific Pages To PDF	423-424
Save Multiple Workbooks to Single PDF	424-426
Export Worksheet to PDF	426-428
Working With Page Setup	428-429
Support Security Options	429-431
Support Document Properties	431-432
Adjust Column Width and Row Height	432
Export Charts	432-437
Export Slicers	437-438
Export Barcodes	438-439
Export Signature Lines	439-440
Support Sheet Background Image	440-441
Support Background Color Transparency	442
Control Image Quality	442-443
Track Export Progress	443-445
Export to HTML	445-449
Import and Export CSV File	449-451
Import and Export CSV File with Delimiters	451-453

Import and Export JSON Stream	453-456
Import and Export from JSON string	456
Import and Export SpreadJS Files	456-468
Support for SpreadJS Features	468-469
Import and Export Macros	469-470
Import and Export OLE Objects	470-471
Convert to Image	471-475
API Reference	476
Release Notes	477
Release Notes for Version 4.1.0	477-478
Release Notes for Version 4.0.0	478-479
Release Notes for Version 3.2.0	479
Release Notes for Version 3.1.0	479-480
Release Notes for Version 3.0.0	480-481
Release Notes for Version 2.2.0	481-482
Release Notes for Version 2.1.0	482
Release Notes for Version 2.0.0	482-483
Release Notes for Version 1.5.0.4	483
Release Notes for Version 1.5.0.3	483-484
Release Notes for Version 1.5.0.1	484
Release Notes for Version 1.4.0	484-485
Index	486-492



## 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, summarize 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 spreadsheets 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 from modification or encrypt 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**

You can create tables and pivot tables to automatically calculate the count, total or average of data in the spreadsheets. You can also rename pivot table fields, manage grand total visibility settings and change row Axis layout of pivot field.

- **Export to PDF**

Using the export to PDF feature, users can save spreadsheets to PDF files with different page settings, features, document properties and security options. You can also export Excel sheets with charts, slicers and sheet background images.

- **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.

- **Shapes and Pictures**

With GcExcel API, you can insert and customize shapes and pictures on cells of a worksheet, apply formatting, gradient fill, configure text, insert hyperlinks, set adjustment points of the shapes, group/ungroup them in a worksheet and determine the position and size of an image.

- **Use Templates to create custom Excel reports**

GcExcel provides templates with comprehensive API to create custom Excel reports with advanced layouts. You can use multiple data sources to bind the data. The templates provide flexible syntax, easy notations and extended reusability making it an ideal solution to generate Excel reports.

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

## Getting Started

### 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. To work with GcExcel .NET, you need to have following references in your application:

Reference	Purpose
<a href="#">GrapeCity.Documents.Excel</a>	To use GcExcel in an application, you need to reference (install) just the GrapeCity.Documents.Excel package. It pulls in the required infrastructure packages.
<a href="#">GrapeCity.Documents.Common</a>	GrapeCity.Documents.Common is an infrastructure package used by other packages. You do not need to reference it directly.

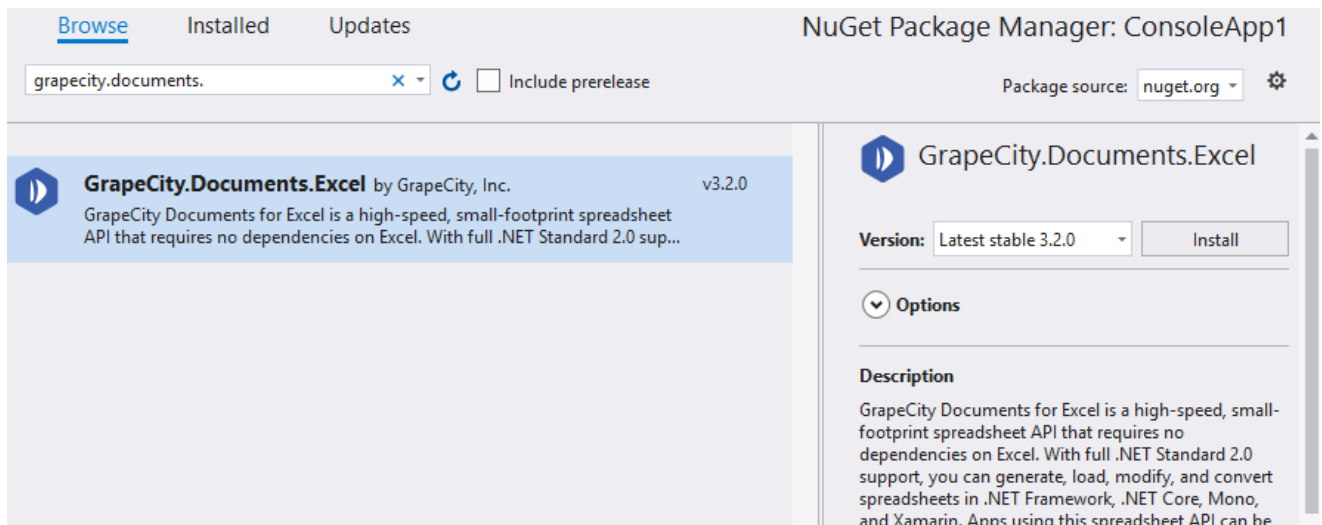
#### Add reference to GcExcel .NET in your application

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

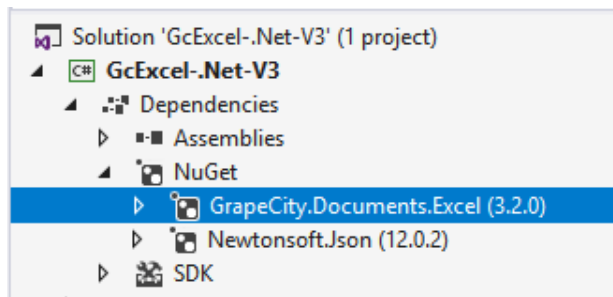
## VISUAL STUDIO FOR WINDOWS

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

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. Create a console application 'myApp' by using the command: `dotnet new console -o myApp`
3. Use the `cd` command to navigate to your project folder: `cd myApp`
4. 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">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="GrapeCity.Documents.Excel" Version="3.2.0" />
  </ItemGroup>

</Project>
```

After this step, follow the steps in the [Quick Start](#) section.

## VISUAL STUDIO FOR MAC

---

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

This automatically adds references of the package and its dependencies to your application. After this step, follow the steps in the [Quick Start](#) section.

## VISUAL STUDIO FOR LINUX

---

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

This adds references of the package to your application. After this step, follow the steps in the [Quick Start](#) section.

## Quick Start

The following quick start section helps you in getting started with the GcExcel library:

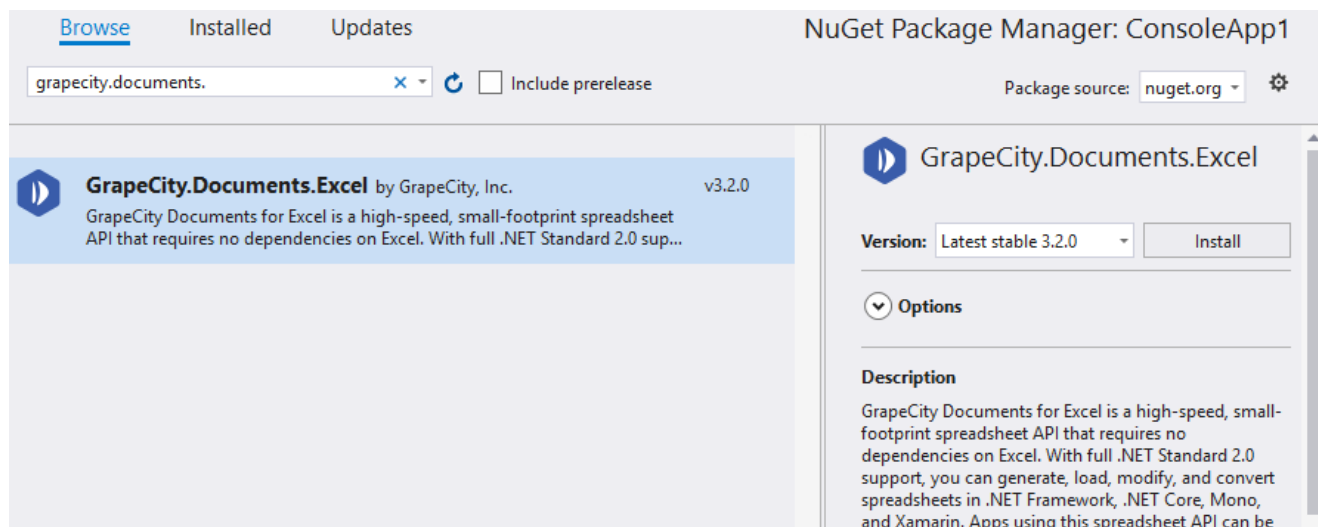
## .NET CORE CONSOLE APPLICATION

Follow the below steps to create a simple .NET Core Console application:

- Step 1: Create a new Console App (.NET Core)
- Step 2: Create and save a new workbook
- Step 3: Build and Run the Project

### Step 1: Create a new Console App (.NET Core)

1. In Visual Studio, select **File | New | Project** to create a new ASP.NET Core Console Application.
2. From the 'New Project' dialog, select **Installed | Visual C# | .NET Core | Console App (.NET Core)**, and click **OK**.
3. 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 'grapecity.documents', select **GrapeCity.Documents.Excel**, and click **Install**.



### Step 2: Create and save a new workbook

1. In Program.cs, include the following namespace using GrapeCity.Documents.Excel;
2. Create a new workbook using the **Workbook** class, add a new worksheet to it and save the workbook using the **Save** method of workbook class.

#### Program.cs

```
Workbook workbook = new Workbook();  
workbook.Worksheets[0].Range["A1"].Value = "Hello Word!";  
workbook.Save("HelloWord.xlsx");
```

### Step 3: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.
3. Once the project is executed, a console window is displayed and **HelloWord.xlsx** file is created at the specified location.



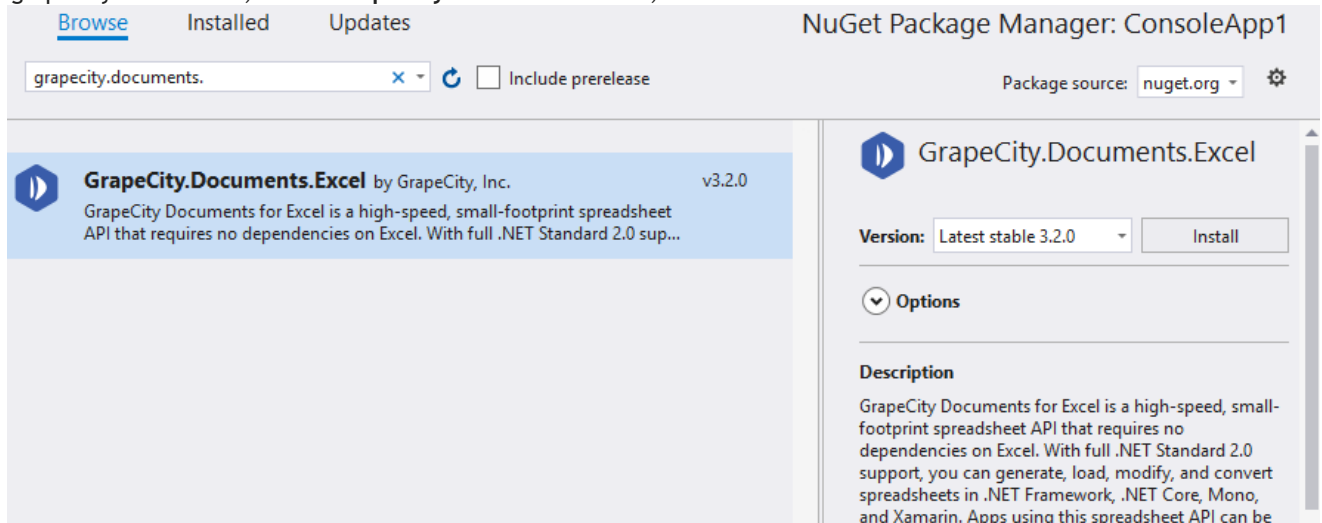
## .NET CORE MVC APPLICATION

Follow the below steps to create a simple .NET Core MVC Application:

- 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. From the 'New Project' dialog, select **Installed | Visual C# | .NET Core | ASP.NET Core Web Application**, and click **OK**.
3. In the 'New ASP.NET Core Web Application(.NET Core)' dialog, select **Web Application (Model-View-Controller)**, and 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 'grapecity.documents', select **GrapeCity.Documents.Excel**, 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 | Controller**. The **Add Scaffold** dialog appears.
3. Complete the following steps in the **Add Scaffold** dialog:
  - Select **MVC Controller-Empty** and click **Add**.
  - Set name of the MVC controller (For example: GcExcelController) and click **Add**.
4. Add the GcExcel reference in Controller file:  
**using GrapeCity.Documents.Excel;**
5. In the **Index()** method of the Controller, add the following code:

GcExcelController.cs

```
public IActionResult Index()
{
    Workbook workbook = new Workbook();
    workbook.Worksheets[0].Range["A1"].Value = "Hello Word!";
    workbook.Save("HelloWord.xlsx");

    return View();
}
```

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

## Step 3: Add a View

1. From the **Solution Explorer**, right click the folder **Views** and select **Add | New Folder**.
2. Name the new folder. Provide the same name as the name of your controller, minus the suffix Controller (in our example: GcExcel).
3. Right click the folder GcExcel, and select **Add | View**. The **Add MVC View** dialog appears.
4. Complete the following steps in the **Add MVC View** dialog:
  - Set View name same as the Action name, Index (for example: Index.cshtml).
  - Click **Add**.
5. Replace the code in Index.cshtml file with below:

```
Index.cshtml

@{
    ViewData["Title"] = "Documents for Excel, .Net Edition";
}
<script>
    onload = function () {
        alert("File Saved: HelloWorld.xlsx");
    }
</script>
```

## Step 4: Build and Run the Project

1. Click **Build | Build Solution** to build the project.
2. Press **F5** to run the project.
3. Once the project is executed, access the URL: <http://localhost:1234/GcExcel/Index> to generate the Excel file. An alert box is displayed and **HelloWord.xlsx** file is created at the specified location.

## License Information

### 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, up-to 100 Excel files can be opened or saved 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.

**This topic  
includes:**

**Types of Licenses  
Apply License**

- 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	Remark
IRange	Throws an exception, same as create an instance of Workbook.
IWorkbook.Worksheets.Add()	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](mailto: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](mailto: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](mailto: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 apply evaluation/production license in GcExcel .NET, the long string key needs to be copied to the code in one of the following two ways.

### .NET CORE CONSOLE APPLICATION

- **To license all the workbooks in a project**

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

- **To license an instance of the workbook**

```
C#
var workbook = new Workbook("Your License Key");
```

## .NET CORE MVC APPLICATION

- To license all the workbooks in a project, add the license key in Startup.cs file by using SetLicenseKey method. This will license all the workbooks even across multiple Controllers.

### Startup.cs

```
public Startup(IConfiguration configuration)
{
    //Apply license before using the API, otherwise it will be considered
    as no license.
    Workbook.SetLicenseKey(" Your License Key");
}
```

- To license an instance of the workbook, add the license key when an instance of workbook is created. Add the following code in the Index() method of the controller:

### GcExcelController.cs

```
public IActionResult Index()
{
    //Apply license before using the API, otherwise it will be considered
    as no license.
    var workbook = new Workbook("Your License Key");
}
```

## Upgrade to Latest Version

Follow one of the below to upgrade your GcExcel.NET license to the latest version:

- If you are using the current major version, the existing license key is still valid.
- If you are upgrading from a previous major version, a new license key will be needed.

The new license key must have been provided if you are under current Maintenance. But in case the Maintenance has expired, please contact [us.sales@grapecity.com](mailto:us.sales@grapecity.com) to purchase the upgrade.

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](mailto: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	<a href="https://www.grapecity.com/">https://www.grapecity.com/</a>
E-mail	<a href="mailto:us.sales@grapecity.com">us.sales@grapecity.com</a>
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

This section comprises the features available in GcExcel.

### Worksheet

Work with cells, range and basic worksheet operations.

### Workbook

Work with basic workbook operations.

### Comments

Add or delete comments, show or hide comments, set rich text, comment layout or author of a comment.

### Hyperlinks

Add, configure or delete hyperlinks.

### Sort

Apply sorting and its various types.

### Filter

Apply filtering and its several types.

### Group

Apply grouping over data in rows or columns.

### Conditional Formatting

Apply conditional formatting in a cell or range of cells.

### Data Validations

Add, modify and delete data validations.

### Data Binding

Bind data to sheet, cell or table columns.

### Digital Signatures

Add, countersign, verify and delete digital signatures.

### Formulas

Use formulas to carry complex calculations.

### Custom Functions

Create custom functions to implement custom arithmetic logic.

### Shapes and Pictures

Work with shapes and pictures in a worksheet.

### Styles

Set styles to format cell appearance.

### Barcodes

Supports 11 types of barcodes in worksheet, their JSON I/O and export to PDF.

### Theme

Apply built-in or custom themes to change the workbook appearance.

### Chart

Work with charts to display data graphically.

### Table

Use tables to organize large amount of data efficiently.

### Pivot Table

Use pivot table to perform analysis of complex information and summarize data.

### Pivot Chart

Use pivot chart to display pivot table's data graphically.

### Sparkline

Use sparklines to insert graphical illustration of trends in data.

### Slicer

Use slicers to perform quick filtration of data in tables and pivot tables.

## Print Settings

Configure print settings to manage printing options.

## Logging

Capture debug, error, information and warning logs to locate issues and finding out their root cause.

## 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** 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](#)
- [Freeze Trailing Panes in a Worksheet](#)
- [Customize Worksheets](#)
- [Worksheet Views](#)
- [Cell Types](#)
- [Quote Prefix](#)
- [Tags](#)
- [Rich Text](#)

## 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

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);
```



**Note:** The **Add** method in **IWorksheets** interface has an overload with **SheetType**, which lets you add two types of sheets, Worksheet or Chart sheet. By default, this method adds a Worksheet in the Workbook.

## 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** method of the **IWorksheet interface**.

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** or using the **Name property** of the **IWorksheet interface**.

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

A worksheet can be protected by transforming it into a read-only sheet so that the data lying in the cells cannot be modified. The worksheet can be prevented from modification either by using a password or without it.

### Protect worksheet from modification without password

A worksheet can be protected by setting the **Protection** property of the **IWorksheet** interface to true. Further, you can

use the properties of the **IProtectionSettings** 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 from modification without password.

C#

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

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

## Protect Worksheet from Modification using Password

A worksheet can be made password protected to restrict modification by using the **Protect** method of **IWorksheet** interface. The password is a case sensitive string which can be passed as a parameter to the **Protect** method.

Refer to the following example code to protect a worksheet from modification using password.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet workSheet = workbook.Worksheets[0];

//Protects the worksheet with password
workSheet.Protect("Ygs_87@ytr");
// Save workbook to xlsx
workbook.Save(@"ProtectWorksheetWithPassword.xlsx", SaveFileFormat.Xlsx);
```

A password protected worksheet can be unprotected by using the **Unprotect** method of **IWorksheet** interface. The correct password (password set in **Protect** method) needs to be passed as a parameter to the **Unprotect** method. In case, the password is omitted or an incorrect password is passed, an exception message "Invalid Password" is thrown.

Refer to the following example code to unprotect a worksheet from modification using password.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet workSheet = workbook.Worksheets[0];

workSheet.Protect("Ygs_87@ytr");
//Removes the above protection from the worksheet
workSheet.Unprotect("Ygs_87@ytr");
// Save workbook to xlsx
workbook.Save(@"UnprotectWorksheetWithPassword.xlsx", SaveFileFormat.Xlsx);
```

## 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()** method, the **CopyAfter()** method, the **CopyBefore()** method, the **Move()** method, the **MoveBefore()** method and the **MoveAfter()** 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);
```

## Select Multiple Worksheets

GcExcel allows you to select multiple worksheets at once by using **Select** method of **IWorksheets** interface. The method takes an optional parameter **replace**, which:

- Replaces the current selection with the specified object when set to True (default value).
- Extends the current selection to include any previously selected objects and the specified object when set to False.

The selected worksheets can also be retrieved by using **SelectedSheets** property of **IWorkbook** interface. In addition, Excel files with multiple selected worksheets can be loaded, modified and saved back to Excel. GcExcel displays following behavior when multiple worksheets are selected:

- If a non-selected sheet is activated, the selected sheets are deselected.

- If a selected sheet is deleted, it is removed from selected sheets.
- If all worksheets are selected and a worksheet is activated, it is set as the active sheet and all selected sheets are deselected.
- If a worksheet is added, copied or moved, the selected sheets are deselected.

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

C#

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

var sheet1 = workbook.ActiveSheet;
var sheet2 = workbook.Worksheets.Add();
var sheet3 = workbook.Worksheets.Add();

// Select sheet2 and sheet3
workbook.Worksheets[new[] { sheet2.Name, sheet3.Name }].Select();

// Write names of selected sheets to console
foreach (var sheet in workbook.SelectedSheets)
{
    Console.WriteLine(sheet.Name);
}

// Add sheet1 to selected sheets
sheet1.Select(replace: false);

// Write count of selected sheets to console
Console.WriteLine(workbook.SelectedSheets.Count);

//save to an excel file
workbook.Save("selectworksheets.xlsx");
```

## Limitation

A valid license is required to select multiple worksheets in a workbook. Otherwise, the evaluation warning sheet will overwrite the sheet selection and active sheet.

## 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** property of **IWorksheet** 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](#)
- [Get Address of Cell 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 Custom Objects to a Range**
- [Set Row Height and Column Width](#)
- [Auto Fit Row Height and Column Width](#)
- [Work with Used Range](#)
- [Measure Digital Width](#)

## 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** interface. The **Count** property of the **IAreas** interface represents the area count (number of areas) of the multiple-area range.

The **Areas** property of the **IRange** 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;
```

## Get Special Cell Ranges

Special cell ranges refer to the ranges containing specified data type or values. For example, cells containing comments, text values, formulas, blanks, constants, numbers etc.

GcExcel allows you to get special cell ranges by using **SpecialCells** method of **IRange** interface. It takes the following enumerations as parameters:

- **SpecialCellType**: Specifies the type of cells like formula, constant, blank etc.
- **SpecialCellsValue**: Specifies cells with a particular type of value like numbers, text values etc.

## Find Special Cell Ranges by Type

Refer to the following example code to find the range of special cells by specifying the type of cells.

```
C#
//create a new workbook
var workbook = new Workbook();

IWorksheet ws = workbook.ActiveSheet;

//prepare data
var rngA1D2 = new object[,] {
    { "Register", null, null, null},
    { "Field name", "Wildcard", "Validation error", "User input"}
};
ws.Range["$A$1:$D$2"].Value = rngA1D2;

var rngA3C6 = new object[,] {
    { "User name", "??*", "At least 2 characters"},
    { "Captcha", "?????", "5 characters required"},
    { "E-mail", "?*@?*.?*", "The format is incorrect"},
    { "Security code", "#####", "7 digits required"}
};
ws.Range["$A$3:$C$6"].Value = rngA3C6;

var rngA8D14 = new object[,] {
    { "User table", null, null, null},
    { "Id", "Name", "Email", "Banned"},
    { 1d, "User 1", "8zgnvlp2@163.com", true},
    { 2d, "User 2", "b9fvaswb@163.com", false},
    { 3d, "User", "md78b", false},
    { 4d, "User 4", "lqasghjfg@163.com", false},
    { 5d, "U", "mncx23k8@163.com", false}
};
ws.Range["$A$8:$D$14"].Value = rngA8D14;

ws.Range["A1:D1"].Merge();
ws.Range["A1:D1"].HorizontalAlignment = HorizontalAlignment.Center;
ws.Range["A8:D8"].Merge();
ws.Range["A8:D8"].HorizontalAlignment = HorizontalAlignment.Center;

ws.Range["D3"].AddComment("Required");
ws.Range["D4"].AddComment("Required");
ws.Range["D5"].AddComment("Required");
ws.Range["D6"].AddComment("Required");

ws.Range["D10:D14"].Validation.Add(
    ValidationType.List, ValidationAlertStyle.Stop,
    ValidationOperator.Between, "True,False");

var condition = (IFormatCondition)ws.Range["C10:C14"].FormatConditions.Add(
    FormatConditionType.Expression, formula1: "=ISERROR(MATCH($B$5,C10,0))");
condition.Font.Color = Color.Red;

var condition2 = (IFormatCondition)ws.Range["B10:B14"].FormatConditions.Add(
    FormatConditionType.Expression, formula1: "=LEN(B10)<=2");
condition2.Font.Color = Color.Red;

ws.Range["4:4"].EntireRow.Hidden = true;

IRange searchScope = ws.Range["1:14"];

// Find comments
var comments = searchScope.SpecialCells(SpecialCellType.Comments);
```



```
// Find last cell
var lastCell = searchScope.SpecialCells(SpecialCellType.LastCell);

// Find visible
var visible = searchScope.SpecialCells(SpecialCellType.Visible);

// Find blanks
var blanks = searchScope.SpecialCells(SpecialCellType.Blanks);

// Find all format conditions
var allFormatConditions = searchScope.SpecialCells(SpecialCellType.AllFormatConditions);

// Find all validation
var allValidation = searchScope.SpecialCells(SpecialCellType.AllValidation);

// Find same format condition as B10
var sameFormatConditions = ws.Range["B10"].SpecialCells(SpecialCellType.SameFormatConditions);

// Find same validation as D10
var sameValidation = ws.Range["D10"].SpecialCells(SpecialCellType.SameValidation);

// Find merged cells
var merged = searchScope.SpecialCells(SpecialCellType.MergedCells);

// Output
ws.Range["A16"].Value = "Find result";
ws.Range["A16:C16"].Merge();
ws.Range["A16:C16"].HorizontalAlignment = HorizontalAlignment.Center;
ws.Range["$A$17:$A$25"].Value = new object[,]{
    {"Comments"},
    {"LastCell"},
    {"Visible"},
    {"Blanks"},
    {"AllFormatConditions"},
    {"AllValidation"},
    {"SameFormatConditions B10"},
    {"SameValidation D10"},
    {"MergedCells"}
};
ws.Range["$C$17:$C$25"].Value = new object[,]{
    {comments.Address},
    {lastCell.Address},
    {visible.Address},
    {blanks.Address},
    {allFormatConditions.Address},
    {allValidation.Address},
    {sameFormatConditions.Address},
    {sameValidation.Address},
    {merged.Address}
};

ws.UsedRange.EntireColumn.AutoFit();

//save to an excel file
workbook.Save("specialcellsfindmiscellaneous.xlsx");
```

### Find Special Cells by Type in Existing File

Refer to the following example code to load an existing file, find special cells containing formulas and constants and change their background color.

```
C#

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

workbook.Open("FinancialReport.xlsx");

IRange cells = workbook.ActiveSheet.Cells;

// Find all formulas
var allFormulas = cells.SpecialCells(SpecialCellType.Formulas);
```

```
// Find all constants
var allConstants = cells.SpecialCells(SpecialCellType.Constants);

// Change background color of found cells
allFormulas.Interior.Color = Color.LightGray;
allConstants.Interior.Color = Color.DarkGray;

//save to an excel file
workbook.Save("specialcellsinexistingfiles.xlsx");
```

## Find Special Cells by Type and Values

Refer to the following example code to find special cells by specifying cell type and values.

```
C#

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

IWorksheet ws = workbook.ActiveSheet;

// Set data
ws.Range["A1"].Formula = "=\"Text \" & 1";
ws.Range["B1"].Formula = "=8*10^6";
ws.Range["C1"].Formula = "=SEARCH(A1,9)";
ws.Range["A2"].Value = "Text";
ws.Range["B2"].Value = 1;

// Find text formulas
var textFormula = ws.Cells.SpecialCells(SpecialCellType.Formulas, SpecialCellsValue.TextValues);

// Find number formulas
var numberFormula = ws.Cells.SpecialCells(SpecialCellType.Formulas, SpecialCellsValue.Numbers);

// Find error formulas
var errorFormula = ws.Cells.SpecialCells(SpecialCellType.Formulas, SpecialCellsValue.Errors);

// Find text values
var textValue = ws.Cells.SpecialCells(SpecialCellType.Constants, SpecialCellsValue.TextValues);

// Find number values
var numberValue = ws.Cells.SpecialCells(SpecialCellType.Constants, SpecialCellsValue.Numbers);

// Display search result
ws.Range["A4:E5"].Value = new object[,] {
    { "Text formula", "Number Formula", "Error Formula", "Text Value", "Number Value"},
    { textFormula.Address, numberFormula.Address, errorFormula.Address, textValue.Address, numberValue.Address}
};

ws.UsedRange.EntireColumn.AutoFit();

//save to an excel file
workbook.Save("specialcellsquickstart.xlsx");
```

Refer to the following example code to find special cell ranges by cell type and values. The formatting of cells is defined to easily distinguish between different types of special cells.

```
C#

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

IWorksheet ws = workbook.ActiveSheet;

//prepare data
ws.Range["$A$1:$F$1"].Value = new object[,] {
    { "Test id", "Group id", "Group item id", "New test id", "Test result", "Error code"}
};

ws.Range["$B$2:$C$2"].Value = 1d;
```

```
ws.Range["$E$2,$E$7,$E$12,$E$21,$E$27,$E$36,$E$40,$E$47:$E$48,$E$51,$E$59:$E$60,$E$70:$E$71,$E$80:$E$81,$E$88,$E$90:$E$91"].Value
= "Error 80073cf9";
ws.Range["$G$1:$G$2,$I$1:$I$7,$H$8:$I$8,$A$93:$B$93,$E$93:$F$93"].Value = null;

ws.Range["$H$1:$H$7"].Value = new object[,]{
{ "Constants"}, { "Formulas"}, { "String constants"}, { "Number constants"}, { "String formulas"}, { "Number formulas"}, { "Error
formulas"}
};

ws.Range["$A$2:$A$13"].Value = "Test00001";
ws.Range["$A$14:$A$67"].Value = "Test00153";
ws.Range["$A$68:$A$92"].Value = "Test05789";
ws.Range["$E$3:$E$5,$E$9:$E$11,$E$25:$E$26,$E$37:$E$38,$E$57,$E$75:$E$76,$E$86:$E$87"].Value = "Runtime Error c0000005";
ws.Range["$E$6,$E$13:$E$20,$E$28:$E$35,$E$41:$E$46,$E$52:$E$56,$E$61:$E$64,$E$72:$E$74,$E$77:$E$78,$E$82:$E$85,$E$89,$E$92"].Value
= "Passed";
ws.Range["$E$8,$E$22:$E$24,$E$39,$E$49:$E$50,$E$58,$E$65:$E$69,$E$79"].Value = "Deploy Error 80073cf9";

ws.Range["$D$2:$D$92"].FormulaR1C1 = "=\"X-Test-G\" & RC[-2] & \"-I\" & RC[-1]";
ws.Range["$B$3:$B$92"].FormulaR1C1 = "=IF(RC[-1]=R[-1]C[-1],R[-1]C,R[-1]C+1)";
ws.Range["$C$3:$C$92"].FormulaR1C1 = "=IF(RC[-2]=R[-1]C[-2],R[-1]C+1,1)";
ws.Range["$F$2:$F$92"].FormulaR1C1 = "=MID(RC[-1], SEARCH(\"Error \",RC[-1])+6,8)";

Color constantBgColor;
Color formulasBgColor;
Color stringForeColor;
Color errorForeColor;
unchecked
{
    constantBgColor = Color.FromArgb((int)0xFFDDEBF7);
    formulasBgColor = Color.FromArgb((int)0xFFFFF2F2);
    stringForeColor = Color.FromArgb((int)0xFF0000C0);
}
errorForeColor = Color.DarkRed;

var searchScope = ws.Range["$A:$F"];

// Find constant cells and change background color
IRange allConsts = searchScope.SpecialCells(SpecialCellType.Constants);
allConsts.Interior.Color = constantBgColor;

// Find formula cells and change background color
IRange allFormulas = searchScope.SpecialCells(SpecialCellType.Formulas);
allFormulas.Interior.Color = formulasBgColor;

// Find text constant cells and change foreground color
IRange textConsts = searchScope.SpecialCells(
    SpecialCellType.Constants, SpecialCellsValue.TextValues);
textConsts.Font.Color = stringForeColor;

// Find text formula cells and change foreground color
IRange textFormulas = searchScope.SpecialCells(
    SpecialCellType.Formulas, SpecialCellsValue.TextValues);
textFormulas.Font.Color = stringForeColor;

// Find number constant cells and change font weight
IRange numberConsts = searchScope.SpecialCells(
    SpecialCellType.Constants, SpecialCellsValue.Numbers);
numberConsts.Font.Bold = true;

// Find number formula cells and change font weight
IRange numberFormulas = searchScope.SpecialCells(
    SpecialCellType.Formulas, SpecialCellsValue.Numbers);
numberFormulas.Font.Bold = true;

// Find error formula cells and change foreground color and font style
IRange errorFormulas = searchScope.SpecialCells(
    SpecialCellType.Formulas, SpecialCellsValue.Errors);
errorFormulas.Font.Color = errorForeColor;
errorFormulas.Font.Italic = true;
```

```
// Set sample cell styles
ws.Range["$H$1,$H$3,$H$4"].Interior.Color = constantBgColor;
ws.Range["$H$2,$H$5:$H$7"].Interior.Color = formulasBgColor;
ws.Range["$H$3,$H$5"].Font.Color = stringForeColor;
ws.Range["$H$4,$H$6"].Font.Bold = true;
ws.Range["$H$7"].Font.Color = errorForeColor;
ws.Range["$H$7"].Font.Italic = true;

ws.UsedRange.EntireColumn.AutoFit();

//save to an excel file
workbook.Save("specialcellsfindvaluesandformulas.xlsx");
```

### Limitations

When the result contains cell ranges with multiple adjoining rectangles, the merging strategy in GcExcel is different from Excel.

For example, if you find number constants with Excel, the result is \$A\$2:\$C\$3,\$C\$4:\$D\$4

	A	B	C	D
1				
2	1	1	1	
3	1	1	1	
4			1	1
5				

Whereas with GcExcel, the result is \$A\$2:\$B\$3,\$C\$2:\$C\$4,\$D\$4

	A	B	C	D
1				
2	1	1	1	
3	1	1	1	
4			1	1
5				

## Access Cells, Rows and Columns in a Range

You can access cells, rows and columns in a range by using the **Cells** property, **Rows** property and **Columns** property of the **IRange** 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();
```

## Get Address of Cell Range

In GcExcel, the address of cells or their ranges can be retrieved in A1 or R1C1 notation (both absolute and relative references). The read-only **Address** property of **IRange** interface can be used to get the range reference in absolute A1 format. However, you can use the **GetAddress** method of **IRange** interface to define the reference notation and absolute and relative references. It takes 4 optional parameters which when omitted, return the same value as **Address** property.

The below table elaborates how to use GcExcel API members to retrieve the address of cell[0,0] in different notations and references.

Cell Reference Notation	Absolute Reference	Relative Reference
<b>A1</b>	Address property <i>Output: \$A\$1</i>	GetAddress method (set rowAbsolute and columnAbsolute parameters to False) <i>Output: A1</i>
<b>R1C1</b>	GetAddress method (set referenceStyle parameter to R1C1) <i>Output: R1C1</i>	GetAddress method (set referenceStyle parameter to R1C1, rowAbsolute and columnAbsolute parameters to False) <i>Output: RC</i>

Refer to the below example code to retrieve the address of a cell in different notations and references.

```
C#

//create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();
var mc = workbook.Worksheets["Sheet1"].Cells[0,0];
```

```
//get absolute address in A1 notation
Console.WriteLine(mc.Address);

//get row's relative and column's absolute address in A1 notation
Console.WriteLine(mc.GetAddress(rowAbsolute: false));

//get absolute address in R1C1 notation
Console.WriteLine(mc.GetAddress(referenceStyle: ReferenceStyle.R1C1));

//get relative address in R1C1 notation
Console.WriteLine(mc.GetAddress(referenceStyle: ReferenceStyle.R1C1,
    rowAbsolute: false,
    columnAbsolute: false,
    relativeTo: workbook.Worksheets[0].Cells[2, 2]));
```

## 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** method of **IRange**. 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 <b>B3:D12</b> and pastes the data to cell <b>E5</b> onwards.
Copy(sheet.Range["E5:G14"])	This method copies data from cell range <b>B3:D12</b> and pastes the data in cell range <b>E5:G14</b> . 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** 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** method of the **IRange** 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	Description
Cut(sheet.Range["E5"])	This method cuts the data from cell range <b>B3:D12</b> and pastes the data to cell <b>E5</b> onwards.
Cut(sheet.Range["E5:G14"])	This method cuts the data from cell range <b>B3:D12</b> and pastes the data in cell range <b>E5:G14</b> . 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()** method of the **IRange** interface.

In order to perform the cut operation, you can use the **Cut()** method of the **IRange** 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()** method of the **IShape** 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", "catel1");
ISlicer slicer = cache1.Slicers.Add(workbook.Worksheets["Sheet1"], "catel1", "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** method of the **IRange** interface can be used to find the first, next or the previously matched cell range.
- The **Replace** method of the **IRange** 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** property of the **IRange** 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** property of the **IRange** 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** method of **IRange**. 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#
<pre>//Insert the range of cell worksheet.Range["A3"].Insert();  // Insert the range of cells worksheet.Range["A3:A5"].Insert();</pre>

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

## Delete cell range

GcExcel allow you to delete a cell or a range of cells in the worksheets by calling **Delete** method of **IRange**. 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** method of **IRange**.

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** 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** 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** method of **IRange**.

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** method of **IRange**. 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** 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** property and **UseStandardWidth** property of the **IRange** interface respectively.

You can use the **ColumnWidth** 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** 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** property and **RowHeightInPixel** property of the **IRange** interface.

In order to specify custom total height and total width, you can use the **Height** (in points), **HeightInPixel** (in pixels), **Width** (in characters) and **WidthInPixel** (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()** method of the **IRange** interface in order to auto fit row height and column width.
- If the type of the cell range used is a column (**IRange.Columns/IRange.EntireColumn** 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 for columns 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** 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** of the **IWorksheet interface**. After you accomplish this, you can customize the used range using the properties of the **IRange interface**.

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** 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.

## Measure Digital Width

If you want to get or set column width by pixels, the result may appear different in GcExcel and Excel as both of them store column width by characters. To overcome this issue, GcExcel supports measuring digital width in order to calculate the accurate pixel value of a single digit.

GcExcel provides **IGraphicsInfo** interface in its API which can be implemented to know the accurate pixel value of a single digit. The **GetDigitWidth** method in IGraphicsInfo interface measures the text (or characters) based on different font attributes like font family, font size, font style etc. GcExcel API also supports GUI frameworks, such as WPF and Windows Forms.

Refer to the following example code which calculates the pixel value and exports the column width correctly in Excel.

C#

```
class FakeGraphicsInfo : IGraphicsInfo
{
    public double Width { get; set; }
    public int GetDigitWidthCount { get; set; }
    public double GetDigitWidth(TextFormatInfo textFormat)
    {
        GetDigitWidthCount++;
        return Width;
    }
}

private void Form1_Load(object sender, EventArgs e)
{
    var workbook = new Workbook();
    var theme = new Theme("custom");
    theme.ThemeFontScheme.Major[FontLanguageIndex.Latin].Name = "宋体";
    theme.ThemeFontScheme.Minor[FontLanguageIndex.Latin].Name = "宋体";
    workbook.Theme = theme;
    var sheet = workbook.ActiveSheet;
    var fakeGraphicsInfo = new FakeGraphicsInfo { Width = 8 };
    workbook.GraphicsInfo = fakeGraphicsInfo;
    sheet.StandardWidthInPixel = 20;
    sheet.Columns[0].ColumnWidthInPixel = 20;
    sheet.Range[1, 1].Value = "abc";
    workbook.Save("ColumnWidth.xlsx");
}
```



**Note:** Please note below points:

- The above sample requires WinForms application to build and run.
- The result of **GetDigitWidth** method is environment-dependent. Sometimes, it returns different results when running on different computers.

## Freeze Panes in a Worksheet

GcExcel provides the ability to freeze panes in a worksheet. This feature enables users to keep some specific rows or columns visible while 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.

Additionally, it allows to set the custom color of lines of frozen panes. However, these colors are only visible while interacting with SpreadJS by doing JSON I/O and are not visible in Excel or PDF.

### Freeze Panes

You can freeze panes in a worksheet using the **FreezePanes()** method of the **IWorksheet** interface. 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** and **FreezeColumn** 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);
```

You can also set custom color of lines of frozen panes using the **FrozenLineColor** property of **IWorksheet** interface.

Refer to the following example code to set blue color for lines of frozen panes in a worksheet.

C#

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

//Freeze panes
worksheet.FreezePanes(5, 5);

//Set frozen line color as blue
worksheet.FrozenLineColor = Color.Blue;
```

```
//Export workbook to json string and save to ssjson
System.IO.File.WriteAllText("frozenlinecolor.ssjson", workbook.ToJson());
```

### Unfreeze Panes

You can unfreeze the split panes using the **UnfreezePanes()** method of the **IWorksheet** interface.

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

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

## Freeze Trailing Panes in a Worksheet

GcExcel allows users to freeze trailing panes in a worksheet. The trailing panes correspond to the rows and columns at the extreme bottom and right of the worksheet. Hence, it enables users to keep those specific rows or columns visible while scrolling through the rest of the sheet.

However, the frozen trailing panes are only visible while interacting with SpreadJS by doing JSON I/O and are not visible in Excel or PDF.

### Freeze Trailing Panes

The trailing panes in a worksheet can be frozen by using the **FreezeTrailingPanes** method of **IWorksheet** interface which takes row and column positions as parameters. The number of frozen rows and column can also be retrieved by using the **FreezeTrailingRow** and **FreezeTrailingColumn** properties respectively.

Refer to the following example code to freeze trailing panes and retrieve the number of trailing frozen rows and columns in a worksheet.

```
C#
//create a new workbook
var workbook = new Workbook();

//use sheet index to get worksheet
IWorksheet worksheet = workbook.Worksheets[0];

//freeze trailing pane
worksheet.FreezeTrailingPanes(2, 3);

//get the number of frozen trailing rows and columns
Console.WriteLine("Number of trailing rows and columns" + worksheet.FreezeTrailingRow +
worksheet.FreezeTrailingColumn);

System.IO.File.WriteAllText("freezetrailingrowscolumns.ssjson", workbook.ToJson());
```

### Unfreeze Trailing Panes

Similarly, the frozen trailing panes can be unfrozen by using the **UnfreezeTrailingPanels** method of **IWorksheet** interface. Refer to the following example code to unfreeze trailing panes in a worksheet.

```
C#

//create a new workbook
var workbook = new Workbook();

//use sheet index to get worksheet
IWorksheet worksheet = workbook.Worksheets[0];

//freeze trailing pane
worksheet.FreezeTrailingPanels(2, 3);

//unfreeze trailing pane
worksheet.UnfreezeTrailingPanels();

System.IO.File.WriteAllText("unfreezetrailingrowscolumns.ssjson", workbook.ToJson());
```

## Customize Worksheets

GcExcel allows you to customize worksheets using the properties of **IWorksheet Interface**. 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 of the **IWorksheet** interface. 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 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** property of the **IWorksheet** 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** and **StandardWidth** 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;
```

## 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** interface, **IPane** interface and **IWorksheetView** 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
<b>IWorksheet.SplitPanes(int row, int column)</b>	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.
<b>IWorksheet.UnsplitPanes(int row, int column)</b>	This method can be used to unsplit the split panes. Using this method is similar to using <code>IWorksheet.SplitPanes(0,0)</code> .
<b>IWorksheet.SplitRow / IWorksheet.SplitColumn</b>	This method gets the split distances (row count and column count) from top (in case of row) or left (in case of column).
<b>IWorksheet.Panes</b>	A range object that represents the frozen or split panes of the worksheet.
<b>IWorksheet.ActivePane</b>	This property can be used to get the active pane in a worksheet.
<b>IPane.Activate()</b>	This method activates the current pane.
<b>IPane.Index</b>	This property can be used to get the index of the current pane in <code>IWorksheet.Panes</code> .
<b>IPane.ScrollColumn / IPane.ScrollRow</b>	This property can be used to get or set the top left cell position of the current pane.
<b>IWorksheet.SheetView</b>	This property can be used to get the view of the worksheet.
<b>IWorksheetView.Zoom</b>	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.
<b>IWorksheetView.GridlineColor</b>	This property can be used to get and set the gridline color.
<b>IWorksheetView.ScrollColumn</b>	This property can be used to get and set the number of the leftmost column in the worksheet.
<b>IWorksheetView.ScrollRow</b>	This property can be used to get and set the number of the row that appears at the top of the worksheet.
<b>IWorksheetView.DisplayRightToLeft</b>	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.
<b>IWorksheetView.DisplayFormulas</b>	This property can be used to get and set whether the worksheet displays formulas.
<b>IWorksheetView.DisplayGridlines</b>	This property can be used to get and set whether the gridlines are displayed.
<b>IWorksheetView.DisplayVerticalGridlines</b>	This property can be used to get and set whether the vertical gridlines are displayed.
<b>IWorksheetView.DisplayHorizontalGridlines</b>	This property can be used to get and set whether the horizontal gridlines are displayed.
<b>IWorksheetView.DisplayHeadings</b>	This property can be used to get and set whether the headers are displayed.
<b>IWorksheetView.DisplayOutline</b>	This property can be used to get and set whether the outline symbols are displayed.
<b>IWorksheetView.DisplayRuler</b>	This property can be used to get and set whether a ruler is displayed for the specified worksheet.
<b>IWorksheetView.DisplayWhitespace</b>	This property can be used to get and set whether the whitespace is displayed.
<b>IWorksheetView.DisplayZeros</b>	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 SplitPanels() method to split the worksheet into panes.

```
C#
//Split worksheet using SplitPanels() 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;
```

The following code snippet shows how to use the **DisplayVerticalGridlines** and **DisplayHorizontalGridlines** properties to display the vertical and horizontal gridlines of a worksheet. These gridlines are only visible while interacting with SpreadJS by doing JSON I/O and are not visible in Excel or PDF.

```
C#

//create a new workbook
var workbook = new Workbook();

IWorksheet worksheet = workbook.Worksheets[0];

worksheet.Range["A10"].Value = 10;

//Set to not show horizontal gridlines
worksheet.SheetView.DisplayHorizontalGridlines = false;

//Set to show vertical gridlines
worksheet.SheetView.DisplayVerticalGridlines = true;


//Export workbook to json string and save to ssjson
System.IO.File.WriteAllText("gridlines.ssjson", workbook.ToJson());
```

 **Note:** If the value of **DisplayGridlines** is set, **DisplayVerticalGridlines** and **DisplayHorizontalGridlines** are also set to the same value.

## Cell Types

GcExcel supports **Button**, **CheckBox**, **ComboBox**, and **Hyperlink** cell types. These cell types define the type of information in a cell and its behavior.

Cell types can be defined for a cell, range of cells, row, column or a worksheet. GcExcel library provides the **CellType** property in **IRange** interface to get or set cell type for a cell or range of cells. If the cell types are different in a range of cells, the cell type of the top-left cell of the range will be returned. The **CellType** property of **IWorksheet** interface can be used to get or set cell type for a worksheet. Further, the **EntireColumn** and **EntireRow** property of **IRange** interface can be used to get or set cell types for columns and rows respectively.

 **Note:** Cell types are not supported by Excel. So, these are lost after saving to Excel files. But the cell types work well with [SpreadJS](#), and is retained during JSON I/O with SpreadJS.

### Button Cell Type

Refer to the following code to create a Button cell type:

C#

```
public void ButtonCellTypes()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    //Creating Button cell type
    ButtonCellType button = new ButtonCellType();
    button.Text = "Click Me..!!";
    button.ButtonBackColor = "LightBlue";
    button.MarginLeft = 10;
    worksheet.Range["A1:B2"].CellType = button;

    // Saving workbook to Pdf
    workbook.Save(@"ButtonCellTypes.pdf", SaveFileFormat.Pdf);
}
```

## CheckBox Cell Type

Refer to the following code to create a CheckBox cell type:

C#

```
public void CheckBoxCellTypes()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Creating CheckBoxCellType
    CheckBoxCellType checkBox = new CheckBoxCellType();
    checkBox.Caption = "Caption";
    checkBox.TextTrue = "True";
    checkBox.TextFalse = "False";
    checkBox.IsThreeState = false;
    worksheet.Range["A1:C3"].CellType = checkBox;

    worksheet.Range["A1"].Value = true;
    worksheet.Range["B2"].Value = true;

    // Saving workbook to Pdf
    workbook.Save(@"CheckBoxCellTypes.pdf", SaveFileFormat.Pdf);
}
```

```
}
```

## ComboBox Cell Type

Refer to the following code to create a ComboBox cell type:

C#

```
public void ComboCellTypes()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Creating ComboBoxCellType
    ComboBoxCellType comboBox = new ComboBoxCellType();
    comboBox.EditorValueType = EditorValueType.Value;

    ComboBoxCellItem comboItem = new ComboBoxCellItem();
    comboItem.Value = "US";
    comboItem.Text = "United States";
    comboBox.Items.Add(comboItem);

    comboItem = new ComboBoxCellItem();
    comboItem.Value = "CN";
    comboItem.Text = "China";
    comboBox.Items.Add(comboItem);

    comboItem = new ComboBoxCellItem();
    comboItem.Value = "JP";
    comboItem.Text = "Japan";
    comboBox.Items.Add(comboItem);

    worksheet.Range["A1:B2"].CellType = comboBox;
    worksheet.Range["A1"].Value = "CN";

    // Saving workbook to Pdf
    workbook.Save(@"ComboCellTypes.pdf", SaveFileFormat.Pdf);
}
```

## Hyperlink Cell Type

Refer to the following code to create a Hyperlink cell type:

C#

```
public void HyperlinkCellTypes()
{
```

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Creating HyperLinkCellType
HyperLinkCellType hyperlinkCell = new HyperLinkCellType();
hyperlinkCell.Text = "GrapeCity Website";
hyperlinkCell.LinkColor = "Blue";
hyperlinkCell.LinkToolTip = "GrapeCity Website";
hyperlinkCell.VisitedLinkColor = "Green";
hyperlinkCell.Target = HyperLinkTargetType.Blank;

worksheet.Range["A1"].CellType = hyperlinkCell;
worksheet.Range["A1"].Value = "https://www.grapacity.com/";

// Saving workbook to Pdf
workbook.Save(@"HyperlinkCellTypes.pdf", SaveFileFormat.Pdf);
}
```

## Range Template Cell

GcExcel supports Range Template cell type which allows you to specify a cell range in the worksheet which acts as a range template. The range template is considered as a single cell and can be applied to a cell or cell range, as desired. The data into the range template can be loaded from a data source.

This feature is particularly useful when you want to display some specific ranges of data with identical structures (as displayed in the screenshots below) without having to configure the same style for multiple ranges again and again.

### Range Template

Asset Type	Amount	Rate
Savings		#DIV/0!
Shares		#DIV/0!
Stocks		#DIV/0!
House		#DIV/0!
Bonds		#DIV/0!
Car		#DIV/0!
Total	\$0	

The above Range Template when applied to a cell range A1:B2 and is loaded with data from data source looks like below:

Peyton			Icey		
Asset Type	Amount	Rate	Asset Type	Amount	Rate
Savings	\$25,000	6.88%	Savings	\$30,000	14.08%
Shares	\$55,000	15.13%	Shares	\$45,000	21.13%
Stocks	\$15,000	4.13%	Stocks	\$25,000	11.74%
House	\$250,000	68.78%	House	\$20,000	9.39%
Bonds	\$11,000	3.03%	Bonds	\$18,000	8.45%
Car	\$7,500	2.06%	Car	\$75,000	35.21%
Total	\$363,500		Total	\$213,000	
Walter			Chris		
Asset Type	Amount	Rate	Asset Type	Amount	Rate
Savings	\$20,000	9.30%	Savings	\$70,000	25.93%
Shares	\$4,000	1.86%	Shares	\$85,000	31.48%
Stocks	\$95,000	44.19%	Stocks	\$35,000	12.96%
House	\$30,000	13.95%	House	\$20,000	7.41%
Bonds	\$10,000	4.65%	Bonds	\$15,000	5.56%
Car	\$56,000	26.05%	Car	\$45,000	16.67%
Total	\$215,000		Total	\$270,000	

The following steps must be performed to create a Range Template cell type:

1. **Create a Range Template:** Design the layout of Range Template in a worksheet. The template can be bound to data by using **BindingPath** property.
2. **Configure Data:** Configure a Data source to bind the template.
3. **Create & Apply Range Template cell type:** Create a Range Template cell type by using **RangeTemplateCellType** method and apply it to the desired cell range.

Refer to the following code to create a Range Template cell type.

C#

```
//create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();
GrapeCity.Documents.Excel.Workbook.ValueJsonSerializer = new
CustomObjectJsonSerializer();

var sheet1 = workbook.ActiveSheet;
// Step 1. Create a worksheet for designing range template
var sheet2 = workbook.Worksheets.Add();

// Step 2. Configure Data
var record1 = new PersonalAssets
{
    Name = "Peyton",
```

```
Savings = 25000,
Shares = 55000,
Stocks = 15000,
House = 250000,
Bonds = 11000,
Car = 7500
};
var record2 = new PersonalAssets
{
    Name = "Icey",
    Savings = 30000,
    Shares = 45000,
    Stocks = 25000,
    House = 20000,
    Bonds = 18000,
    Car = 75000
};
var record3 = new PersonalAssets
{
    Name = "Walter",
    Savings = 20000,
    Shares = 4000,
    Stocks = 95000,
    House = 30000,
    Bonds = 10000,
    Car = 56000
};
var record4 = new PersonalAssets
{
    Name = "Chris",
    Savings = 70000,
    Shares = 85000,
    Stocks = 35000,
    House = 20000,
    Bonds = 15000,
    Car = 45000
};

// Set binding path for cell.
sheet2.Range["A1:C1"].Merge();
sheet2.Range["A1:C1"].HorizontalAlignment = HorizontalAlignment.Center;
sheet2.Range["A1:C1"].VerticalAlignment = VerticalAlignment.Center;

sheet2.Range["A1"].BindingPath = "Name";
sheet2.Range["A1"].Font.Name = "Arial";
sheet2.Range["A1"].Font.Size = 15;
sheet2.Range["1:1"].RowHeight = 30;
sheet2.Range["A2"].Value = "Asset Type";
```



```
sheet2.Range["B2"].Value = "Amount";
sheet2.Range["C2"].Value = "Rate";
sheet2.Range["A3"].Value = "Savings";
sheet2.Range["A3"].Interior.Color = Color.FromArgb(145, 159, 129);
sheet2.Range["B3"].BindingPath = "Savings";
sheet2.Range["C3"].Formula = "=B3/B9";
sheet2.Range["A4"].Value = "Shares";
sheet2.Range["A4"].Interior.Color = Color.FromArgb(215, 145, 62);
sheet2.Range["B4"].BindingPath = "Shares";
sheet2.Range["C4"].Formula = "=B4/B9";
sheet2.Range["A5"].Value = "Stocks";
sheet2.Range["A5"].Interior.Color = Color.FromArgb(206, 167, 34);
sheet2.Range["B5"].BindingPath = "Stocks";
sheet2.Range["C5"].Formula = "=B5/B9";
sheet2.Range["A6"].Value = "House";
sheet2.Range["A6"].Interior.Color = Color.FromArgb(181, 128, 145);
sheet2.Range["B6"].BindingPath = "House";
sheet2.Range["C6"].Formula = "=B6/B9";
sheet2.Range["A7"].Value = "Bonds";
sheet2.Range["A7"].Interior.Color = Color.FromArgb(137, 116, 169);
sheet2.Range["B7"].BindingPath = "Bonds";
sheet2.Range["C7"].Formula = "=B7/B9";
sheet2.Range["A8"].Value = "Car";
sheet2.Range["A8"].Interior.Color = Color.FromArgb(114, 139, 173);
sheet2.Range["B8"].BindingPath = "Car";
sheet2.Range["C8"].Formula = "=B8/B9";
sheet2.Range["A9"].Value = "Total";

sheet2.Range["B9:C9"].Merge();
sheet2.Range["B9:C9"].HorizontalAlignment = HorizontalAlignment.Center;
sheet2.Range["B9:C9"].NumberFormat = "$#,##0_);( $#,##0) ";
sheet2.Range["B9:C9"].Formula = "=SUM(B3:B8)";

sheet2.Range["B3:B8"].NumberFormat = "$#,##0_);( $#,##0) ";
sheet2.Range["C3:C8"].NumberFormat = "0.00%";
sheet2.Range["C3:C8"].FormatConditions.AddDatabar();

// Set data source
sheet1.Range["A:B"].ColumnWidthInPixel = 300;
sheet1.Range["1:2"].RowHeightInPixel = 200;
sheet1.Range["A1"].Value = record1;
sheet1.Range["B1"].Value = record2;
sheet1.Range["A2"].Value = record3;
sheet1.Range["B2"].Value = record4;

// Step 3. Create a range template celltype
var rangeTemplateCelltype = new RangeTemplateCellType(sheet2);
```

```
// Apply cell type to "A1:B2"
sheet1.Range["A1:B2"].CellType = rangeTemplateCelltype;

//save to a pdf file
workbook.Save("addrangetemplatecelltype.pdf");

}
class CustomObjectJsonSerializer : IJsonSerializer
{
    public object Deserialize(string json)
    {
        return Newtonsoft.Json.JsonConvert.DeserializeObject(json);
    }

    public string Serialize(object value)
    {
        return Newtonsoft.Json.JsonConvert.SerializeObject(value);
    }
}
class PersonalAssets
{
    public string Name;
    public int Savings;
    public int Shares;
    public int Stocks;
    public int House;
    public int Bonds;
    public int Car;
}
```

### Limitation

Excel doesn't support Range Template cell type. Hence, it would be lost after saving to xlsx file.

## Quote Prefix

GcExcel library provides the Quote Prefix feature just like Microsoft Excel. You can add a single quote or an apostrophe as a prefix to handle the cell value as text. The quote prefix remains hidden and only the cell value is visible. The single quote prefix can be seen in the formula bar when the user selects the cell.

Refer to the following example code to see how the quote prefix works in an Excel spreadsheet using GcExcel.

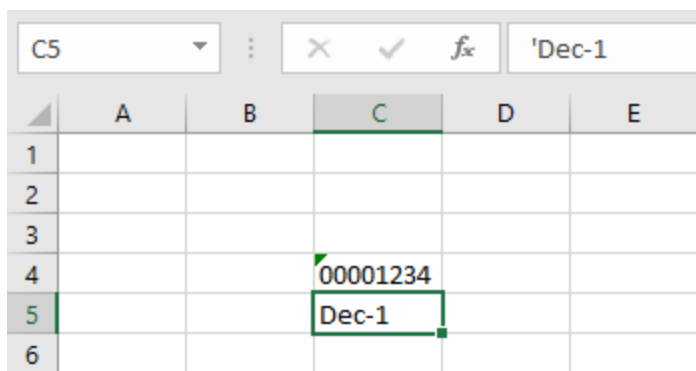
C#

```
workbook.Worksheets[0].Range["C4"].Value = "'00001234";
workbook.Worksheets[0].Range["C5"].Value = "'Dec-1";

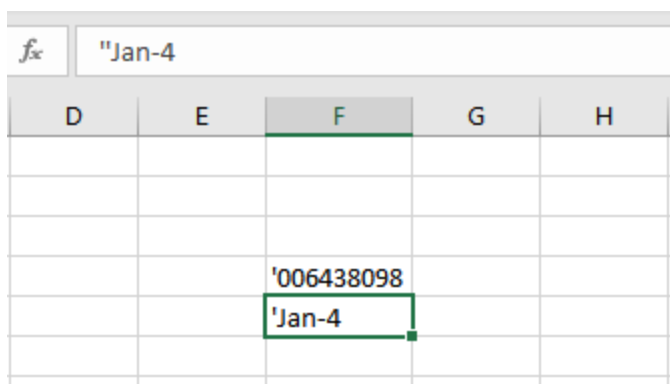
workbook.Worksheets[0].Range["F4"].Value = "'006438098";
```

```
workbook.Worksheets[0].Range["F5"].Value = "'Jan-4";
```

The below image shows the output of a prefixed quote. The quote is displayed in the formula bar whereas the cell hides it.



The below image shows the output of two prefixed quotes where both the quotes are displayed in the formula bar whereas the cell retains only one.



## Tags

With GcExcel, you can configure tags for worksheets which allow you to store private data in a cell, row, column, range or spreadsheet. The tags can store any type of data and are not visible to the end user. Tags are retained while performing the import or export operations from or to JSON.

Tags for worksheets can be configured using the **Tag** property of **IWorksheet** interface. Whereas for a cell or a range of cells, the tags can be configured using the **Tag** property of **IRange** interface. If the tag values are different in a range of cells, the tag value of the top-left cell of the range will be returned. The **EntireColumn** and **EntireRow** property of **IRange** interface, along with the **Tag** property can be used to get or set tags for columns and rows respectively.

You can also configure custom tags by instantiating a class. A json serializer or deserializer should be provided to support json I/O by implementing **IJsonSerializer**.

### Using Code

Refer to the following code to use tags:

```
C#  
  
public void Tags()  
{
```

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

//Add Tag for worksheet
worksheet.Tag = "This is a Tag for sheet.";
// Add Tag for Cell C1
worksheet.Range["C1"].Tag = "This is a Tag for Cell C1";
//Add Tag for Row 4
worksheet.Range["A4"].EntireRow.Tag = "This is a Tag for Row 4";
//Add Tag for Column F
worksheet.Range["F5"].EntireColumn.Tag = "This is a Tag for Column F";
//Add tag for Range A1:B2
worksheet.Range["A1:B2"].Tag = "This is a Tag for A1:B2";

// Exporting workbook to JSON stream
var jsonstr = workbook.ToJson();

// Initialize another workbook
Workbook workbook2 = new Workbook();

// Importing JSON stream in workbook
workbook2.FromJson(jsonstr);

// Get Tag of Range A1:B2
object tag = workbook2.Worksheets[0].Range["A1:B2"].Tag;

// Tags are preserved while exporting and importing json stream
Console.WriteLine(" Tag for CellRange[A1:B2] is : " + tag);
}
```

Refer to the following code to use custom tags:

C#

```
class CustomTags
{
    public void CustomTag()
    {
        // Initialize workbook
        Workbook workbook = new Workbook();
        // Fetch default worksheet
        IWorksheet worksheet = workbook.Worksheets[0];

        Workbook.TagJsonSerializer = new NetJsonSerializer();
    }
}
```

```
// Set Tag of "A1" as custom type "Student"
worksheet.Range["A1"].Tag = new Student("Robin", 7);

// Exporting workbook to JSON stream
string json = workbook.ToJson();

// Initialize another workbook
Workbook workbook2 = new Workbook();

// Importing JSON stream in workbook
workbook2.FromJson(json);

// Get Tag as JObject
object tag = workbook2.Worksheets[0].Range["A1"].Tag;

// Convert JObject to "Student"
Student student = (tag as Newtonsoft.Json.Linq.JObject).ToObject<Student>();

// Tags are preserved while exporting and importing json stream
Console.WriteLine(" Tag for CellRange[A1] is of class: " + student);
Console.WriteLine(" Tag for CellRange[A1] is: " + tag);

}
}

internal class Student
{
    public string name;
    public int age;

    public Student(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
}

internal class NetJsonSerializer : IJsonSerializer
{
    public object Deserialize(string json)
    {
        return Newtonsoft.Json.JsonConvert.DeserializeObject(json) as
Newtonsoft.Json.Linq.JObject;
    }

    public string Serialize(object value)
    {
        return Newtonsoft.Json.JsonConvert.SerializeObject(value);
    }
}
```

```
}  
}
```

## Rich Text

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 allows you to apply multiple styles to the text by highlighting important characters or alphabets using different colors, font family, font effects (bold, underline, double underline, strikethrough, subscript, superscript) and font size etc.

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** method of the **IRichText** 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()** method of the **IRange** 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()** method of the **IRange** 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** interface provides the properties and methods for adding and customizing the rich text entered in the cells of the worksheet. The **InsertAfter()** and **InsertBefore()** 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** 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;
```

## 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 and Save Workbook](#)
- [Protect Workbook](#)
- [Cut or Copy Across Sheets](#)
- [Enable or Disable Calculation Engine](#)
- [Workbook Views](#)

## Create Workbook

In GcExcel, you can create a new instance of a workbook by using the constructor of **Workbook** 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 and Save Workbook

Once you create a workbook, you can open the workbook to make modifications and save the changes back to the workbook.

This topic includes the following tasks:

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

### Open a workbook

You can open an existing workbook by calling the **Open** 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** method of the Workbook class. If your workbook is password protected, you would need to provide a password everytime you open it.

Apart from .xlsx files, you can also open the below file formats by using the overloads of **Open** method in Workbook class:

- .xlsm
- .csv
- .json
- .ssjson

However, an exception is thrown when unsupported file formats are opened. While opening a JSON file, the **DeserializationOptions** are supported as well.

Refer to the following example code to open a JSON file with and without options.

C#

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

// Import JSON without options
workbook.Open("file.json");

// Import JSON with options
var options = new DeserializationOptions { IgnoreStyle = true };
workbook.Open("file.json", options);
```

## Save a workbook

You can save the changes made in the existing workbook by calling the **Save** 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 Workbook

GcExcel allows you to protect a workbook in case it contains any critical and confidential information that cannot be shared with others. Additionally, you can also protect it from modification so that other users can't perform certain operations on the workbook.

To protect or unprotect a workbook, you can perform the following tasks:

- **Protect Workbook Using Password**
- **Protect Workbook from Modification**
- **Unprotect Workbook from Modification**

## Protect Workbook using Password

GcExcel enables 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 cannot be shared with everyone. You can secure a workbook using the **Password** property of **XlsxSaveOptions** class.

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

C#

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

## Protect Workbook from Modification

GcExcel allows you to protect a workbook from modification. Optionally, a password can be set to achieve the same.

The **Workbook** class provides two overloaded **Protect** methods, one of which takes password as a parameter. Both the methods have two optional parameters, **structure** and **windows**, which provide different types of modification protection when set.

- If **structure** is true, worksheets cannot be added, moved, deleted, hidden or renamed, and hidden worksheets cannot be viewed. Its default value is true.
- If **windows** is true, the workbook window cannot be moved, resized, closed or hidden or unhidden. This option is supported only in Excel 2007, Excel 2010, Excel for Mac 2011 and Excel 2016 for Mac. Its default value is false.

Refer to the following example code to protect the workbook from modification using password.

C#

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

//Protects the workbook with password so that other users cannot view hidden worksheets,
add, move, delete, hide, or rename worksheets.
workbook.Protect("Ygs_87@ytr");
// Save workbook to xlsx
workbook.Save(@"ProtectWorkbookWithPassword.xlsx", SaveFileFormat.Xlsx);
```

Refer to the following example code to protect the workbook from modification without using password.

C#

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

```
//Protects the workbook so that other users cannot view hidden worksheets, add, move,
delete, hide, or rename worksheets.
workbook.Protect();
// Saving workbook to xlsx
workbook.Save(@"ProtectWorkbook.xlsx", SaveFileFormat.Xlsx);
```

## Unprotect Workbook from Modification

A protected workbook can be unprotected to make modifications using the **Unprotect** method of the **Workbook** class, which removes the protection from a workbook.

To unprotect a password protected workbook, the correct password needs to be passed as a parameter to the **Unprotect** method. In case, the password is omitted or an incorrect password is passed, an exception message "Invalid Password" is thrown.

Refer to the following example code to unprotect a password protected workbook.

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

workbook.Protect("Ygs_87@ytr");
//Removes the above protection from the workbook
workbook.Unprotect("Ygs_87@ytr");
// Save workbook to xlsx
workbook.Save(@"UnprotectWorkbookWithPassword.xlsx", SaveFileFormat.Xlsx);
```

If a workbook is not protected with a password, the password argument is ignored by the **Unprotect** method.

Refer to the following example code to unprotect the protected workbook.

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

workbook.Protect();
//Removes the above protection from the workbook.
workbook.Unprotect();
// Saving workbook to xlsx
workbook.Save(@"UnprotectWorkbook.xlsx", SaveFileFormat.Xlsx);
```

## 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;
```

## Workbook Views

GcExcel allows users to personalize the display of the workbook. You can use the **BookView** property of the **IWorkbook** interface to set the view of the workbook as per your preferences.

The following properties of the **IWorkbookView** interface allows users to further customize various display settings in the workbook.

1. **DisplayHorizontalScrollBar** - This property gets and sets the display of the horizontal scrollbar.
2. **DisplayVerticalScrollBar** - This property gets and sets the display of the vertical scrollbar.
3. **DisplayWorkbookTabs** - This property gets and sets if the workbook tabs are displayed.
4. **TabRatio** - 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;
```

## Comments

GcExcel .NET enables you to annotate a worksheet by writing comments on cells in order to specify additional information about the data it contains.

For instance, let us assume you want 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

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**. You can insert comment to a cell or a range of cells using the **AddComment method** of the **IRange interface**. You can also set the text of the comment using the **Text property** of the **IComment interface**.

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

C#

```
// Create a comment for the range ["C3"]
IComment commentC3 = worksheet.Range["C3"].AddComment("Range C3's comment.");

//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** of the **IComment interface**.

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** of the **IComment interface**.

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

```
C#  
  
//Show Comment  
worksheet.Range["C3"].Comment.Visible = true;  
  
//Hide Comment  
worksheet.Range["C3"].Comment.Visible = false;
```

## Author Comments

You can represent the author of the comment by using the **Author property** of the **IComment interface**. 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** that control the

text style.

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

C#

```
IComment commentC3 = worksheet.Range["C3"].AddComment("Cell ");

// Set paragraph's font style
commentC3.Shape.TextFrame.TextRange.Paragraphs[0].Font.Bold = true;

// Add comment in paragraph
commentC3.Shape.TextFrame.TextRange.Paragraphs[0].Runs.Add(" C3");
commentC3.Shape.TextFrame.TextRange.Paragraphs[0].Runs.Add(" Comment");
commentC3.Shape.TextFrame.TextRange.Paragraphs[0].Runs.Add(" Added");

// Set the style of the second comment
commentC3.Shape.TextFrame.TextRange.Paragraphs[0].Runs[2].Font.Italic = true;
commentC3.Shape.TextFrame.TextRange.Paragraphs[0].Runs[2].Font.Bold = true;

// Show comment
commentC3.Visible = true;
```

## Delete Comment

You can delete the comment added to a cell or to a cell range using the **Delete method** of the **IComment interface** and the **IRange interface** respectively.

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

C#

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

## 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. 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** and the **IRange interface** respectively.

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** of the **IHyperLinks interface**.

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**.

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();
```

## Sort

GcExcel provides the **Sort** method to perform data sorting based on a range of cells, range by value, color or icon in a worksheet. The **Apply** 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();
```



## Filter

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 of the **IWorksheet** interface.

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 criterial1 = new DateTime(2008, 1, 1).ToString();
var criteria2 = new DateTime(2008, 8, 1).ToString();
worksheet.Range["D20:F29"].AutoFilter(2, ">=" + criterial1, 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 yersterday.
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);
```

## Group

GcExcel .NET provides you with the ability to summarize large amounts of information in groups so that complex spreadsheets are easier to navigate. The data in rows or columns can be grouped 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** of the **IRange Interface** 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](#)
- [Summary Row](#)
- [Outline Subtotals](#)
- [Outline Column](#)



**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** of the **IRange interface** 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 of the `IRange` interface. You can also choose to display specified levels of row or column groups using the **ShowLevels** method of the **IOutline** interface.

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 or **SummaryRow** property 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** and **Ungroup method** of the **IRange interface** 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** 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** 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;
```

## 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** of the **IOutline interface**.

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();
```

## Outline Subtotals

You can derive meaningful and summarized insights from outline data by applying the subtotals to the grouped values.

In GcExcel, you can apply outline subtotals to organize the sorted data into groups and display subtotals at the end of each group.

### Create Outline Subtotals

The outline subtotals are created using the **Subtotal** method of **IRange** interface. The method provides different parameters to group by fields, assign subtotal function, replace existing subtotals, add page breaks and place summary data.

The below sample data is used to create outline subtotals:

```
C#  
  
public void PopulateData(Workbook workbook)  
{  
    IWorksheet worksheet = workbook.Worksheets[0];  
  
    // Defining data in the range  
    worksheet.Range["A1:C20"].Value = new object[,]  
    {  
        {"Item", "Units", "Unit Price"},  
        {"Pen Set", 62, 4.99},  
        {"Binder", 29, 1.99},  
        {"Pen Set", 55, 12.49},  
        {"Binder", 81, 19.99},  
        {"Pen Set", 42, 23.95},
```

```
{ "Pencil", 35, 4.99 },
{ "Desk", 3, 275 },
{ "Desk", 2, 125 },
{ "Pencil", 7, 1.29 },
{ "Pen Set", 16, 15.99 },
{ "Pen", 76, 1.99 },
{ "Binder", 28, 8.99 },
{ "Binder", 57, 19.99 },
{ "Pen", 64, 8.99 },
{ "Pencil", 14, 1.29 },
{ "Pen", 15, 19.99 },
{ "Binder", 11, 4.99 },
{ "Pen Set", 96, 4.99 },
{ "Binder", 94, 19.99 }
};
```

Refer to the below example code to create outline subtotals.

C#

```
IWorksheet _worksheet = workbook.Worksheets[0];

//Sort by value, use Sort() method.
_worksheet.Range["A2:C20"].Sort(_worksheet.Range["A2:A20"], orientation:
SortOrientation.Columns);

//Create groups and sub-total the grouped values using Subtotal() method
_worksheet.Range["$A$1:$D$20"].Subtotal(1, ConsolidationFunction.Sum, new[] { 2, 3 });

//Save workbook
workbook.Save("OutlineSubtotal.xlsx");
```

1	2	3		A	B	C	
1				Item	Units	Unit Price	
2				Binder	29	1.99	
3				Binder	81	19.99	
4				Binder	28	8.99	
5				Binder	57	19.99	
6				Binder	11	4.99	
7				Binder	94	19.99	
8				<b>Binder Total</b>	300	75.94	
9				Desk	3	275	
10				Desk	2	125	
11				<b>Desk Total</b>	5	400	
12				Pen	76	1.99	
13				Pen	64	8.99	
14				Pen	15	19.99	
15				<b>Pen Total</b>	155	30.97	

### Remove Outline Subtotals

The outline subtotals can be removed using the **RemoveSubtotal** method of the **IRange** interface.

Refer to the below example code to remove outline subtotals.

C#

```
Workbook workbook = new Workbook();
workbook.Open("OutlineSubtotal.xlsx");

IWorksheet _worksheet = workbook.Worksheets[0];

//Remove Subtotals, pass the cell range inclusive of the subtotal/total rows
_worksheet.Range["$A$1:$C$26"].RemoveSubtotal();

//Save workbook
workbook.Save("OutlineNoSubtotal.xlsx");
```

## Outline Column

Outline columns can be used to organize large amounts of data into meaningful groups.

GcExcel allows you to add outline columns to view hierarchical data in a tree view and show or hide it from view. The **OutlineColumn** property of **IWorksheet** interface can be used to add the outline column. The row outlines are automatically created by adding the outline column. When a worksheet is saved to Excel, the outline column is not displayed but the row outlines are retained.

The indent level of a cell can be set by using the **IndentLevel** property of the **IRange** interface. The maximum indentation level can be set by using **MaxLevel** property of **IOutlineColumn** interface whose default value is 10.



You can also use the **Refresh** method of **IOutlineColumn** interface to rebuild the tree data structure based on the current outline column options and indents.

The outline column can also be exported to PDF and imported or exported to JSON to interact with SpreadJS.

### Using Code

Refer to the below example code to create outline column.

C#

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

//Set data.
object[,] data = new object[[],]{
{ "Preface", "1", 1 },
{ "Java SE5 and SE6", "1.1", 2 },
{ "Java SE6", "1.1.1", 2 },
{ "The 4th edition", "1.2", 2 },
{ "Changes", "1.2.1", 3 },
{ "Note on the cover design", "1.3", 4 },
{ "Acknowledgements", "1.4", 4 },
{ "Introduction", "2", 9 },
{ "Prerequisites", "2.1", 9 },
{ "Learning Java", "2.2", 10 },
{ "Goals", "2.3", 10 },
{ "Teaching from this book", "2.4", 11 },
{ "JDK HTML documentation", "2.5", 11 },
{ "Exercises", "2.6", 12 },
{ "Foundations for Java", "2.7", 12 },
{ "Source code", "2.8", 12 },
{ "Coding standards", "2.8.1", 14 },
{ "Errors", "2.9", 14 },
{ "Introduction to Objects", "3", 15 },
{ "The progress of abstraction", "3.1", 15 },
{ "An object has an interface", "3.2", 17 },
{ "An object provides services", "3.3", 18 },
{ "The hidden implementation", "3.4", 19 },
{ "Reusing the implementation", "3.5", 20 },
{ "Inheritance", "3.6", 21 },
{ "Is-a vs. is-like-a relationships", "3.6.1", 24 },
{ "Interchangeable objects with polymorphism", "3.7", 25 },
{ "The singly rooted hierarchy", "3.8", 28 },
{ "Containers", "3.9", 28 },
{ "Parameterized types (Generics)", "3.10", 29 },
{ "Object creation & lifetime", "3.11", 30 },
{ "Exception handling: dealing with errors", "3.12", 31 },
{ "Concurrent programming", "3.13", 32 },
```

```
{ "Java and the Internet",          "3.14",    33 },
{ "What is the Web?",              "3.14.1",  33 },
{ "Client-side programming",       "3.14.2",  34 },
{ "Server-side programming",       "3.14.3",  38 },
{ "Summary",                       "3.15",    38 }
};

worksheet.Range["A1:C38"].Value = data;

//Set ColumnWidth.
worksheet.Range["A:A"].ColumnWidthInPixel = 310;
worksheet.Range["B:C"].ColumnWidthInPixel = 150;

//Set IndentLevel.
for (int i = 0; i < data.GetLength(0); i++)
{
    worksheet.Range[i, 0].IndentLevel = (int)data[i, 3];
}

//Show the summary row above the detail rows.
worksheet.Outline.SummaryRow = SummaryRow.Above;


//Don't show the row outline when interacting with SJS, the exported excel file
still show the row outline.
worksheet.ShowRowOutline = false;

//Set outline column.
worksheet.OutlineColumn.ColumnIndex = 0;
worksheet.OutlineColumn.ShowCheckBox = true;
worksheet.OutlineColumn.ShowImage = true;
worksheet.OutlineColumn.MaxLevel = 2;
worksheet.OutlineColumn.Images.Add(new ImageSource(File.Open("archiverFolder.png",
FileMode.Open), ImageType.PNG));
worksheet.OutlineColumn.Images.Add(new ImageSource(File.Open("newFolder.png",
FileMode.Open), ImageType.PNG));
worksheet.OutlineColumn.Images.Add(new ImageSource(File.Open("docFile.png",
FileMode.Open), ImageType.PNG));
worksheet.OutlineColumn.CollapseIndicator = new
ImageSource(File.Open("decreaseIndicator.png", FileMode.Open), ImageType.PNG);
worksheet.OutlineColumn.ExpandIndicator = new
ImageSource(File.Open("increaseIndicator.png", FileMode.Open), ImageType.PNG);

worksheet.OutlineColumn.SetCheckStatus(0, true);
worksheet.OutlineColumn.SetCollapsed(1, true);

//Print the headings & gridlines.
worksheet.PageSetup.PrintHeadings = true;
worksheet.PageSetup.PrintGridlines = true;
```

```
//Save to json/excel/pdf.
workbook.ToJson(new FileStream("outlineColumn1.json", FileMode.Create));
workbook.Save("outlineColumn1.xlsx");
workbook.Save("outlineColumn1.pdf");
```

 **Note:** The images, checkbox, expand or collapse indicator images are not visible in Excel as it does not supports them but they can be viewed in PDF and SpreadJS.

The below image shows the Excel output of above code snippet:

	A	B	C
1	Preface	1	1
2	Java SE5 and SE6	1.1	2
4	The 4th edition	1.2	2
5	Changes	1.2.1	3
6	Note on the cover design	1.3	4
7	Acknowledgements	1.4	4
8	Introduction	2	9
9	Prerequisites	2.1	9
10	Learning Java	2.2	10
11	Goals	2.3	10
12	Teaching from this book	2.4	11
13	JDK HTML documentation	2.5	11
14	Exercises	2.6	12
15	Foundations for Java	2.7	12
16	Source code	2.8	12
18	Errors	2.9	14
19	Introduction to Objects	3	15
20	The progress of abstraction	3.1	15
21	An object has an interface	3.2	17
22	An object provides services	3.3	18
23	The hidden implementation	3.4	19
24	Reusing the implementation	3.5	20
25	Inheritance	3.6	21
26	Is-a vs. is-like-a relationships	3.6.1	24
27	Interchangeable objects with polymorphism	3.7	25
28	The singly rooted hierarchy	3.8	28
29	Containers	3.9	28
30	Parameterized types (Generics)	3.10	29
31	Object creation & lifetime	3.11	30
32	Exception handling: dealing with errors	3.12	31
33	Concurrent programming	3.13	32
34	Java and the Internet	3.14	33
38	Summary	3.15	38

The below image shows the PDF output of above code snippet:

	A	B	C
1	Preface	1	1
2	Java SE5 and SE6	1.1	2
4	The 4th edition	1.2	2
5	Changes	1.2.1	3
6	Note on the cover design	1.3	4
7	Acknowledgements	1.4	4
8	Introduction	2	9
9	Prerequisites	2.1	9
10	Learning Java	2.2	10
11	Goals	2.3	10
12	Teaching from this book	2.4	11
13	JDK HTML documentation	2.5	11
14	Exercises	2.6	12
15	Foundations for Java	2.7	12
16	Source code	2.8	12
17	Coding standards	2.8.1	14
18	Errors	2.9	14
19	Introduction to Objects	3	15
20	The progress of abstraction	3.1	15
21	An object has an interface	3.2	17
22	An object provides services	3.3	18
23	The hidden implementation	3.4	19
24	Reusing the implementation	3.5	20
25	Inheritance	3.6	21
26	Is-a vs. is-like-a relationships	3.6.1	24
27	Interchangeable objects with polymorphism	3.7	25
28	The singly rooted hierarchy	3.8	28
29	Containers	3.9	28
30	Parameterized types (Generics)	3.10	29
31	Object creation & lifetime	3.11	30
32	Exception handling: dealing with errors	3.12	31
33	Concurrent programming	3.13	32
34	Java and the Internet	3.14	33
35	What is the Web?	3.14.1	33
36	Client-side programming	3.14.2	34
37	Server-side programming	3.14.3	38
38	Summary	3.15	38

## Conditional Formatting

In order 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.

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 of the **IRange** interface.

Shared below is a list of conditional formatting rules that can be applied in a worksheet.

- [Cell Value Rule](#)
- [Date Occurring Rule](#)
- [Average Rule](#)
- [Color Scale Rule](#)
- [Data Bar Rule](#)
- [Top Bottom Rule](#)
- [Unique Rule](#)
- [Icon Sets Rule](#)
- [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 of **IFormatCondition** interface

## 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 and **Formula2** property of the **IFormatCondition** interface. You can also use the **Operator** property 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";
```

## 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 of the **IFormatCondition** interface.

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);
```

## Average Rule

The average rule in conditional formatting can be added and deleted using the properties and methods of the **IAboveAverage** interface.

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";
```

## 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.

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);
```

## 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.

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;
```

## 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**.

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);
```

## 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**.

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";
```

## 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.

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;
```

## 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.

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);
```

## Data Validations

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.

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** of the **IValidation interface** 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** of the **IRange interface**. If you want to

know whether a cell range already contains the validation rule, you can use the **HasValidation property** 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** 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** of the **IValidation interface**.

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** (**Type property**, **Formula1 property**, **Formula2 property**, and many more).
- Use **Delete method** of the **IValidation interface** to first delete validation rule and then use the **Add method** 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();
```

## Data Binding

GcExcel supports data binding which allows you to generate data bound reports and view them in Excel. Data binding can be achieved by binding a data source with a sheet, cell or table column. You can also perform JSON I/O of the binding path to interact with SpreadJS.

### Sheet Binding

A data source can be bound to a sheet by using the **DataSource** property of **IWorksheet** interface. The data sources supported for binding a sheet are DataTable or an IEnumerable collection. Each worksheet can have only one data source.

To bind the data source fields to sheet columns automatically, you can set the **AutoGenerateColumns** property of IWorksheet interface to true. The default value is also true.

To bind the data source fields to sheet columns manually, you can set the **AutoGenerateColumns** property of IWorksheet interface to false and use the **BindingPath** property of **IRange** interface to set the binding path of the data source field to the sheet columns.

For eg. If you want to display the 'TeamName' field in column D, the binding path for the 'TeamName' field will be column D.

Refer to the below example code to bind a DataTable to the sheet columns manually.

C#

```
//create a new workbook
Workbook workbook = new Workbook();

DataTable teamInfo = new DataTable();
teamInfo.Columns.Add(new DataColumn("ID", typeof(Int32)));
teamInfo.Columns.Add(new DataColumn("Name", typeof(string)));
teamInfo.Columns.Add(new DataColumn("Score", typeof(Int32)));
teamInfo.Columns.Add(new DataColumn("Team", typeof(string)));
```

```
teamInfo.Rows.Add(10, "Bob", 12, "Xi'An");
teamInfo.Rows.Add(11, "Tommy", 6, "Xi'An");
teamInfo.Rows.Add(12, "Jaguar", 15, "Xi'An");
teamInfo.Rows.Add(12, "Lusia", 9, "Xi'An");

IWorksheet worksheet = workbook.Worksheets[0];

// Set AutoGenerateColumns as false
worksheet.AutoGenerateColumns = false;

//Bind columns manually.
worksheet.Range["A:A"].EntireColumn.BindingPath = "ID";
worksheet.Range["B:B"].EntireColumn.BindingPath = "Name";
worksheet.Range["C:C"].EntireColumn.BindingPath = "Score";
worksheet.Range["D:D"].EntireColumn.BindingPath = "Team";

// Set data source
worksheet.DataSource = teamInfo;

//save to an excel file
workbook.Save("SheetBindDatatable.xlsx");
```

## Cell Binding

A data source can be bound to a cell by using the **DataSource** property of **IWorksheet** interface. The data source supported for binding a cell is custom object.

The **BindingPath** property of **IRange** interface can be used to set the binding path of the data source field to a cell. For eg. If 'Area' field is to be displayed in cell A1, the binding path for the 'Area' field will be cell A1.

Refer to the below example code to bind custom object to cells.

C#

```
public void CellBinding()
{
    // create a new workbook
    Workbook workbook = new Workbook();

    var record = new SalesRecord
    {
        Area = "NorthChina",
        Salesman = "Hellen",
        Product = "Apple",
        ProductType = "Fruit",
        Sales = 120
    };
};
```

```
IWorksheet worksheet = workbook.Worksheets[0];

// Set binding path for cell.
worksheet.Range["A1"].BindingPath = "Area";
worksheet.Range["B2"].BindingPath = "Salesman";
worksheet.Range["C2"].BindingPath = "Product";
worksheet.Range["D3"].BindingPath = "ProductType";

// Set data source.
worksheet.DataSource = record;

//save to an excel file
workbook.Save("cellbinding.xlsx");
}
internal class SalesRecord
{
    public string Area;
    public string Salesman;
    public string Product;
    public string ProductType;
    public int Sales;
}
```

## Table Binding

A data source can be bound to a table by using the **DataSource** property of **IWorksheet** interface. The data sources supported for binding a table are **DataSet** or custom object which contains an **IEnumerable** field or property. The **BindingPath** property of **ITable** interface can be used to set the binding path of data source to a table.

To bind the data source fields to table columns automatically, you can set the **AutoGenerateColumns** property of **IWorksheet** interface to true. The default value is also true.

To bind the data source fields to table columns manually, you can set the **AutoGenerateColumns** property of **IWorksheet** interface to false and use the **DataField** property of **ITableColumn** interface to set the binding path of the data source field to the table columns.

For eg. 'T1' **DataTable** is bound to the first table and 'ID' field is bound to the first column of table.

Refer to the below example code to bind a dataset to table columns manually.

```
C#

//create a new workbook
Workbook workbook = new Workbook();

// DataSet
var team1 = new DataTable("T1");
team1.Columns.Add(new DataColumn("ID", typeof(Int32)));
team1.Columns.Add(new DataColumn("Name", typeof(string)));
team1.Columns.Add(new DataColumn("Score", typeof(Int32)));
```



```
team1.Columns.Add(new DataColumn("Team", typeof(string)));

team1.Rows.Add(10, "Bob", 12, "Xi'An");
team1.Rows.Add(11, "Tommy", 6, "Xi'An");
team1.Rows.Add(12, "Jaguar", 15, "Xi'An");
team1.Rows.Add(12, "Lusia", 9, "Xi'An");

var team2 = new System.Data.DataTable("T2");
team2.Columns.Add(new DataColumn("ID", typeof(Int32)));
team2.Columns.Add(new DataColumn("Name", typeof(string)));
team2.Columns.Add(new DataColumn("Score", typeof(Int32)));
team2.Columns.Add(new DataColumn("Team", typeof(string)));

team2.Rows.Add(2, "Phillip", 9, "BeiJing");
team2.Rows.Add(3, "Hunter", 10, "BeiJing");
team2.Rows.Add(4, "Hellen", 8, "BeiJing");
team2.Rows.Add(5, "Jim", 9, "BeiJing");

var datasource = new System.Data.DataSet();
datasource.Tables.Add(team1);
datasource.Tables.Add(team2);

IWorksheet worksheet = workbook.Worksheets[0];

// Add tables
ITable table = worksheet.Tables.Add(worksheet.Range["B2:E6"], true);
ITable table2 = worksheet.Tables.Add(worksheet.Range["G2:J6"], true);

// Set not to auto generate table columns
table.AutoGenerateColumns = false;
table2.AutoGenerateColumns = false;

// Set table binding path
table.BindingPath = "T1";
table2.BindingPath = "T2";

// Set table column data field
table.Columns[0].DataField = "ID";
table.Columns[1].DataField = "Name";
table.Columns[2].DataField = "Score";
table.Columns[3].DataField = "Team";

table2.Columns[0].DataField = "ID";
table2.Columns[1].DataField = "Name";
table2.Columns[2].DataField = "Score";
table2.Columns[3].DataField = "Team";

// Set DataSet as datasource
```

```
worksheet.DataSource = datasource;  
  
//save to an excel file  
workbook.Save("TableBindDataset.xlsx");
```

## Limitation

GcExcel supports one-time data binding which means that the data will be populated only the first time when data source is set, afterwards the data will not change even if the data in datasource changes.

## Digital Signatures

Digital signatures are the proof of a document's authenticity. A digitally signed document assures that it has been created by the signer and has not been changed in any way.

GcExcel allows users to add digital signatures to Excel spreadsheets to make them authentic and easier to validate.

## Signature Lines

Signature lines act as a signature placeholder for digital signatures. They can be added to worksheet as signature line shapes which can be signed further.

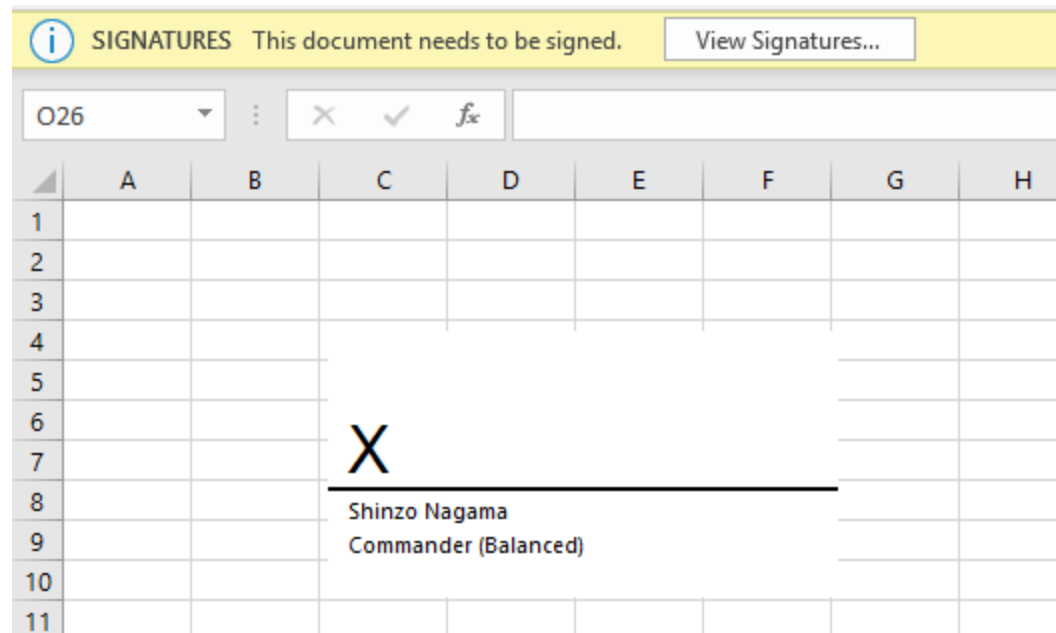
### Add Signature Line

The **AddSignatureLine** method of **ISignatureSet** interface adds signature lines in a worksheet. You can also add information about the intended signer and instructions for the signer by using various properties of **ISignatureSetup** interface. When the workbook is opened again or sent to the intended signer as an Excel file, the signature line can be seen along with a notification that their signature is requested.

Refer to the following example code to add signature line in a worksheet.

```
C#  
  
var workbook = new Workbook();  
IWorksheet activeSheet = workbook.ActiveSheet;  
  
//add signature line  
ISignatureSetup setup = workbook.Signatures.AddSignatureLine(activeSheet, 100.0,  
50.0).Setup;  
setup.ShowSignDate = false;  
setup.AllowComments = false;  
setup.SigningInstructions = "Please check the content before signing.";  
setup.SuggestedSigner = "Shinzo Nagama";  
setup.SuggestedSignerEmail = "shinzo.nagama@ea.com";  
setup.SuggestedSignerLine2 = "Commander (Balanced)";  
  
//save to Excel file  
workbook.Save("addsignaturelines.xlsx");
```

The below image shows the signature lines in Excel:



### Copy Signature Lines

You can copy a signature line to another range of worksheet or to another worksheet by using any of the below:

- Duplicate signature line - By using **Duplicate** method of **IShape** interface
- Copy signature line's cell range - By using **Copy** method of **IRange** interface
- Copy worksheet containing signature line - By using **Copy** method of **IWorksheet** interface

Refer to the following example code to copy a signature line to another range and another worksheet.

C#

```
//copy signature line to another range
IRange srcRange = activeSheet.Range["A1:I15"];
IRange destRange = activeSheet.Range["A16:I30"];
srcRange.Copy(destRange);

//duplicate signature line
signature.SignatureLineShape.Duplicate();

//copy signature line to another worksheet
activeSheet.Copy();
```

### Delete Signature Lines

You can delete a signature line by using any of the below:

- Delete signature line - By using **Delete** method of **ISignature** interface

- Delete shape associated with signature line - By using **Delete** method of **IShape** interface

Refer to the following example code to delete signature line in a worksheet.

C#

```
//create a new signature line and delete with Signature.Delete
ISignature signatureForTest = newSignatureLine();
signatureForTest.Delete();

//create a new signature line and delete with Shape.Delete
signatureForTest = newSignatureLine();
IShape signatureLineShape = signatureForTest.SignatureLineShape;
signatureLineShape.Delete();
```

## Move Signature Lines

Refer to the following example code to move signature lines to another range or a worksheet.

C#

```
//move signature line
signature.SignatureLineShape.Top += 100;
signature.SignatureLineShape.Left += 50;
```

## List Signature Lines

Refer to the following example code to list signature lines in a worksheet.

C#

```
//add first signature line
ISignature signatureShinzo = signatures.AddSignatureLine(
    activeSheet, 100.0, 50.0);
ISignatureSetup setup1 = signatureShinzo.Setup;
setup1.ShowSignDate = false;
setup1.AllowComments = false;
setup1.SigningInstructions = "Please check the content before signing.";
setup1.SuggestedSigner = "Shinzo Nagama";
setup1.SuggestedSignerEmail = "shinzo.nagama@ea.com";
setup1.SuggestedSignerLine2 = "Commander (Balanced)";

//add second signature line
ISignature signatureKenji = signatures.AddSignatureLine(
    activeSheet, 100.0, 350.0);
ISignatureSetup setup2 = signatureKenji.Setup;
setup2.ShowSignDate = true;
setup2.AllowComments = true;
setup2.SigningInstructions = "Please check the content before signing!";
setup2.SuggestedSigner = "Kenji Tenzai";
```

```

setup2.SuggestedSignerEmail = "kenji.tenzai@ea.com";
setup2.SuggestedSignerLine2 = "Commander (Mecha)";

//list signatures with indexes
for (var i = 0; i < signatures.Count; i++)
{
    var signature = signatures[i];
    //change SuggestedSigner
    if (i == 0)
        signature.Setup.SuggestedSigner = "Shinzo Nagama 123";
    //change SuggestedSignerLine2
    if (i == 1)
        signature.Setup.SuggestedSignerLine2 = "Commander (Mecha 1234)";
}

```

The **SignatureLineShape** property in **ISignature** interface can be used while using signature line as a shape. Its members and their behavior is elaborated in the below table:

SignatureLineShape members	Get or Call Behavior	Set Behavior
Adjustments	Supported	#N/A
Adjustments.Count	Supported	#N/A
Adjustments.Item	Not Supported	Not Supported
Adjustments.GetEnumerator	Not Supported	#N/A
AutoShapeType	Supported	Not Supported
BottomRightCell	Supported	#N/A
Chart	Not Supported	#N/A
Connector	Supported	#N/A
ConnectorFormat	Not Supported	#N/A
Fill	Not Supported	#N/A
GroupItems	Not Supported	#N/A
HasChart	Supported	#N/A
Hyperlink	Not Supported	#N/A
IsPrintable	Supported	Supported
Line	Not Supported	#N/A
Locked	Supported	Supported
Name	Supported	Supported
Parent	Supported	#N/A
ParentGroup	Not Supported	#N/A

PictureFormat	Supported	#N/A
PictureFormat.ColorType	Supported	Supported
PictureFormat.Brightness	Supported	Supported
PictureFormat.Contrast	Supported	Supported
PictureFormat.Crop	Not Supported	#N/A
PictureFormat.CropLeft, CropTop, CropRight and CropBottom	Not Supported	Not Supported
Placement	Supported	Supported
Rotation	Supported	Not Supported
TextFrame	Not Supported	#N/A
ThreeD	Not Supported	#N/A
Title	Not Supported	Not Supported
TopLeftCell	Supported	#N/A
Left, Top, Right and Bottom	Supported	Supported
Type	Supported	Supported
Transparency	Not Supported	Not Supported
Ungroup	Not Supported	#N/A
Visible	Supported	Supported
ZOrderPosition	Supported	Supported

The signature lines can also be exported to PDF documents. Refer [Export Signature Lines](#).

## Add Digital Signatures

Digital signatures can be added to Excel spreadsheet by signing the signature lines using a signing certificate which proves signer's identity. Please follow the steps mentioned in **Generate Certificate document** to generate the certificate file (.pfx).

The **Sign** method of **ISignature** interface can be used to add digital signatures. In order to commit signatures, the workbook should be saved with xlsx or xlsxm extension. A workbook containing digital signatures is 'marked as final' to discourage editing.

Refer to the following example code to add digital signatures in a worksheet.

C#

```
//create a new workbook
var workbook = new Workbook();

//add signature line
ISignature signature = workbook.Signatures.AddSignatureLine(
```

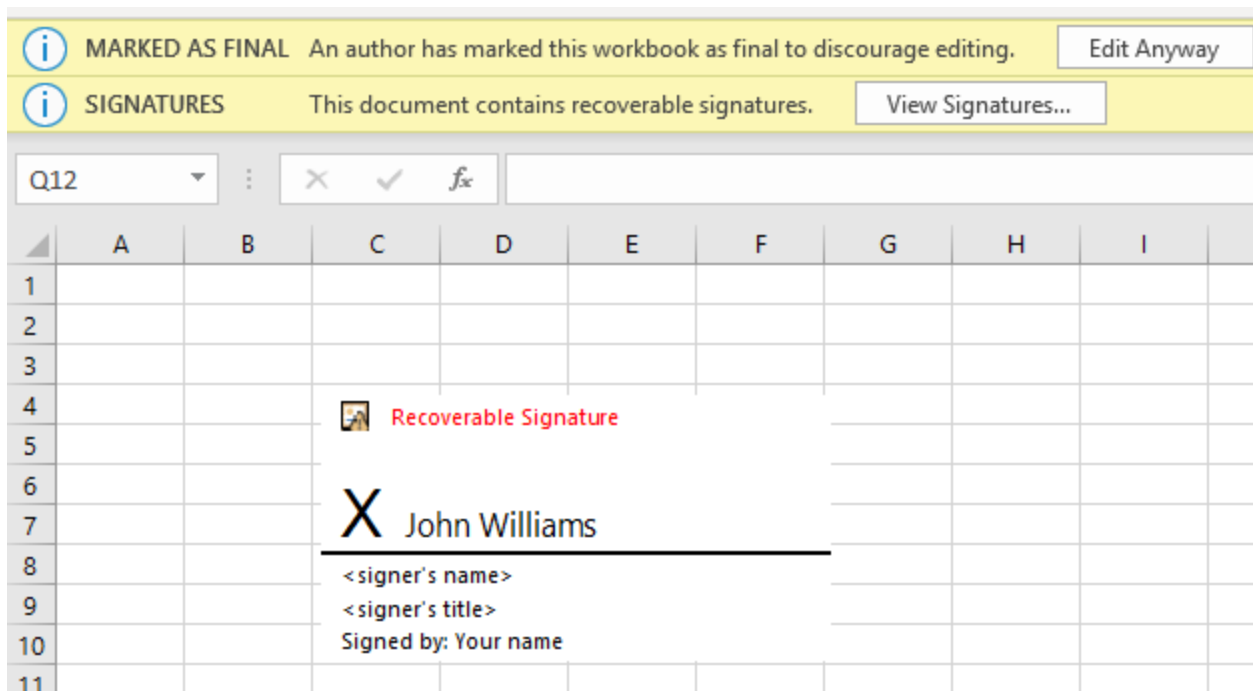
```
        workbook.ActiveSheet, 100.0, 50.0);
ISignatureSetup setup = signature.Setup;
setup.ShowSignDate = true;
setup.AllowComments = true;
setup.SigningInstructions = "<your signing instructions>";
setup.SuggestedSigner = "<signer's name>";
setup.SuggestedSignerEmail = "example@microsoft.com";
setup.SuggestedSignerLine2 = "<signer's title>";

//add signature details
var details = new SignatureDetails
{
    Address1 = "<your address>",
    Address2 = "<address 2>",
    SignatureComments = "Final",
    City = "<your city>",
    StateOrProvince = "<your state or province>",
    PostalCode = "<your postal code>",
    CountryName = "<your country or region>",
    ClaimedRole = "<your role>",
    CommitmentTypeDescription = "Approved",
    CommitmentTypeQualifier = "Final"
};

//initialize certificate
var cert = new X509Certificate2("GcExcelTest.pfx", "test@123");
signature.Sign(cert, "John Williams", details);

//save to an excel file to commit signatures
workbook.Save("digitalsignatures.xlsx");
```

The below image shows digital signature in Excel:



## Add Non Visible Signatures

You can also add invisible digital signatures to a workbook by using **AddNonVisibleSignature** method of **ISignatureSet** interface. The non visible digital signatures do not appear in any worksheet. However, they can be viewed by clicking 'View Signatures' dialog in Excel.

Refer to the following example code to add non visible signatures in a workbook.

```
C#
//create a new workbook
var workbook = new Workbook();

//add non visible signatures
ISignature signature = workbook.Signatures.AddNonVisibleSignature();
var details = new SignatureDetails
{
    Address1 = "<your address>",
    Address2 = "<address 2>",
    SignatureComments = "Final",
    City = "<your city>",
    StateOrProvince = "<your state or province>",
    PostalCode = "<your postal code>",
    CountryName = "<your country or region>",
    ClaimedRole = "<your role>",
    CommitmentTypeDescription = "Approved",
    CommitmentTypeQualifier = "Final"
};
```



```
var cert = new X509Certificate2("GcExcelTest.pfx", "test@123");
signature.Sign(cert, details);

//save to an excel file
workbook.Save("nonvisiblesignatures.xlsx");
```

## Countersign Signatures

A digitally signed workbook becomes read-only. When it is opened again in GcExcel, its digital signatures must be preserved before closing it. To achieve this:

### Countersign the Workbook

A digitally signed workbook should be countersigned if it is opened and any modification is done to it. Otherwise, the existing signatures are removed after saving the workbook as xlsx or xlsxm. The **Countersign** method of **ISignature** interface can be used to countersign a signature using the same certificate.

Refer to the following example code to open a digitally signed workbook and countersign it after modifying the worksheet.

```
C#

//open a digitally signed workbook
workbook.Open("signsignaturelines.xlsx");

//modify the worksheet
workbook.Worksheets[0].Range["A1"].Value = "Modified";

//countersign using same certificate
workbook.Signatures[0].Countersign(cert);

//save to an excel file
workbook.Save("countersign.xlsx");
```

### Open the Workbook in Digital Signature Only Mode

A digitally signed workbook can be opened in digital signature only mode by using **DigitalSignatureOnly** property in **XlsxOpenOptions** class. In this mode, you can perform the following operations while preserving existing signatures:

- Sign existing signature lines
- Remove signatures from signed signature lines
- Add and Remove non visible signatures

Refer to the following example code to open a digitally signed workbook in digital signature only mode and add non visible signatures to it.

```
C#

workbook.Open("signsignaturelines.xlsx");
```

```
//use DigitalSignatureOnly mode, because the workbook was already signed.
//if you don't open it with digital signature only mode,
//all existing signatures will be removed after saving the workbook.
XlsxOpenOptions openOption = new XlsxOpenOptions { DigitalSignatureOnly = true };

//add signature to this workbook
var signature = workbook.Signatures.AddNonVisibleSignature();
signature.Sign(cert, details);

//commit signatures
workbook.Save("AddNonVisibleSignatureToSignedWorkbook.xlsx");
```

## Verify Digital Signatures

GcExcel allows you to verify digital signatures by using **IsValid** property of **ISignature** interface. You can also extract the results of certificate verification by using **X509ChainStatusFlags** enumeration.

Refer to the following example code to verify digital signatures in a signed workbook.

```
C#

//create a new workbook
var workbook = new Workbook();
workbook.Open("digitalsignatures.xlsx");
ISignatureSet signatures = workbook.Signatures;

bool signed = false;
bool valid = false;
X509Certificate2 certificate = null;

// Verify signature
foreach (var signature in signatures)
{
    if (signature.IsSigned)
    {
        signed = true;
        certificate = signature.Details.SignatureCertificate;
        valid = signature.IsValid;
        break;
    }
}

// Verify certificate
if (certificate != null)
{
    var status = X509ChainStatusFlags.NoError;

    // build the certificate chain
```

```
var chain = new X509Chain();
bool certValid = chain.Build(certificate);

// inspect the results
if (!certValid)
{
    var chainStatus = chain.ChainStatus;
    for (var i = 0; i < chainStatus.Length; i++)
    {
        status |= chainStatus[i].Status;
    }
}

// Extract the certificate verification result
var isCertificateExpired = status.HasFlag(X509ChainStatusFlags.NotTimeValid);
var isCertificateRevoked = status.HasFlag(X509ChainStatusFlags.Revoked);
var isCertificateUntrusted = status.HasFlag(X509ChainStatusFlags.UntrustedRoot);

Console.WriteLine("Expired:" + isCertificateExpired);
Console.WriteLine("Revoked:" + isCertificateRevoked);
Console.WriteLine("UnTrusted:" + isCertificateUntrusted);
}
```

## Remove Digital Signatures

GcExcel allows you to remove digital signatures from a signed signature line by using **Delete** method of **ISignature** interface. The signature line is retained but can be deleted separately (as explained above).

Refer to the following example code to delete digital signatures from signed signature line in a workbook.

C#

```
//create a new workbook
var workbook = new Workbook();

// This file contains 1 signed signature line and
// a not signed signature line.
workbook.Open("signsignaturelines.xlsx");

//use DigitalSignatureOnly mode, because the workbook was already signed
XlsxOpenOptions openOption = new XlsxOpenOptions { DigitalSignatureOnly = true };

//remove signature of signed signature line
foreach (var signature in workbook.Signatures)
{
    if (signature.IsSignatureLine && signature.IsSigned)
    {
        //remove digital signature.
    }
}
```

```
//the signature line will not be removed from the SignatureSet
//in digital signature only mode.
//because signature lines are shapes
signature.Delete();
break;
}
}

//commit signatures
workbook.Save("deletesignaturefromsignatureline.xlsx");
```



**Note:** The signature formats observed in this feature have been tested with following versions:

#### Target Office version

The office version used to observe file structures when developing this feature is Office 365, build 16.0.12228.

This version can be observed by using `SignatureDetails.ApplicationVersion` property.

#### Minimum Office version

The minimum Office version required to open the signed workbook is Office 2013.

## Limitations

- Only Microsoft Office signature lines are supported.
- Only PFX certificates are supported.
- When signing a workbook, .NET Framework 4.6.2 or higher is required if the private key is DSA. ECDSA is not supported.
- Emf image files are not supported. Hence, when exporting signature lines, the signature image is skipped if it is in emf format. The preview images are also emf images. Hence, placeholder preview images are exported instead.
- The `DigitalSignatureOnly` mode generates corrupted workbooks if .NET Core 2.x framework is used. The workaround is to fix the generated workbook with zip archive libraries or tools (Do NOT use `System.IO.Packaging.Package` or `System.IO.Compression.ZipArchive`, because this problem is caused by these classes). However, this issue does not exist on .NET Framework or .NET Core 3.x.

## Formulas

GcExcel .NET provides you with the ability to create and use formulas to carry out complex calculations on numerical data residing in cells or a range of cells. You can also use some built-in functions and operators to generate formulas and calculate values in 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.

- [Formula Parser](#)
- [Formula Functions](#)
- [Set Formula to Range](#)
- [Set Table Formula](#)
- [Set Array Formula](#)
- [Precedents and Dependents](#)
- [Iterative Calculation](#)
- [Cross Workbook Formula](#)

## Formula Parser

GcExcel provides **GrapeCity.Documents.Excel.Expressions** library which allows you to parse formula expressions. The formula expressions are exposed at semantic model level so that you can create, visit and modify the formulas by using syntax tree. The **FormulaSyntaxTree** class represents a formula and is the entry point for formula expressions API.

### Syntax Tree

The syntax tree represents semantic model of formulas. The **Parse** method of **FormulaSyntaxTree** class can be used to get syntax tree from text. However, the text should not start with "=" and should not be surrounded with "{= }". The **Root** property of **FormulaSyntaxTree** class can be used to get the root element of syntax tree. An empty syntax tree can be created by using **FormulaSyntaxTree** constructor.

Refer to the following example code to generate a formula with syntax tree.

C#

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

// Build syntax tree
var multiply = new OperatorNode(OperatorKind.Multiply);
var a1 = new Reference
{
    Row = 0,
    Column = 0
};
var a2 = new Reference
{
    Row = 1,
```

```
        Column = 0
    };
    multiply.Children.Add(new ReferenceNode(a1));
    multiply.Children.Add(new ReferenceNode(a2));

    var tree = new FormulaSyntaxTree { Root = multiply };

    // Generates A1*A2
    workbook.ActiveSheet.Range["A1"].Value = "'=" + tree.ToString();

    //save to an excel file
    workbook.Save("generateformula.xlsx");
```

## Syntax Node

The **SyntaxNode** class represents a node in the syntax tree. The **Children** property can be used to get children of a non-terminal node. If the type of syntax node is a terminal node, then this collection is read-only. Similar to syntax tree, the **Parse** method of **SyntaxNode** class can be used to get syntax node from text. An empty syntax node can be created by using **SyntaxNode** constructor.

Refer to the following example code to parse formula, modify the syntax tree by replacing the child of syntax node and convert it to a string.

```
C#

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

var originalFormula = @"LET(AppUptime,NOW()-DATE(2020,4,17)+366, YEAR(AppUptime)-1900-1
& "" years""";

// Replace NOW() with fixed date

// Get syntax tree
var syntaxTree = FormulaSyntaxTree.Parse(originalFormula);

// Find
var nowFunction = new FunctionNode("NOW");

// Replacement
var valentine2021 = new FunctionNode("DATE");
valentine2021.Children.Add(new NumberNode(2021));
valentine2021.Children.Add(new NumberNode(2));
valentine2021.Children.Add(new NumberNode(14));

// Find and replace
void replaceNode(SyntaxNode lookIn, SyntaxNode find, SyntaxNode replacement)
{
    var children = lookIn.Children;
```

```
for (var i = 0; i < children.Count; i++)
{
    var child = children[i];
    if (child.Equals(find))
    {
        children[i] = replacement;
    }
    else
    {
        replaceNode(child, find, replacement);
    }
}

replaceNode(syntaxTree.Root, nowFunction, valentine2021);

// Output original and replaced
var sheet1 = workbook.ActiveSheet;
sheet1.Range["A1"].Value = "Original";
sheet1.Range["A2"].Value = "'=" + originalFormula.ToString();
sheet1.Range["A3"].Value = "Replaced";
sheet1.Range["A4"].Value = "'=" + syntaxTree.ToString();

// Arrange
sheet1.Range["A:A"].EntireColumn.AutoFit();

//save to an excel file
workbook.Save("modifyformula.xlsx");
```

## Parse and Unparse Options

The **ParseContext** and **UnparseContext** classes contain options for converting strings to **FormulaSyntaxTree** and vice versa respectively. The **BaseRow** and **BaseColumn** properties can be used to specify the location of formula and **IsR1C1** property can be used to specify the reference style.

Refer to the following example code to specify base row, base column and R1C1 reference style in options.

```
C#

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

// Convert R1C1 to A1
var r1c1Formula = "R1C:R8C[4]*9";
// At H2
var formulaRow = 1;
var formulaColumn = 7;
```

```
// Parse
var rlc1Option = new ParseContext { IsR1C1 = true };
var syntaxTree = FormulaSyntaxTree.Parse(rlc1Formula, rlc1Option);

// ToString
// Specify BaseRow and BaseColumn in a1Option.
// Because row and column are absolute index in A1 format.
var a1Option = new UnParseContext
{
    BaseColumn = formulaColumn,
    BaseRow = formulaRow
};
var converted = syntaxTree.ToString(a1Option);

// Output
var sheet1 = workbook.ActiveSheet;
sheet1.Range["A1"].Value = "Original formula (at H2)";
sheet1.Range["A2"].Value = "'=' + rlc1Formula.ToString();
sheet1.Range["A3"].Value = "Converted";
sheet1.Range["A4"].Value = "'=' + converted.ToString();

// Arrange
sheet1.Range["A:A"].EntireColumn.AutoFit();

//save to an excel file
workbook.Save("parseandformatoptions.xlsx");
```

Refer to the following example code to parse formula and then print the syntax tree.

C#

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

const string Formula = "RAND()>0.5+0.001";

// Get syntax tree
var syntaxTree = FormulaSyntaxTree.Parse(Formula);

// Flatten nodes
var displayItems = new List<(string TypeName, int IndentLevel, string Content)>();

void flatten(SyntaxNode node, int level)
{
    displayItems.Add((node.GetType().Name, level, node.ToString()));
    foreach (var child in node.Children)
    {
        flatten(child, level + 1);
    }
}
```



```

    }
}

flatten(syntaxTree.Root, 0);

// Output
var sheet1 = workbook.Worksheets["Sheet1"];
sheet1.ShowRowOutline = false;
sheet1.OutlineColumn.ColumnIndex = 1;
sheet1.OutlineColumn.CollapseIndicator = new
ImageSource(GetResourceStream("decreaseIndicator.png"), ImageType.PNG);
sheet1.OutlineColumn.ExpandIndicator = new
ImageSource(GetResourceStream("increaseIndicator.png"), ImageType.PNG);

// Header
sheet1.Range["A1"].Value = "Formula";
sheet1.Range["B1"].Value = "Syntax node";
sheet1.Range["C1"].Value = "Part";

// Values
sheet1.Range["A2"].Value = "'" + Formula;
for (var i = 0; i < displayItems.Count; i++)
{
    var item = displayItems[i];
    var text = "'" + item.TypeName;

    sheet1.Range[i + 1, 1].Value = text;
    sheet1.Range[i + 1, 1].IndentLevel = item.IndentLevel;
    sheet1.Range[i + 1, 2].Value = "'" + item.Content;
}

// Arrange
sheet1.Range["A:C"].EntireColumn.AutoFit();
sheet1.Range["B:B"].EntireColumn.ColumnWidthInPixel += 40;

//save to an excel file
workbook.Save("printformulasyntax.xlsx");

```

### Other Classes in GrapeCity.Documents.Excel.Expressions Library

The **ReferenceNode** class represents a reference expression in the syntax tree.


The **Reference** class represents a range reference in formula. The reference can be across a cell, range, cross-worksheet, cross-worksheet 3D or cross-workbook.



**Note:** If a row or column index is relative, **BaseRow** or **BaseColumn** properties should be used to convert to absolute index.

The **WorkbookReference** class is an immutable class which represents a reference to an external workbook by name or

local file path. If the workbook reference is from file path, the **BaseDirectory** property contains the directory information.

 **Note:** The path separator is platform specific and affects the result of workbook reference. For example, 'C:\Temp\[Book1.xlsx]Sheet1!A2 is a valid reference on Windows but invalid on Linux.

For example, the parsed object for a workbook referenced by name: *[Book1]Sheet1!A2* will look like below:

C#

```
new ReferenceNode(  
    new Reference {  
        Workbook=WorkbookReference.FromName("Book1"),  
        WorksheetName="Sheet1",  
        Row=1,  
        Column=0  
    }  
);
```

The parsed object for a workbook referenced by file path: 'C:\Temp\[Book1.xlsx]Sheet1!A2 will look like below:

C#

```
new ReferenceNode(  
    new Reference {  
        Workbook=WorkbookReference.FromFilePath(@"C:\Temp\Book1.xlsx"),  
        WorksheetName="Sheet1",  
        Row=1,  
        Column=0  
    }  
)
```

The **FunctionNode** class represents a function invocation expression in the syntax tree.

For example, the parsed object for Excel formula: *COUNTIF(A:A,"\*?")* will look like below:

C#

```
new FunctionNode("COUNTIF") {  
    Children = {  
        new ReferenceNode(  
            new Reference {  
                HasRow=false, LastColumn=0  
            }  
        ),  
        new TextNode("*?")  
    }  
};
```

The **NameNode** class represents the name in a syntax tree.

For example, the parsed object for a workbook referenced by name: '*[BuildingSales]JanIn2021!RawData*' will look like

below:

```
C#  
new NameNode("RawData", WorkbookReference.FromName("BuildingSales"), "JanIn2021", null);
```

The parsed object for a workbook referenced by file path: 'E:\[BuildingSales.xlsx]JanIn2021!RawData' will look like below:

```
C#  
new NameNode("RawData", WorkbookReference.FromFilePath(@"E:\BuildingSales.xlsx"),  
"JanIn2021", null);
```

The **ErrorNode** class represents an error literal node in the syntax tree. The following error types are not supported:

- #BLOCKED!
- #CALC!
- #CONNECT!
- #FIELD!
- #SPILL!
- #UNKNOWN!
- #REF! error is parsed to ReferenceNode

The **ArrayNode** class represents an array literal in the syntax tree. There are following array constraints:

- The length of array must be > 0
- Elements can be Double, String, Boolean or CalcError. Primitive number types are converted to double implicitly.
- The lower bound of each ranks must be 0
- The array and Elements can't be null

To know more about other classes, please refer [GrapeCity.Documents.Excel.Expressions](#) API documentation.

### Limitations

- Dynamic array formulas are not supported.
- GetHashCode method of FormulaSyntaxTree and SyntaxNode class are not supported. They return constant values because all fields are mutable.
- GcExcel does not support resolving workbook index defined in OpenXML or JSON file storage. They are treated as workbook reference by name.

## Formula Functions

GcExcel.NET provides support for the following built-in functions, listed alphabetically.

Function Name	Function Category	Function Description
ABS	Math and Trigonometry	Returns the absolute value of a number.
ACCRINT	Financial	Returns the accrued interest for a security that pays periodic interest.
ACCRINTM	Financial	Returns the accrued interest for a security that pays interest at maturity.

ACOS	Math and Trigonometry	Returns the arccosine of a number.
ACOSH	Math and Trigonometry	Returns the inverse hyperbolic cosine of a number.
ACOT	Math and Trigonometry	Returns the arccotangent of a number.
ACOTH	Math and Trigonometry	Returns the hyperbolic arccotangent of a number.
ADDRESS	Lookup and Reference	Returns a reference as text to a single cell in a worksheet.
AMORDEGRC	Financial	Returns the depreciation for each accounting period by using a depreciation coefficient.
AMORLINC	Financial	Returns the depreciation for each accounting period.
AND	Logical	Returns TRUE if all of its arguments are TRUE.
ARABIC	Math and Trigonometry	Converts a Roman number to Arabic, as a number.
AREAS	Lookup and Reference	Returns the number of areas in a reference.
ASIN	Math and Trigonometry	Returns the arcsine of a number.
ASINH	Math and Trigonometry	Returns the inverse hyperbolic sine of a number.
ATAN	Math and Trigonometry	Returns the arctangent of a number.
ATAN2	Math and Trigonometry	Returns the arctangent from x- and y-coordinates.
ATANH	Math and Trigonometry	Returns the inverse hyperbolic tangent of a number.
AVEDEV	Statistical	Returns the average of the absolute deviations of data points from their mean.
AVERAGE	Statistical	Returns the average of its arguments.
AVERAGEA	Statistical	Returns the average of its arguments, including numbers, text, and logical values.
AVERAGEIF	Statistical	Returns the average (arithmetic mean) of all the cells in a range that meet a given criteria.
AVERAGEIFS	Statistical	Returns the average (arithmetic mean) of all cells that meet multiple criteria.
BAHTTEXT	Text	Converts a number to text, using the ฿ (baht) currency format.

BASE	Math and Trigonometry	Converts a number into a text representation with the given radix (base).
BESSELI	Engineering	Returns the modified Bessel function $I_n(x)$ .
BESSELJ	Engineering	Returns the Bessel function $J_n(x)$ .
BESSELK	Engineering	Returns the modified Bessel function $K_n(x)$ .
BESSELY	Engineering	Returns the Bessel function $Y_n(x)$ .
BETA.DIST	Statistical	Returns the beta cumulative distribution function.
BETA.INV	Statistical	Returns the inverse of the cumulative distribution function for a specified beta distribution.
BETADIST	Compatibility	Returns the beta cumulative distribution function.
BETAINV	Compatibility	Returns the inverse of the cumulative distribution function for a specified beta distribution.
BIN2DEC	Engineering	Converts a binary number to decimal.
BIN2HEX	Engineering	Converts a binary number to hexadecimal.
BIN2OCT	Engineering	Converts a binary number to octal.
BINOM.DIST	Statistical	Returns the individual term binomial distribution probability.
BINOM.DIST.RANGE	Statistical	Returns the probability of a trial result using a binomial distribution.
BINOM.INV	Statistical	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value.
BINOMDIST	Compatibility	Returns the individual term binomial distribution probability.
BITAND	Engineering	Returns a 'Bitwise And' of two numbers.
BITLSHIFT	Engineering	Returns a value number shifted left by shift_amount bits.
BITOR	Engineering	Returns a bitwise OR of 2 numbers.
BITRSHIFT	Engineering	Returns a value number shifted right by shift_amount bits.
BITXOR	Engineering	Returns a bitwise 'Exclusive Or' of two numbers.
CEILING	Math and Trigonometry	Rounds a number to the nearest integer or to the nearest multiple of significance.
CEILING.MATH	Math and Trigonometry	Rounds a number up, to the nearest integer or to the nearest multiple of significance.
CELL	Information	Returns information about the formatting, location, or contents of a cell.
CHAR	Text	Returns the character specified by the code number.
CHIDIST	Compatibility	Returns the one-tailed probability of the chi-squared distribution.
CHIINV	Compatibility	Returns the inverse of the one-tailed probability of the chi-squared

		distribution.
CHISQ.DIST	Statistical	Returns the cumulative beta probability density function.
CHISQ.DIST.RT	Statistical	Returns the one-tailed probability of the chi-squared distribution.
CHISQ.INV	Statistical	Returns the cumulative beta probability density function.
CHISQ.INV.RT	Statistical	Returns the inverse of the one-tailed probability of the chi-squared distribution.
CHISQ.TEST	Statistical	Returns the test for independence.
CHITEST	Compatibility	Returns the test for independence.
CHOOSE	Lookup and reference	Chooses a value from a list of values.
CLEAN	Text	Removes all nonprintable characters from text.
CODE	Text	Returns a numeric code for the first character in a text string.
COLUMN	Lookup and reference	Returns the column number of a reference.
COLUMNS	Lookup and reference	Returns the number of columns in a reference.
COMBIN	Math and trigonometry	Returns the number of combinations for a given number of objects.
COMBINA	Math and trigonometry	Returns the number of combinations for a specified number of items including the repetitions.
COMPLEX	Engineering	Converts real and imaginary coefficients into a complex number.
CONCAT	Text	Combines the text from multiple ranges and/or strings, but it doesn't provide the delimiter or IgnoreEmpty arguments.
CONCATENATE	Text	Joins several text items into one text item.
CONFIDENCE	Compatibility	Returns the confidence interval for a population mean.
CONFIDENCE.NORM	Statistical	Returns the confidence interval for a population mean.
CONFIDENCE.T	Statistical	Returns the confidence interval for a population mean, using a Student's t distribution.
CONVERT	Engineering	Converts a number from one measurement system to another.
CORREL	Statistical	Returns the correlation coefficient between two data sets.
COS	Math and trigonometry	Returns the cosine of a number.
COSH	Math and trigonometry	Returns the hyperbolic cosine of a number.
COT	Math and trigonometry	Returns the cotangent of an angle.
COTH	Math and trigonometry	Returns the hyperbolic cotangent of an angle.

COUNT	Statistical	Counts how many numbers are in the list of arguments.
COUNTA	Statistical	Counts how many values are in the list of arguments.
COUNTBLANK	Statistical	Counts the number of blank cells within a range.
COUNTIF	Statistical	Counts the number of cells within a range that meet the given criteria.
COUNTIFS	Statistical	Counts the number of cells within a range that meet multiple criteria.
COUPDAYBS	Financial	Returns the number of days from the beginning of the coupon period to the settlement date.
COUPDAYS	Financial	Returns the number of days in the coupon period that contains the settlement date.
COUPDAYSNC	Financial	Returns the number of days in the coupon period that contains the settlement date.
COUPNCD	Financial	Returns the next coupon date after the settlement date.
COUPNUM	Financial	Returns the number of coupons payable between the settlement date and maturity date.
COUPPCD	Financial	Returns the previous coupon date before the settlement date.
COVAR	Compatibility	Returns covariance, the average of the products of paired deviations.
COVARIANCE.P	Statistical	Returns covariance, the average of the products of paired deviations.
COVARIANCE.S	Statistical	Returns the sample covariance, the average of the products deviations for each data point pair in two data sets.
CRITBINOM	Compatibility	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value.
CSC	Math and trigonometry	Returns the cosecant of an angle.
CSCH	Math and trigonometry	Returns the hyperbolic cosecant of an angle.
CUMIPMT	Financial	Returns the cumulative interest paid between two periods.
CUMPRINC	Financial	Returns the cumulative principal paid on a loan between two periods.
DATE	Date and time	Returns the serial number of a particular date.
DATEDIF	Date and time	Calculates the number of days, months, or years between two dates. This function is useful in formulas where you need to calculate an age.
DATEVALUE	Date and time	Converts a date in the form of text to a serial number.
DAVERAGE	Database	Returns the average of selected database entries.
DAY	Date and time	Converts a serial number to a day of the month.
DAYS	Date and time	Returns the number of days between two dates.
DAYS360	Date and time	Calculates the number of days between two dates based on a 360-day

		year.
DB	Financial	Returns the depreciation of an asset for a specified period by using the fixed-declining balance method.
DCOUNT	Database	Changes half-width (single-byte) English letters or katakana within a character string to full-width (double-byte) characters.
DCOUNTA	Database	Counts nonblank cells in a database.
DDB	Financial	Returns the depreciation of an asset for a specified period by using the double-declining balance method or some other method that you specify.
DEC2BIN	Engineering	Converts a decimal number to binary.
DEC2HEX	Engineering	Converts a decimal number to hexadecimal.
DEC2OCT	Engineering	Converts a decimal number to octal.
DECIMAL	Math and trigonometry	Converts a text representation of a number in a given base into a decimal number.
DEGREES	Math and trigonometry	Converts radians to degrees.
DELTA	Engineering	Tests whether two values are equal.
DEVSQ	Statistical	Returns the sum of squares of deviations.
DGET	Database	Extracts from a database a single record that matches the specified criteria.
DISC	Financial	Returns the discount rate for a security.
DMAX	Database	Returns the maximum value from selected database entries.
DMIN	Database	Returns the minimum value from selected database entries.
DOLLAR	Text	Converts a number to text, using the \$ (dollar) currency format.
DOLLARDE	Financial	Converts a dollar price, expressed as a fraction, into a dollar price, expressed as a decimal number.
DOLLARFR	Financial	Converts a dollar price, expressed as a decimal number, into a dollar price, expressed as a fraction.
DPRODUCT	Database	Multiplies the values in a particular field of records that match the criteria in a database.
DSTDEV	Database	Estimates the standard deviation based on a sample of selected database entries.
DSTDEVP	Database	Calculates the standard deviation based on the entire population of selected database entries.
DSUM	Database	Adds the numbers in the field column of records in the database that match the criteria.



DURATION	Financial	Returns the annual duration of a security with periodic interest payments.
DVAR	Database	Estimates variance based on a sample from selected database entries.
DVARP	Database	Calculates variance based on the entire population of selected database entries.
EDATE	Date and time	Returns the serial number of the date that is the indicated number of months before or after the start date.
EFFECT	Financial	Returns the effective annual interest rate.
ENCODEURL	Web	Returns a URL-encoded string.
EOMONTH	Date and time	Returns the serial number of the last day of the month before or after a specified number of months.
ERF	Engineering	Returns the error function.
ERF.PRECISE	Engineering	Returns the error function.
ERFC	Engineering	Returns the complementary error function.
ERFC.PRECISE	Engineering	Returns the complementary ERF function integrated between x and infinity.
ERROR.TYPE	Information	Returns a number corresponding to an error type.
EUROCONVERT	Add-in and Automation	Converts a number to euros, converts a number from euros to a euro member currency, or converts a number from one euro member currency to another by using the euro as an intermediary (triangulation).
EVEN	Math and Trigonometry	Rounds a number up to the nearest even integer.
EXACT	Text	Checks to see if two text values are identical.
EXP	Math and Trigonometry	Returns e raised to the power of a given number.
EXPON.DIST	Statistical	Returns the exponential distribution.
EXPONDIST	Compatibility	Returns the exponential distribution.
F.DIST	Statistical	Returns the F probability distribution
F.DIST.RT	Statistical	Returns the F probability distribution
F.INV	Statistical	Returns the inverse of the F probability distribution.
F.INV.RT	Statistical	Returns the inverse of the F probability distribution.
F.TEST	Statistical	Returns the result of an F-test.
FACT	Math and trigonometry	Returns the factorial of a number.
FACTDOUBLE	Math and	Returns the double factorial of a number.

	trigonometry	
FALSE	Logical	Returns the logical value FALSE.
FDIST	Compatibility	Returns the F probability distribution.
FIND	Text	Finds one text value within another (case-sensitive).
FINDBS	Text	Finds one text value within another (case-sensitive).
FINV	Statistical	Returns the inverse of the F probability distribution.
FISHER	Statistical	Returns the Fisher transformation.
FISHERINV	Statistical	Returns the inverse of the Fisher transformation.
FIXED	Text	Formats a number as text with a fixed number of decimals.
FLOOR	Compatibility	Rounds a number down, toward zero.
FLOOR.MATH	Math and trigonometry	Rounds a number down, to the nearest integer or to the nearest multiple of significance.
FLOOR.PRECISE	Math and trigonometry	Rounds a number the nearest integer or to the nearest multiple of significance. Regardless of the sign of the number, the number is rounded up.
FORECAST	Statistical	Returns a value along a linear trend.
FORMULATEXT	Lookup and reference	Returns the formula at the given reference as text.
FREQUENCY	Statistical	Returns a frequency distribution as a vertical array.
FTEST	Compatibility	Returns the result of an F-test.
FV	Financial	Returns the future value of an investment.
FVSCHEDULE	Financial	Returns the future value of an initial principal after applying a series of compound interest rates.
GAMMA	Statistical	Returns the Gamma function value.
GAMMA.DIST	Statistical	Returns the Gamma distribution.
GAMMA.INV	Statistical	Returns the inverse of the gamma cumulative distribution.
GAMMADIST	Compatibility	Returns the gamma distribution.
GAMMAINV	Compatibility	Returns the inverse of the gamma cumulative distribution.
GAMMALN	Statistical	Returns the natural logarithm of the gamma function, $G(x)$ .
GAMMALN.PRECISE	Statistical	Returns the natural logarithm of the gamma function, $G(x)$ .
GAUSS	Statistical	Returns 0.5 less than the standard normal cumulative distribution.
GCD	Math and trigonometry	Returns the greatest common divisor.
GEOMEAN	Statistical	Returns the geometric mean.
GESTEP	Engineering	Tests whether a number is greater than a threshold value.

GROWTH	Statistical	Returns values along an exponential trend.
HARMEAN	Statistical	Returns the harmonic mean.
HEX2BIN	Engineering	Converts a hexadecimal number to binary.
HEX2DEC	Engineering	Converts a hexadecimal number to decimal.
HEX2OCT	Engineering	Converts a hexadecimal number to octal.
HLOOKUP	Lookup and reference	Looks in the top row of an array and returns the value of the indicated cell.
HOUR	Date and time	Converts a serial number to an hour.
HYPERLINK	Lookup and reference	Creates a shortcut or jump that opens a document stored on a network server, an intranet, or the Internet.
HYPGEOM.DIST	Statistical	Returns the hypergeometric distribution.
HYPGEOMDIST	Compatibility	Returns the hypergeometric distribution.
IF	Logical	Specifies a logical test to perform
IFERROR	Logical	Returns a value you specify if a formula evaluates to an error; otherwise, returns the result of the formula.
IFNA	Logical	Returns the value you specify if the expression resolves to #N/A, otherwise returns the result of the expression.
IFS	Logical	Checks whether one or more conditions are met and returns a value that corresponds to the first TRUE condition..
IMABS	Engineering	Returns the absolute value (modulus) of a complex number.
IMAGINARY	Engineering	Returns the imaginary coefficient of a complex number.
IMARGUMENT	Engineering	Returns the argument theta, an angle expressed in radians.
IMCONJUGATE	Engineering	Returns the complex conjugate of a complex number.
IMCOS	Engineering	Returns the cosine of a complex number.
IMCOSH	Engineering	Returns the hyperbolic cosine of a complex number.
IMCOT	Engineering	Returns the cotangent of a complex number.
IMCSC	Engineering	Returns the cosecant of a complex number.
IMCSCH	Engineering	Returns the hyperbolic cosecant of a complex number.
IMDIV	Engineering	Returns the quotient of two complex numbers.
IMEXP	Engineering	Returns the exponential of a complex number.
IMLN	Engineering	Returns the natural logarithm of a complex number.
IMLOG10	Engineering	Returns the base-10 logarithm of a complex number.
IMLOG2	Engineering	Returns the base-2 logarithm of a complex number.

IMPOWER	Engineering	Returns a complex number raised to an integer power.
IMPRODUCT	Engineering	Returns the product of complex numbers.
IMREAL	Engineering	Returns the real coefficient of a complex number.
IMSEC	Engineering	Returns the secant of a complex number.
IMSECH	Engineering	Returns the hyperbolic secant of a complex number.
IMSIN	Engineering	Returns the sine of a complex number.
IMSINH	Engineering	Returns the hyperbolic sine of a complex number.
IMSQRT	Engineering	Returns the square root of a complex number.
IMSUB	Engineering	Returns the difference between two complex numbers.
IMSUM	Engineering	Returns the sum of complex numbers.
IMTAN	Engineering	Returns the tangent of a complex number.
INDEX	Lookup and reference	Uses an index to choose a value from a reference or array.
INDIRECT	Lookup and reference	Returns a reference indicated by a text value.
INT	Math and trigonometry	Rounds a number down to the nearest integer.
INTERCEPT	Statistical	Returns the intercept of the linear regression line.
INTRATE	Financial	Returns the interest rate for a fully invested security.
IPMT	Financial	Returns the interest payment for an investment for a given period.
IRR	Financial	Returns the internal rate of return for a series of cash flowsReturns the internal rate of return for a series of cash flows.
ISBLANK	Information	Returns TRUE if the value is blank.
ISERR	Information	Returns TRUE if the value is any error value except #N/A.
ISERROR	Information	Returns TRUE if the value is any error value.
ISEVEN	Information	Returns TRUE if the number is even.
ISFORMULA	Information	Returns TRUE if there is a reference to a cell that contains a formula.
ISLOGICAL	Information	Returns TRUE if the value is a logical value.
ISNA	Information	Returns TRUE if the value is the #N/A error value.
ISNONTEXT	Information	Returns TRUE if the value is not text.
ISNUMBER	Information	Returns TRUE if the value is a number.
ISO.CEILING	Math and trigonometry	Returns a number that is rounded up to the nearest integer or to the nearest multiple of significance.
ISODD	Information	Returns TRUE if the number is odd.
ISOWEEKNUM	Date and time	Returns the number of the ISO week number of the year for a given

		date.
ISPMT	Financial	Calculates the interest paid during a specific period of an investment.
ISREF	Information	Returns TRUE if the value is a reference.
ISTEXT	Information	Returns TRUE if the value is text.
KURT	Statistical	Returns TRUE if the value is text.
LARGE	Statistical	Returns the k-th largest value in a data set.
LCM	Math and trigonometry	Returns the least common multiple.
LEFT	Text	Returns the leftmost characters from a text value.
LEFTBs	Text	Returns the leftmost characters from a text value.
LEN	Text	Returns the number of characters in a text string.
LENBs	Text	Returns the number of characters in a text string.
LINEST	Statistical	Returns the parameters of a linear trend.
LN	Math and trigonometry	Returns the natural logarithm of a number.
LOG	Math and trigonometry	Returns the logarithm of a number to a specified base.
LOG10	Math and trigonometry	Returns the base-10 logarithm of a number.
LOGEST	Statistical	Returns the parameters of an exponential trend.
LOGINV	Compatibility	Returns the inverse of the lognormal cumulative distribution.
LOGNORM.DIST	Statistical	Returns the cumulative lognormal distribution.
LOGNORM.INV	Statistical	Returns the inverse of the lognormal cumulative distribution.
LOGNORMDIST	Compatibility	Returns the cumulative lognormal distribution.
LOOKUP	Lookup and reference	Looks up values in a vector or array.
LOWER	Text	Converts text to lowercase.
MATCH	Lookup and reference	Looks up values in a reference or array.
MAX	Statistical	Returns the maximum value in a list of arguments.
MAXA	Statistical	Returns the maximum value in a list of arguments, including numbers, text, and logical values.
MAXIFS	Statistical	Returns the maximum value among cells specified by a given set of conditions or criteria.
MDETERM	Math and trigonometry	Returns the matrix determinant of an array.

MDURATION	Financial	Returns the Macauley modified duration for a security with an assumed par value of \$100.
MEDIAN	Statistical	Returns the median of the given numbers.
MID	Text	Returns a specific number of characters from a text string starting at the position you specify.
MIDBs	Text	Returns a specific number of characters from a text string starting at the position you specify.
MIN	Statistical	Returns the minimum value in a list of arguments.
MINA	Statistical	Returns the smallest value in a list of arguments, including numbers, text, and logical values.
MINIFS	Statistical	Returns the minimum value among cells specified by a given set of conditions or criteria.
MINUTE	Date and time	Converts a serial number to a minute.
MINVERSE	Math and trigonometry	Returns the matrix inverse of an array.
MIRR	Financial	Returns the internal rate of return where positive and negative cash flows are financed at different rates.
MMULT	Math and trigonometry	Returns the matrix product of two arrays.
MOD	Math and trigonometry	Returns the remainder from division.
MODE	Compatibility	Returns the most common value in a data set.
MODE.MULT	Statistical	Returns a vertical array of the most frequently occurring, or repetitive values in an array or range of data.
MODE.SNGL	Statistical	Returns the most common value in a data set.
MONTH	Date and time	Converts a serial number to a month.
MROUND	Math and trigonometry	Returns a number rounded to the desired multiple.
MULTINOMIAL	Math and trigonometry	Returns the multinomial of a set of numbers.
MUNIT	Math and trigonometry	Returns the unit matrix or the specified dimension.
N	Information	Returns a value converted to a number.
NA	Information	Returns the error value #N/A.
NEGBINOM.DIST	Statistical	Returns the negative binomial distribution.
NEGBINOMDIST	Compatibility	Returns the negative binomial distribution.

NETWORKDAYS	Date and time	Returns the number of whole workdays between two dates.
NETWORKDAYS.INTL	Date and time	Returns the number of whole workdays between two dates using parameters to indicate which and how many days are weekend days.
NOMINAL	Financial	Returns the annual nominal interest rate.
NORM.DIST	Statistical	Returns the normal cumulative distribution.
NORM.INV	Compatibility	Returns the inverse of the normal cumulative distribution.
NORM.S.DIST	Statistical	Returns the standard normal cumulative distribution.
NORM.S.INV	Statistical	Returns the inverse of the standard normal cumulative distribution.
NORMDIST	Compatibility	Returns the normal cumulative distribution.
NORMINV	Statistical	Returns the inverse of the normal cumulative distribution.
NORMSDIST	Compatibility	Returns the standard normal cumulative distribution.
NORMSINV	Compatibility	Returns the inverse of the standard normal cumulative distribution.
NOT	Logical	Reverses the logic of its argument.
NOW	Date and time	Returns the serial number of the current date and time.
NPER	Financial	Returns the number of periods for an investment.
NPV	Financial	Returns the net present value of an investment based on a series of periodic cash flows and a discount rate.
NUMBERVALUE	Text	Converts text to number in a locale-independent manner.
OCT2BIN	Engineering	Converts an octal number to binary.
OCT2DEC	Engineering	Converts an octal number to decimal.
OCT2HEX	Engineering	Converts an octal number to hexadecimal.
ODD	Math and trigonometry	Rounds a number up to the nearest odd integer.
ODDFPRICE	Financial	Returns the price per \$100 face value of a security with an odd first period.
ODDFYIELD	Financial	Returns the yield of a security with an odd first period.
ODDLPRICE	Financial	Returns the price per \$100 face value of a security with an odd last period.
ODDLYIELD	Financial	Returns the yield of a security with an odd last period.
OFFSET	Lookup and reference	Returns a reference offset from a given reference.
OR	Logical	Returns TRUE if any argument is TRUE.
PDURATION	Financial	Returns the number of periods required by an investment to reach a specified value.
PEARSON	Statistical	Returns the Pearson product moment correlation coefficient.

PERCENTILE	Compatibility	Returns the k-th percentile of values in a range.
PERCENTILE.EXC	Statistical	Returns the k-th percentile of values in a range, where k is in the range 0..1, exclusive.
PERCENTILE.INC	Statistical	Returns the k-th percentile of values in a range.
PERCENTRANK	Compatibility	Returns the percentage rank of a value in a data set.
PERCENTRANK.EXC	Statistical	Returns the rank of a value in a data set as a percentage (0..1, exclusive) of the data set.
PERCENTRANK.INC	Statistical	Returns the percentage rank of a value in a data set.
PERMUT	Statistical	Returns the number of permutations for a given number of objects.
PERMUTATIONA	Statistical	Returns the number of permutations for a given number of objects (with repetitions) that can be selected from the total objects.
PHI	Statistical	Returns the value of the density function for a standard normal distribution.
PI	Math and trigonometry	Returns the value of pi.
PMT	Financial	Returns the periodic payment for an annuity.
POISSON	Compatibility	Returns the Poisson distribution.
POISSON.DIST	Statistical	Returns the Poisson distribution.
POWER	Math and trigonometry	Returns the result of a number raised to a power.
PPMT	Financial	Returns the payment on the principal for an investment for a given period.
PRICE	Financial	Returns the price per \$100 face value of a security that pays periodic interest.
PRICEDISC	Financial	Returns the price per \$100 face value of a discounted security.
PRICEMAT	Financial	Returns the price per \$100 face value of a security that pays interest at maturity.
PROB	Statistical	Returns the probability that values in a range are between two limits.
PRODUCT	Math and trigonometry	Multiplies its arguments.
PROPER	Text	Capitalizes the first letter in each word of a text value.
PV	Financial	Returns the present value of an investment.
QUARTILE	Compatibility	Returns the quartile of a data set.
QUARTILE.EXC	Statistical	Returns the quartile of the data set, based on percentile values from 0..1, exclusive.
QUARTILE.INC	Statistical	Returns the quartile of a data set.



QUOTIENT	Math and trigonometry	Returns the integer portion of a division.
RADIANS	Math and trigonometry	Converts degrees to radians.
RAND	Math and trigonometry	Returns a random number between 0 and 1.
RANDBETWEEN	Math and trigonometry	Returns a random number between the numbers you specify.
RANK	Compatibility	Returns the rank of a number in a list of numbers.
RANK.AVG	Statistical	Returns the rank of a number in a list of numbers.
RANK.EQ	Statistical	Returns the rank of a number in a list of numbers.
RATE	Financial	Returns the interest rate per period of an annuity.
RECEIVED	Financial	Returns the amount received at maturity for a fully invested security.
REPLACE	Text	Replaces characters within text.
REPLACEBs	Text	Replaces characters within text
REPT	Text	Repeats text a given number of times.
RIGHT	Text	Returns the rightmost characters from a text value.
RIGHTBs	Text	Returns the rightmost characters from a text value.
ROMAN	Math and trigonometry	Converts an arabic numeral to roman, as text.
ROUND	Math and trigonometry	Rounds a number to a specified number of digits.
ROUNDDOWN	Math and trigonometry	Rounds a number down, toward zero.
ROUNDUP	Math and trigonometry	Rounds a number up, away from zero.
ROW	Lookup and reference	Returns the row number of a reference.
ROWS	Lookup and reference	Returns the number of rows in a reference.
RRI	Financial	Returns an equivalent interest rate for the growth of an investment.
RSQ	Statistical	Returns the square of the Pearson product moment correlation coefficient.
SEARCH	Text	Finds one text value within another (not case-sensitive).
SEARCHBs	Text	Finds one text value within another (not case-sensitive).
SEC	Math and trigonometry	Returns the secant of an angle.

SECH	Math and trigonometry	Returns the hyperbolic secant of an angle.
SECOND	Date and Time	Converts a serial number to a second.
SERIESSUM	Math and trigonometry	Returns the sum of a power series based on the formula.
SHEET	Information	Returns the sheet number of the referenced sheet.
SHEETS	Information	Returns the number of sheets in a reference.
SIGN	Math and trigonometry	Returns the sign of a number.
SIN	Math and trigonometry	Returns the sine of the given angle.
SINH	Math and trigonometry	Returns the hyperbolic sine of a number.
SKEW	Statistical	Returns the skewness of a distribution.
SKEW.P	Statistical	Returns the skewness of a distribution based on a population: a characterization of the degree of asymmetry of a distribution around its mean.
SLN	Financial	Returns the straight-line depreciation of an asset for one period.
SLOPE	Statistical	Returns the slope of the linear regression line.
SMALL	Statistical	Returns the k-th smallest value in a data set.
SQRT	Math and trigonometry	Returns a positive square root.
SQRTPI	Math and trigonometry	Returns the square root of (number * pi).
STANDARDIZE	Statistical	Returns a normalized value.
STDEV	Compatibility	Estimates standard deviation based on a sample.
STDEV.P	Statistical	Calculates standard deviation based on the entire population.
STDEV.S	Statistical	Estimates standard deviation based on a sample.
STDEVA	Statistical	Estimates standard deviation based on a sample, including numbers, text, and logical values.
STDEVP	Compatibility	Calculates standard deviation based on the entire population.
STDEVPA	Statistical	Calculates standard deviation based on the entire population, including numbers, text, and logical values.
STEYX	Statistical	Returns the standard error of the predicted y-value for each x in the regression.
SUBSTITUTE	Text	Substitutes new text for old text in a text string.

SUBTOTAL	Math and trigonometry	Returns a subtotal in a list or database.
SUM	Math and trigonometry	Adds its arguments.
SUMIF	Math and trigonometry	Adds the cells specified by a given criteria.
SUMIFS	Math and trigonometry	Adds the cells in a range that meet multiple criteria.
SUMPRODUCT	Math and trigonometry	Returns the sum of the products of corresponding array components.
SUMSQ	Math and trigonometry	Returns the sum of the squares of the arguments.
SUMX2MY2	Math and trigonometry	Returns the sum of the difference of squares of corresponding values in two arrays.
SUMX2PY2	Math and trigonometry	Returns the sum of the sum of squares of corresponding values in two arrays.
SUMXMY2	Math and trigonometry	Returns the sum of squares of differences of corresponding values in two arrays.
SWITCH	Logical	Evaluates an expression against a list of values and returns the result corresponding to the first matching value. If there is no match, an optional default value may be returned.
SYD	Financial	Returns the sum-of-years' digits depreciation of an asset for a specified period.
T	Text	Converts its arguments to text.
T.DIST	Statistical	Returns the Percentage Points (probability) for the Student t-distribution.
T.DIST.2T	Statistical	Returns the Percentage Points (probability) for the Student t-distribution.
T.DIST.RT	Statistical	Returns the Student's t-distribution.
T.INV	Statistical	Returns the t-value of the Student's t-distribution as a function of the probability and the degrees of freedom.
T.INV.2T	Statistical	Returns the inverse of the Student's t-distribution.
T.TEST	Statistical	Returns the probability associated with a Student's t-test.
TAN	Math and trigonometry	Returns the tangent of a number.
TANH	Math and trigonometry	Returns the hyperbolic tangent of a number.
TBILLEQ	Financial	Returns the bond-equivalent yield for a Treasury bill.

TBILLPRICE	Financial	Returns the price per \$100 face value for a Treasury bill.
TBILLYIELD	Financial	Returns the yield for a Treasury bill.
TDIST	Compatibility	Returns the Student's t-distribution.
TEXT	Text	Formats a number and converts it to text.
TEXTJOIN	Text	Combines the text from multiple ranges and/or strings, and includes a delimiter you specify between each text value that will be combined. If the delimiter is an empty text string, this function will effectively concatenate the ranges.
TIME	Date and time	Returns the serial number of a particular time.
TIMEVALUE	Date and time	Converts a time in the form of text to a serial number.
TINV	Compatibility	Returns the inverse of the Student's t-distribution.
TODAY	Date and time	Returns the serial number of today's date.
TRANSPOSE	Lookup and reference	Returns the transpose of an array.
TREND	Statistical	Returns values along a linear trend.
TRIM	Text	Removes spaces from text.
TRIMMEAN	Statistical	Returns the mean of the interior of a data set.
TRUE	Logical	Returns the logical value TRUE.
TRUNC	Math and trigonometry	Truncates a number to an integer.
TTEST	Compatibility	Returns the probability associated with a Student's t-test.
TYPE	Information	Returns a number indicating the data type of a value.
UNICHAR	Text	Returns the Unicode character that is referenced by the given numeric value.
UNICODE	Text	Returns the number (code point) that corresponds to the first character of the text.
UPPER	Text	Converts text to uppercase.
VALUE	Text	Converts a text argument to a number.
VAR	Compatibility	Estimates variance based on a sample.
VAR.P	Statistical	Calculates variance based on the entire population.
VAR.S	Statistical	Estimates variance based on a sample.
VARA	Statistical	Estimates variance based on a sample, including numbers, text, and logical values.
VARP	Compatibility	Calculates variance based on the entire population.
VARPA	Statistical	Calculates variance based on the entire population, including numbers,

		text, and logical values.
VDB	Financial	Returns the depreciation of an asset for a specified or partial period by using a declining balance method.
VLOOKUP	Lookup and reference	Looks in the first column of an array and moves across the row to return the value of a cell.
WEEKDAY	Date and time	Converts a serial number to a day of the week.
WEEKNUM	Date and time	Converts a serial number to a number representing where the week falls numerically with a year.
WEIBULL	Compatibility	Calculates variance based on the entire population, including numbers, text, and logical values.
WEIBULL.DIST	Statistical	Returns the Weibull distribution.
WORKDAY	Date and time	Returns the serial number of the date before or after a specified number of workdays.
WORKDAY.INTL	Date and time	Returns the serial number of the date before or after a specified number of workdays using parameters to indicate which and how many days are weekend days.
XIRR	Financial	Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.
XNPV	Financial	Returns the net present value for a schedule of cash flows that is not necessarily periodic.
XOR	Logical	Returns a logical exclusive OR of all arguments.
YEAR	Date and time	Converts a serial number to a year.
YEARFRAC	Date and time	Returns the year fraction representing the number of whole days between start_date and end_date.
YIELD	Financial	Returns the yield on a security that pays periodic interest.
YIELDDISC	Financial	Returns the annual yield for a discounted security; for example, a Treasury bill.
YIELDMAT	Financial	Returns the annual yield of a security that pays interest at maturity.
Z.TEST	Statistical	Returns the one-tailed probability-value of a z-test.
ZTEST	Compatibility	Returns the one-tailed probability-value of a z-test.

## Set Formula to Range

You can set formula to a cell range using the **Formula** property of the **IRange** interface.

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** 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** property of the **IWorkbook** interface.

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;
```

## Defer the Update of Dirty State for Formula Cells

The value calculated by a formula is stored in cache first and the cached result is returned upon retrieving the cell value. When a worksheet contains huge amount of data which depends on the result of formulas and the value of a cell is changed, all the formula cells are recalculated and the cached values are stored again which could degrade the performance of worksheet.

Hence, GcExcel provides **DeferUpdateDirtyState** property in **Workbook** class, which when set to true does not update the dirty state of formula cells immediately when the value of a cell is changed.

Refer to the following example code to defer the update of dirty state for formula cells.

C#

```
Workbook wb = new Workbook();
wb.Open("formulas.xlsx");
//Defer the update of dirty cell state
wb.DeferUpdateDirtyState = true;
for (int i = 0; i < 1000; i++)
{
    wb.Worksheets[0].Range[i, 0].Value = i;
```

```

}
//Resume the update of dirty cell state
wb.DeferUpdateDirtyState = false;

```

### Limitation

When `Workbook.DeferUpdateDirtyState` is set to `True`, `GcExcel` does not update the dirty state of formula cells immediately. At this point the referred ranges for other features (such as chart etc.) won't be dirty, so their caches would not be updated. If you retrieve the state of such features, they may not be correct at that particular point of time.

## 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 <code>Table1</code> , or change it to use a custom name. Example: <code>DeptSales</code> 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 <code>[]</code> square brackets when they are written in the table formula. Example: <code>[SalesAmount]</code> and <code>[ComAmt]</code>
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#
<pre>// Define Data worksheet.Range["A1:E3"].Value = new object[,]</pre>



```

{
    {"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** of the **IRange interface** 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** of the **IRange interface**. In order to get an entire array if specified range is part of an array, you can use **CurrentArray property**.

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();

```

## Precedents and Dependents

Sometimes, in worksheets containing lots of formulas, it becomes difficult to identify which cell values or ranges are taken into consideration while doing calculations or how the result is calculated. Also, which cells are impacted if a cell value is modified. Hence, comes the need for precedent and dependent cells or ranges. GcExcel library provides **GetPrecedents** and **GetDependents** methods in the **IRange** interface, which help in identifying the precedent and dependent cells or ranges in excel worksheets.

- **Precedents:** Cells or ranges which are referred to, by the formulas in other cells
- **Dependents:** Cells or ranges which contain formulas that refer to other cells

For example, the value in cell A1 = 10, A2 = 20 and B1 = Sum (A1+A2), then A1 and A2 are the precedent cells of B1 which are used for calculating the value of B1. Also, B1 is the dependent cell for A1 and A2 whose value is calculated based on values of cell A1 and A2.

### Precedents

Refer to the following example code to get the Precedent ranges in a worksheet.

```

C#
public void GetPrecedents()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Set Formula in Cell E2
    worksheet.Range["E2"].Formula = "=sum(A1:A2, B4,C1:C3)";
    // Set Value of Cells
    worksheet.Range["A1"].Value = 1;
    worksheet.Range["A2"].Value = 2;
    worksheet.Range["B4"].Value = 3;
    worksheet.Range["C1"].Value = 4;
}

```

```

worksheet.Range["C2"].Value = 5;
worksheet.Range["C3"].Value = 6;

// Get Precedent cells of Range E2
foreach (var item in worksheet.Range["E2"].GetPrecedents())
{
    item.Interior.Color = Color.Pink;
}

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

```

The below image shows the precedent ranges (highlighted in pink).

	E2		fx =SUM(A1:A2, B4,C1:C3)			
	A	B	C	D	E	F
1	1		4			
2	2		5		21	
3			6			
4		3				
5						
6						

## Dependents

Refer to the following example code to get dependent ranges in a worksheet.

C#

```

public void GetDependents()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Set Value of Cell A1
    worksheet.Range["A1"].Value = 100;
    // Set Formula in Cell C1
    worksheet.Range["C1"].Formula = "=$A$1";
    // Set Formula in Range E1:E5
    worksheet.Range["E1:E5"].Formula = "=$A$1";

    // Get Dependent cells of Range A1
    foreach (var item in worksheet.Range["A1"].GetDependents())

```

```

{
    item.Interior.Color = Color.LightGreen;
}

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

```

The below image shows the dependent ranges (highlighted in green).

	A	B	C	D	E	F
1	100		100		100	
2					100	
3					100	
4					100	
5					100	
6						

### All Precedents

Often multiple precedent ranges are used to calculate cell formula. Refer to the following example code to get all the precedent ranges in a worksheet.

```

C#
public void GetAllPrecedents()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Set Formula in Cell E2
    worksheet.Range["E2"].Formula = "=sum(C1:C2)";
    // Set Formula in Cell C1
    worksheet.Range["C1"].Formula = "=B1";
    // Set Formula in Cell B1
    worksheet.Range["B1"].Formula = "=sum(A1:A2)";
    // Set Value of Cells
    worksheet.Range["A1"].Value = 1;
    worksheet.Range["A2"].Value = 2;
    worksheet.Range["C2"].Value = 3;

    // Get Precedent cells of Range E2
    List<IRange> list = new List<IRange>();
    foreach (var item in worksheet.Range["E2"].GetPrecedents())

```

```

{
    list.Add(item);
}

while (list.Count > 0)
{
    var temp = list;
    list = new List<IRange>();
    foreach (var item in temp)
    {
        for (int i = 0; i < item.RowCount; i++)
        {
            for (int j = 0; j < item.ColumnCount; j++)
            {
                var dependents = item.Cells[i, j].GetPrecedents();
                if (dependents.Count == 0)
                {
                    item.Cells[i, j].Interior.Color = Color.SkyBlue;
                }
                else
                {
                    item.Cells[i, j].Interior.Color = Color.LightGreen;
                    list.AddRange(dependents);
                }
            }
        }
    }
}

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

```

The below image shows all the precedent ranges (highlighted in blue and green).

	E2		fx		=SUM(C1:C2)	
	A	B	C	D	E	F
1	1	3	3			
2	2		3		6	
3						
4						

## Iterative Calculation

Iterative calculation is performed to repeatedly calculate a function until a specific numeric condition is met. GcExcel allows you to enable and perform iterative calculations by using **EnableIterativeCalculation** property of

**IFormulaOptions** interface. Additionally, you can also set or retrieve the following:

- Maximum number of iterations by using **MaximumIterations** property
- Maximum difference between values of iterative formulas by using **MaximumChange** property

For example, if **MaximumIterations** is set to 10 and **MaximumChange** is set to 0.001, GcExcel will stop calculating either after 10 calculations, or when there is a difference of less than 0.001 between the results.

Refer to the following example code to perform iterative calculation in a worksheet by performing 10 iterations.

C#

```
//create a new workbook
Workbook workbook = new Workbook();

//enable iterative calculation
workbook.Options.Formulas.EnableIterativeCalculation = true;
workbook.Options.Formulas.MaximumIterations = 10;
var worksheet = workbook.Worksheets[0];
worksheet.Range["A1"].Formula = "=B1 + 1";
worksheet.Range["B1"].Formula = "=A1 + 1";

Console.WriteLine("A1:" + worksheet.Range["A1"].Value.ToString());
Console.WriteLine("B1:" + worksheet.Range["B1"].Value.ToString());

workbook.Save("IterativeCalculation.xlsx");
```

## Cross Workbook Formula

Cross workbook formulas allow you to refer the data in other workbooks by creating formulas referring to external workbooks. For example, if there are 5 workbooks for different subjects, you can add the marks of all five subjects in a worksheet by using cross workbook formulas.

GcExcel provides **GetExcelLinkSources** property which can be used to get the names of linked excel workbooks and **UpdateExcelLinks** method to update the caches of excel links.

Refer to the following example code to use cross workbook formula and update the excel links.

C#

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

workbook.Worksheets[0].Range["B1"].Formula = @"='D:[A.xlsx]Sheet1'!A1";
// create a new workbook as the intance of external workbook.
var workbook2 = new GrapeCity.Documents.Excel.Workbook();
workbook2.Worksheets[0].Range["A1"].Value = "Hello, World!";
```

```
workbook2.Worksheets[0].Range["A2"].Value = "Hello";  
// update the caches of external workbook data.  
foreach (var item in workbook.GetExcelLinkSources())  
{  
    workbook.UpdateExcelLink(item, workbook2);  
}  
  
//save to an excel file  
workbook.Save("crossworkbookformula.xlsx");
```

## Custom Functions

GcExcel 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** class and declare the custom function in the new class along with the function name, return type, and parameters.

You can also use custom objects in custom functions as demonstrated by the last example in this topic. If one parameter of overloaded **Parameter** method is set to `FunctionValueType.Object` and `acceptCustomObjects` is set to `True`, custom objects can be used. Similarly, if the return type is `FunctionValueType.Object`, the formula can return custom objects.

### Using Code

- Step 1: Define a custom function
- Step 2: Register the custom function in your worksheet using the **AddCustomFunction()** 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

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++)
            {
                if (range.Cells[i, j].DisplayFormat.Interior.Color ==
System.Drawing.Color.Red)
                {
                    yield return range.Cells[i, j].Value;
                }
            }
        }
    }
}
}
}

```

## C#

```

// Step2: Register the custom function using AddCustomFunction() method
var workbook = new GrapeCity.Documents.Excel.Workbook();
GrapeCity.Documents.Excel.Workbook.AddCustomFunction(new MyConditionalSumFunctionX());
IWorksheet worksheet = workbook.Worksheets[0];

// Step3- Implement the Custom Function
worksheet.Range["A1:A10"].Value = new object[,] { { 1 }, { 2 }, { 3 }, { 4 }, { 5 }, { 6 }, { 7 }, { 8 }, { 9 }, { 10 } };
IFormatCondition cellValueRule =
worksheet.Range["A1:A10"].FormatConditions.Add(FormatConditionType.CellValue,
FormatConditionOperator.Greater, 5) as IFormatCondition;
cellValueRule.Interior.Color = System.Drawing.Color.Red;

```

```
// Sum cells value which display format interior color are red.
worksheet.Range["C1"].Formula = "=MyConditionalSum(A1:A10)";
// Range["C1"]'s value is 40.
var result = worksheet.Range["C1"].Value;
// Display result in cell E2
worksheet.Range["E2"].Value = result;
```

### Example 2: Custom Concatenation Function

Refer to the following example code to create and use custom concatenation function in your spreadsheet.

C#

```
// Step 1- Defining custom function: MyConcatenate
// Creating a new class MyConcatenateFunctionX by inheriting the CustomFunction class
public class MyConcatenateFunctionX : CustomFunction
{
    public MyConcatenateFunctionX() : base("MyConcatenate", FunctionValueType.Text,
CreateParameters())
    {
    }
    private static Parameter[] CreateParameters()
    {
        Parameter[] parameters = new Parameter[254];
        for (int i = 0; i < 254; i++)
        {
            parameters[i] = new Parameter(FunctionValueType.Variant);
        }
        return parameters;
    }
    public override object Evaluate(object[] arguments, ICalcContext context)
    {
        StringBuilder sb = new StringBuilder();
        string result = string.Empty;
        foreach (var argument in arguments)
        {
            if (argument is CalcError)
            {
                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;

```

### Example 5: Greatest Common Division Function using Custom Objects

Refer to the following example code to create and use BigInteger function to calculate greatest common division.

C#

```

// Step 1.1- Defining custom function: BigIntegerMultiplyFunction
internal class BigIntegerMultiplyFunction : CustomFunction
{
    public BigIntegerMultiplyFunction() : base("BIG.INTEGER.MULT", FunctionValueType.Object,
new[]
    {
        new Parameter(FunctionValueType.Text),
        new Parameter(FunctionValueType.Text)
    })
    {
    }
}

public override object Evaluate(object[] arguments, ICalcContext context)
{
    if (!(arguments[0] is string) || !(arguments[1] is string))
    {
        return CalcError.Value;
    }
}

```

```
}
var leftNumber = (string)arguments[0];
var rightNumber = (string)arguments[1];
try
{
    return BigInteger.Parse(leftNumber) * BigInteger.Parse(rightNumber);
}
catch (FormatException)
{
    return CalcError.Value;
}
catch (ArgumentException)
{
    return CalcError.Value;
}
}

}
// Step 1.2- Defining custom function: BigIntegerPowFunction
internal class BigIntegerPowFunction : CustomFunction
{
public BigIntegerPowFunction() : base("BIG.INTEGER.POW", FunctionValueType.Object, new[]
{
    new Parameter(FunctionValueType.Text),
    new Parameter(FunctionValueType.Number)
    })
{
}

public override object Evaluate(object[] arguments, ICalcContext context)
{
    if (!(arguments[0] is string) || !(arguments[1] is double))
    {
        return CalcError.Value;
    }
    var number = (string)arguments[0];
    var exp = (double)arguments[1];
    if (exp > int.MaxValue || exp < int.MinValue)
    {
        return CalcError.Value;
    }
    var iExp = Convert.ToInt32(exp);
    try
    {
        return BigInteger.Pow(BigInteger.Parse(number), iExp);
    }
    catch (FormatException)
    {

```

```

        return CalcError.Value;
    }
    catch (ArgumentException)
    {
        return CalcError.Value;
    }
}
}
// Step 1.3- Defining custom function: GreatestCommonDivisionFunction
internal class GreatestCommonDivisionFunction : CustomFunction
{
    public GreatestCommonDivisionFunction() : base("BIG.INTEGER.GCD",
FunctionValueType.Object, new[] {
        new Parameter(FunctionValueType.Object, false, true),
        new Parameter(FunctionValueType.Object, false, true)
    })
    {
    }

    public override object Evaluate(object[] arguments, ICalcContext context)
    {
        if (!(arguments[0] is BigInteger) || !(arguments[1] is BigInteger))
        {
            return CalcError.Value;
        }
        var leftNumber = (BigInteger)arguments[0];
        var rightNumber = (BigInteger)arguments[1];
        try
        {
            return BigInteger.GreatestCommonDivisor(leftNumber, rightNumber);
        }
        catch (ArgumentException)
        {
            return CalcError.Value;
        }
    }
}

```

## C#

```

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

try
{
    // Step2.1: Register the custom function using AddCustomFunction() method
    GrapeCity.Documents.Excel.Workbook.AddCustomFunction(new BigIntegerPowFunction());
}

```

```
catch (Exception)
{
    // Function was added
} // End Try
try
{
    // Step2.2: Register the custom function using AddCustomFunction() method
    GrapeCity.Documents.Excel.Workbook.AddCustomFunction(new
BigIntegerMultiplyFunction());
}
catch (Exception)
{
    // Function was added
} // End Try
try
{
    // Step2.3: Register the custom function using AddCustomFunction() method
    GrapeCity.Documents.Excel.Workbook.AddCustomFunction(new
GreatestCommonDivisionFunction());
}
catch (Exception)
{
    // Function was added
} // End Try

// Use BigInteger to calculate results
IWorksheet worksheet = workbook.ActiveSheet;
// Step3- Implement the Custom Function
worksheet.Range["A1"].Value = "154382190 ^ 3 = ";
worksheet.Range["A2"].Value = "1643590 * 166935 = ";
worksheet.Range["A3"].Value = "Greatest common division = ";
worksheet.Range["B1"].Formula = "=BIG.INTEGER.POW(\"154382190\", 3)";
worksheet.Range["B2"].Formula = "=BIG.INTEGER.MULT(\"1643590\", \"166935\")";
worksheet.Range["B3"].Formula = "=BIG.INTEGER.GCD(B1,B2) ";

// Arrange
worksheet.Columns[0].AutoFit();
worksheet.Columns[1].ColumnWidth = worksheet.Range["B1"].Text.Length + 1;

//save to a pdf file
workbook.Save("customobjectincustomfunction.pdf");
```

## Shapes and Pictures

GcExcel .NET allows you to embed drawing objects like 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** and the **IShapes interface**.



With GcExcel library, you can create different shape types such as Connector, Shape and Picture.

## Connector

A connector is used when you need to connect or disconnect two general shapes. In GcExcel, you can use the **BeginConnect method**, **EndConnect method**, **BeginDisconnect method** and **EndDisconnect method** of the **IConnectorFormat interface** 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.

A name can also be assigned to a shape, be it a chart, picture, connector or any autoshape, by using different methods provided in **IShapes** interface. By assigning a name to a shape, it be directly accessed and its properties can be modified rather than traversing through the list of all shapes.

Refer to the below example code to assign a name to an autoshape.

```
C#  
  
Workbook workbook = new Workbook();  
// Fetch default worksheet  
IWorksheet worksheet = workbook.Worksheets[0];  
  
//Create shape with custom name  
IShape shape = worksheet.Shapes.AddShape("Balloon", AutoShapeType.Balloon, 50, 50, 100,  
200);  
  
//save to an excel file  
workbook.Save("BalloonShape.xlsx");
```

Refer to the below example code to assign a name to a chart.

C#

```
//create a new workbook
Workbook workbook = new Workbook();

IWorksheet worksheet = workbook.Worksheets[0];

//set chart name
IShape shape = worksheet.Shapes.AddChart("Area chart with custom name", ChartType.Area,
250, 20, 360, 230);
worksheet.Range["A1:C13"].Value = new object[,] {
{ null, "Blue Series", "Orange Series" },
{ "Jan", 0, 59.1883603948205 },
{ "Feb", 44.6420211591501, 52.2280901938606 },
{ "Mar", 45.2174930051225, 49.8093056416248 },
{ "Apr", 62, 37.3065749226828 },
{ "May", 53, 34.4312192530766 },
{ "Jun", 31.8933622049831, 69.7834561753736 },
{ "Jul", 41.7930895085093, 63.9418103906982 },
{ "Aug", 73, 57.4049534494926 },
{ "Sep", 49.8773891668518, 33 },
{ "Oct", 50, 74 },
{ "Nov", 54.7658428630216, 22.9587876597096 },
{ "Dec", 32, 54 },
};

//Get chart by name
IShape areaChart = worksheet.Shapes["Area chart with custom name"];
areaChart.Chart.SeriesCollection.Add(worksheet.Range["A1:C13"], RowCol.Columns);
areaChart.Chart.ChartTitle.Text = "Area Chart";

//save to an excel file
workbook.Save("ChartName.xlsx");
```

## Picture

You can insert pictures on cells of a spreadsheet by using the **AddPicture** method of the **IShapes** interface. The **IPictureFormat** interface 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;
```

Refer to the below example code to assign a name to a picture.

```
C#  
  
Workbook workbook = new Workbook();  
// Fetch default worksheet  
IWorksheet worksheet = workbook.Worksheets[0];  
  
//add picture with custom name  
IShape shape = worksheet.Shapes.AddPicture("Custom Name to Image", "image.png", 10, 10,  
250, 150);  
  
//save to an excel file  
workbook.Save("PictureName.xlsx");
```

Working with shapes and pictures in the GcExcel library involves the following tasks:

[Customize Shape Format and Shape Text](#)

[Hyperlink on Shape](#)

[Group or Ungroup Shapes](#)


[Shape Adjustment](#)

[Background Image](#)

[Size and Position of Image](#)

[Image Transparency](#)

[Control Position of Overlapping Shapes](#)

 **Note:** 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](#).

## Customize Shape Format and Shape Text

GcExcel not only allows you to add shapes and picture, the library also lets you customize shape formats and shape texts. A user can enhance the look of a shape in the Excel file by changing fill color, formatting three-dimensional orientation or adding lines around the shape.

Using GcExcel, a user can customize both the shape format and shape text.

### 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**, configuring the shape's line using the properties and methods of the **ILineFormat interface** and applying 3D formatting to the shape using the properties and methods of

the **IThreeDFormat** interface.

### Solid Fill

To format the shape with Solid fill, first you need to use the **Solid method** of the **IFillFormat** interface to specify the fill format and then set the **Color property** and **Transparency property** 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

Gradient fill is a graphical effect which provides the 3D color look as one color blends into another. In gradient fill, you first need to set the shape fill to the gradient fill using the **OneColorGradient method**, **TwoColorGradient method** or **PresetGradient method** 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**, **RotateWithObject property** and **GradientAngle property** of the **IFillFormat** interface.

Four types of gradient fills, namely line, radial, rectangular and path are supported by GcExcel. By default, the 'Line' gradient fill is applied.

Refer to the following example code to fill the shape with gradient fill using **PresetGradient** method.

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;
```

Refer to the following example code to fill the shape with gradient fill using **TwoColorGradient** method.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Add a shape
IShape rectangle = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 20, 20, 300, 100);

// Init a two color gradient fill
rectangle.Fill.TwoColorGradient(GradientStyle.Horizontal, 1);
```

```
//save to an excel file
workbook.Save("LineGradient.xlsx");
```

To set the radial, rectangular or path gradient fill, you also need to set the **GradientPathType** along with using the **TwoColorGradient** method.

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

```
C#

// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Add a shape
IShape rectangle = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 20, 20, 300, 100);

// Init a two color gradient fill
rectangle.Fill.TwoColorGradient(GradientStyle.FromCenter, 1);

// Set gradient path type
rectangle.Fill.GradientPathType = PathShapeType.Radial;

//save to an excel file
workbook.Save("RadialGradient.xlsx");
```

### Pattern Fill

In pattern fill, you first need to set the shape fill to pattern fill using the **Patterned method** of the **IFillFormat** interface. Afterwards, you can set the background color and the pattern color using **Color property** and **PatternColor property** 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** 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**, or **UserTextured method** of the **IFillFormat** interface. Further, you can also use the **TextureAlignment property**, **TextureHorizontalScale property**, **TextureOffsetX property**, **TextureOffsetY property** and **TextureVerticalScale property** 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** 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;
```

You can also set the alignment and position of text on shape by using the **HorizontalAnchor** and **VerticalAnchor** properties of **ITextFrame** interface. These properties configure the horizontal and vertical alignment of text on shape.

Further, the **HorizontalAnchor** and **VerticalAnchor** enumerations specify the position of text in the text frame. The text can be placed horizontally at the center or vertically at top, middle or bottom.

These different alignments and positions of text on shape can also be exported to PDF document.

Refer to the following example code to configure the alignment and position of text on a shape.

C#

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

//Add a shape.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 10, 10, 300, 300);

//Add two paragraphs for the shape.
shape.TextFrame.TextRange.Paragraphs.Add("GrapeCity Documents for Excel");
shape.TextFrame.TextRange.Paragraphs.Add("Middle Centered");

//Centers text vertically.
shape.TextFrame.VerticalAnchor = VerticalAnchor.AnchorMiddle;
//Centers text horizontally.
shape.TextFrame.HorizontalAnchor = HorizontalAnchor.Center;

workbook.Save("Alignment.xlsx");
workbook.Save("Alignment.pdf");
```



**Note:** The right alignment for text on shape is not supported in GcExcel as it is not supported in Excel.

## Hyperlink on Shape

In GcExcel, hyperlinks can be added to various shape types like basic shapes, charts, connectors, pictures and group shapes. It allows users to quickly navigate to related information on a webpage, external file, specific range in the same workbook, or email address by clicking on the shape.



**Note:** Hyperlink cannot be added to **Comment** and **Slicer** shape types.

Hyperlinks can be configured using the following properties of the **IHyperlink** interface.

1. The **Address** and **SubAddress** properties of the **IHyperlink** interface can be used to configure the hyperlink references. The table shown below illustrates both the 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. The **EmailSubject** property can be used to set the text of hyperlink's email subject line.
3. The **ScreenTip** property can be used to set the tip text for the specified hyperlink.
4. The **TextToDisplay** property can be used to set the text to be displayed for the specified hyperlink.

## Add Hyperlink

A user can add hyperlink to a shape in a worksheet using the **Add** method of the **IHyperLinks** interface.

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

C#

```
//Add a hyperlink to external file

//Add a Shape
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Oval, 1, 1, 200, 100);
shape.TextFrame.TextRange.Paragraphs.Add("Link to Test.xlsx file");
//Add Hyperlink
worksheet.Hyperlinks.Add(shape, @"C:\Test.xlsx", null, "Link to Test.xlsx file",
"Test.xlsx");
//Save to an excel file
workbook.Save("ExternalHyperlink.xlsx");
```

C#

```
// Add a hyperlink to web page

//Add a Shape
IShape picture = worksheet.Shapes.AddPicture(@"Images\grapecity-logo.jpg", 1, 1, 100,
100);
//Add Hyperlink
worksheet.Hyperlinks.Add(picture, "https://www.grapecity.com/", null, "Click to Open",
"GrapeCity");
//Save to an excel file
workbook.Save("ShapeHyperlink.xlsx");
```

C#

```
// Create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();
// Fetch a worksheet
IWorksheet worksheet = workbook.Worksheets[0];
// Add another worksheet
IWorksheet worksheet1 = workbook.Worksheets.Add();
#region HyperlinkRange
//Add a hyperlink to a range in Sheet2
//Add a Shape in Sheet1
```

```
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Oval, 1, 1, 200, 100);
shape.TextFrame.TextRange.Paragraphs.Add("Go To sheet2 J3:K4");
//Add Hyperlink in Sheet1 which navigates to range J3:K4 in Sheet2
worksheet.Hyperlinks.Add(shape, null, "Sheet2!$J$3:$K$4", "Go To sheet2 J3:K4");
//Save to an excel file
workbook.Save("RangeHyperlink.xlsx");
```

C#

```
//Add a hyperlink to email address.

//Add a Shape
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Oval, 1, 1, 200, 100);
shape.TextFrame.TextRange.Paragraphs.Add("Send Feedback");
//Add Hyperlink
worksheet.Hyperlinks.Add(shape, "mailto:web_feedback@grapecity.com", null, "Send your
valuable feedback.", "Feedback");
//Save to an excel file
workbook.Save("MailTo.xlsx");
```

## Delete Hyperlink

The hyperlink on the shape can be removed using the **Delete** method of the **IHyperlink** interface.

Refer to the following example code to delete hyperlink.

C#

```
//Delete hyperlink.

//Add Shape
IShape shapeOval = worksheet.Shapes.AddShape(AutoShapeType.Oval, 1, 1, 200, 100);

// Create Hyperlinks
IHyperlink hyperlink1 = worksheet.Hyperlinks.Add(shapeOval,
"https://www.grapecity.com/", null, "Click to Open", "GrapeCity");

//Delete hyperlink1.
hyperlink1.Delete();

//Save to an excel file
workbook.Save("DeleteHyperlink.xlsx");
```

## Group or Ungroup Shapes

GcExcel allows you to group or ungroup shapes in a worksheet. Shapes can be grouped together when there is a need to perform certain action on the bunch of shapes together. For example: adding similar style to shapes, aligning, rotating, copying or pasting the grouped shapes together. It does not only saves a considerable amount of time and efforts but also helps in ensuring that the desired consistency is maintained in all the shapes.

## Group Shapes

Several shapes can be grouped together using the **Group** method of the **IShapeRange** interface. The **IShapeRange** interface represents the range of the shapes which needs to be grouped together. The grouped shapes behave as a single shape.

Refer to the following example code to group shapes.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];
//Creating shapes collection for activeSheet
IShapes shapes = workbook.ActiveSheet.Shapes;

// Adding Shapes to shapes collection
IShape ShapeBegin = shapes.AddShape(AutoShapeType.Wave, 10, 10, 100, 100);
IShape EndBegin = shapes.AddShape(AutoShapeType.RoundedRectangle, 200, 200, 100, 100);
// Adding Connector Shape to shapes collection
IShape ConnectorShape = shapes.AddConnector(ConnectorType.Straight, 10, 10, 101, 101);

//Connecting ShapeBegin & EndBegin shapes by connector shape
ConnectorShape.ConnectorFormat.BeginConnect(ShapeBegin, 3);
ConnectorShape.ConnectorFormat.EndConnect(EndBegin, 0);

//Adding IsoscelesTriangle shape to shapes collection
shapes.AddShape(AutoShapeType.IsoscelesTriangle, 370.8, 50.8, 81.6, 102.0);

//Creating shpRange collection to group certain shapes as given in array
IShapeRange shpRange = shapes.Range[new string[3] { shapes[0].Name, shapes[1].Name,
shapes[2].Name }];

// Grouping Shapes
IShape grouped = shpRange.Group();
// Setting Style for Grouped shape together
grouped.Line.Color.RGB = System.Drawing.Color.DarkOrange;
grouped.Fill.Color.RGB = System.Drawing.Color.LightGreen;
Console.WriteLine("Group Name is: " + grouped.Name);

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

## Ungroup Shapes

A group of shapes in a specified range can be ungrouped using the **Ungroup** method of the **IShape** interface.

Refer to the following example code to ungroup shapes.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Open workbook
workbook.Open(@"9-GroupedShapes.xlsx");
IShapes shapes = workbook.Worksheets[0].Shapes;

// Ungroup Shapes
for (int i = 0; i < shapes.Count; i++)
{
    if (shapes[i].Type == ShapeType.Group) // Or, if (shapes[i].Name == "Group 1")
        shapes[i].Ungroup();
}

// Or, we can just pass GroupName to Ungroup it
// shapes["Group 1"].Ungroup();

// Saving workbook to Xlsx
workbook.Save(@"10-UnGroupedShapes.xlsx", SaveFileFormat.Xlsx);
```

## Shape Adjustment

Apart from changing the size of a shape in GcExcel, you can also change the geometry of a shape and modify its appearance. This can be achieved by setting the adjustment values of shapes, such as AutoShapes or Connectors. It allows you to have more control over the shapes in order to create efficient flowcharts, dashboards and reports.

GcExcel provides the **Adjustments** property in the **IShape** interface to get a collection of adjustment values for the specified AutoShape or Connector.

The valid ranges of adjustment values for different adjustment types are described below:

Adjustment type	Valid values
Linear (horizontal or vertical)	<p>Value 0.0 represents the left or top edge of the shape.</p> <p>Value 1.0 represents the right or bottom edge of the shape.</p> <p>For shapes such as connectors and callouts, the values 0.0 and 1.0 correspond to the rectangle defined by the starting and ending points of the connector or callout line.</p> <p>Values lesser than 0.0 and greater than 1.0 are also valid.</p> <p>The valid values for the adjustment correspond to the valid adjustments that can be made to shapes in Excel by extending the adjustment points.</p> <p>For example, if you can only pull an adjustment point half way across the shape in Excel, the maximum value for the corresponding adjustment will be 0.5.</p>
Radial	<p>Value 1.0 represents the shape width. Hence, the maximum value for radial adjustment is 0.5, which is half way across the shape.</p>

Angle

Value is expressed in degrees. If you specify the value outside the range of 180 degree, it will be normalized to be within that range.

In most cases, if a value exceeds the valid range, it is normalized to the closest valid value.

### Using Code

Refer to the following example code to adjust the dimensions of a shape in Excel:

C#

```
public void AdjustmentPointForShape()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];
    // Add a right arrow callout
    IShape shape = worksheet.Shapes.AddShape(AutoShapeType.RightArrowCallout, 20, 20,
    200, 100);

    IAdjustments adjustments = shape.Adjustments;

    // Get the count of adjustment values for shape
    int c = adjustments.Count;
    Console.WriteLine("Count of Adjustment Values: " + c.ToString());

    // Set adjustment values for shapes
    adjustments[0] = 0.5; // arrow neck width
    adjustments[1] = 0.4; // arrow head width
    adjustments[2] = 0.5; // arrow head height
    adjustments[3] = 0.6; // text box width

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

## Background Image

GcExcel allows you to set background image in a worksheet using the **BackgroundPicture** property of the **IWorksheet** interface. The background image can be saved to Excel and is rendered multiple times, side by side, to cover the whole area of the worksheet.

### Using Code

Refer to the following example code to save sheet background image in Excel.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A1"].Value = "GrapeCity Documents for Excel";
worksheet.Range["A1"].Font.Size = 25;

using (FileStream pictureStream = File.Open(@"image.png", FileMode.Open,
FileAccess.Read))

{
    MemoryStream pictureMemoryStream = new MemoryStream();

    pictureStream.CopyTo(pictureMemoryStream);
    byte[] picturebytes = pictureMemoryStream.ToArray();

    //Add background image of the worksheet
    worksheet.BackgroundImage = picturebytes;
}

workbook.Save(@"SetBackgroundImage.xlsx", SaveFileFormat.Xlsx);
```

The background image can also be included while exporting the worksheet to PDF documents. For more information, refer to [Support Sheet Background Image](#) in this documentation.

## Size and Position of Image

Sometimes, it is required to render an image in a worksheet at a specific position. In such cases, it becomes very difficult to determine the position or size of the image by traversing through the cells of the worksheet.

GcExcel allows you to know the size and absolute position of an image by using **GetRangeBoundary** method of type **Rectangle** in the **CellInfo** class. The method returns the location and size of the image (in pixels).

### Using Code

Refer to the following example code to get the location and size of an image by adding it at a specified range in a worksheet.

C#

```
IWorkbook workbook = new Workbook();
IWorksheet worksheet = workbook.ActiveSheet;
IRange range = worksheet.Range["D4:H8"];

// Get the absolute location and size of the Range["D4:H8"] in the worksheet.
Rectangle rect = CellInfo.GetRangeBoundary(range);
```

```
Console.WriteLine("The location and size of the image:" + rect);  
// Add the image to the Range["D4:H8"].  
worksheet.Shapes.AddPictureInPixel("image.png", rect.X, rect.Y, rect.Width,  
rect.Height);  
workbook.Save("GetRangePosition.xlsx");
```

## Image Transparency

GcExcel supports controlling the transparency of an image by providing **Transparency** property in **IPictureFormat** interface. The value of **Transparency** can vary between 0.0 (opaque) to 1.0 (clear).

### Using Code

Refer to the following example code to set the transparency of an image.

```
C#  
  
//create a new workbook  
var workbook = new GrapeCity.Documents.Excel.Workbook();  
  
//use sheet index to get worksheet  
IWorksheet worksheet = workbook.Worksheets[0];  
  
//add an image  
var picture = worksheet.Shapes.AddPicture("Image.png", 10, 10, 200, 100);  
  
//set image transparency as 60%  
picture.PictureFormat.Transparency = 0.6;  
  
//save to an excel file  
workbook.Save("imagetransparent.xlsx");
```

### Limitation


SpreadJS does not support image transparency, hence this info would be lost when using json I/O.

## Control Position of Overlapping Shapes

The order of overlapping shapes in a worksheet is decided by their z-order positions. GcExcel allows its users to set the z-order of shapes so that their positions can be controlled while creating flow charts or business diagrams etc.

The **ZOrder** method in GcExcel API can be used to move the specified shape in front of or behind the other shapes. It takes **ZOrderType** enum as a parameter to specify the position of a shape relative to the other shapes.

The **ZOrderPosition** property of the **IShape** interface can be used to retrieve the position of a specified shape in the z-order.

 **Note:** If the z-order of a shape is changed, the index of the shape in Worksheet.Shapes collection is also changed.

## Using Code

Refer to the below example code to add various shapes, change their z-order and get their positions in z-order in a worksheet.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

IShapes shapes = worksheet.Shapes;

//add shapes
IShape rectangle = shapes.AddShape(AutoShapeType.Rectangle, 20, 20, 100, 100);
rectangle.Fill.Color.RGB = System.Drawing.Color.Blue;

IShape oval = shapes.AddShape(AutoShapeType.Oval, 50, 50, 100, 100);
oval.Fill.Color.RGB = System.Drawing.Color.Green;

IShape pentagon = shapes.AddShape(AutoShapeType.Pentagon, 80, 80, 100, 100);
pentagon.Fill.Color.RGB = System.Drawing.Color.Red;

IShape triangle = shapes.AddShape(AutoShapeType.IsoscelesTriangle, 100, 100, 100, 100);
triangle.Fill.Color.RGB = System.Drawing.Color.Orange;

//set rectangle above oval
rectangle.ZOrder(ZOrderType.BringForward);

//get position of rectangle in z-order
Console.WriteLine("Z-Order rectangle: " + rectangle.ZOrderPosition);

//set triangle to bottom
triangle.ZOrder(ZOrderType.SendToBack);

//get position of triangle in z-order
Console.WriteLine("Z-Order triangle: " + triangle.ZOrderPosition);

//save to an excel file
workbook.Save("setshapezorder.xlsx");
```

## Styles

GcExcel .NET allows you to format the cells in a spreadsheet with a set of styles that can be utilized to format cell appearance in individual worksheets for enhanced clarity and increased readability. A cell style includes characteristics such as fill (solid fill, gradient fill, pattern fill), fonts, borders, name style, and display format.



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.	<b>IRange.NumberFormat</b>
Alignment	Horizontal and vertical alignment of cell content, indentation, text wrap, text rotation and text shrinking.	<b>IRange.AddIndent</b> <b>IRange.IndentLevel</b> <b>IRange.WrapText</b> <b>IRange.ShrinkToFit</b> <b>IRange.MergeCells</b> <b>IRange.ReadingOrder</b> <b>IRange.Orientation</b>
Font	IRange.Font(IFont)	<b>IRange.Font</b> (IFont)
Borders	Cell border line styles and colors.	<b>IRange.Borders</b> (IBorders)
Fill	Cell pattern fill or gradient fill.	<b>IRange.Interior</b> (IInterior)
Protection	Cell protection options (Locked and Hidden)	<b>IRange.Locked</b> <b>IRange.FormulaHidden</b>

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** of the **IRange interface**. 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** of the **IInterior interface**.

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);
```

### 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**, **ColorIndex**, **ThemeColor** and **TintAndShade** properties of the **IInterior interface**. In order to set the foreground color, you can use the **PatternColor**, **PatternColorIndex**, **PatternThemeColor**, **PatternTintAndShade** properties of the **IInterior interface**.



**Note:** For the **TintAndShade** 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** 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** of the **IInterior interface**.

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**.

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. 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** 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** 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** 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**, **VerticalAlignment property**, **AddIndent property** and **ReadingOrder property** 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** and **Locked property** 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** of **IStyleCollection interface**. 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**.

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** of the **IRange interface**. 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** 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** 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");
```

## Barcodes

GcExcel provides API to add barcodes in worksheets. These are very helpful in scanning information easily and quickly with utmost precision. They also facilitate users to take informed business decisions and improve data analysis.

The following types of barcodes are supported in GcExcel:

- [QRCode](#)
- [EAN-13](#)
- [EAN-8](#)
- [Codabar](#)
- [Code39](#)
- [Code93](#)
- [Code128](#)
- [GS1-128](#)
- [Code49](#)
- [PDF417](#)
- [Data Matrix](#)

GcExcel also supports SpreadJS JSON I/O of barcodes. For more information, refer to [Import and Export SpreadJS Files](#). Similarly, Barcodes can be exported to PDF documents. For more information about PDF exporting support, refer to [Export Barcodes](#).


















**Note:** When saved to xlsx file, the barcode is displayed as error "#NAME", as it is not supported in Excel.

## QRCode

QRCode is a two dimensional barcode representing symbology that enables effective handling of numeric, alphanumeric

and byte data. This barcode can encode up to 7,366 characters.


The below image displays QRCode barcode in a PDF document.

	QR Code						
	Server	Data	Default	Change errorCorrectionLevel	Change model	Change version	Change mask
	Police	911					
	Travel Info Call 511	511					
		www.grapecity.com					

### Formula definition

You can set QRCode in a worksheet using the following formula:

=BC\_QRCODE(value, color, backgroundColor, errorCorrectionLevel, model, version, mask, connection, connectionNo, charCode, charset, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

 **Note:** The 'value' parameter is mandatory and the remaining ones are optional. This holds true for all the barcodes that support 'value' parameter in GcExcel.

### Parameter

Name	Description
value	A string that represents encode on the symbol of QRCode.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
errorCorrectionLevel	A string that represents the error correction level of QRCode. It has 'L M Q H' four error correction levels. The default value is 'L'.
model	A value that represents the model of QRCode. It has 1 and 2 models. The default value is 2.
version	Vesion range is 1-14 for model1 and model 2. It has 'auto 1-14 1-40' values. The default value is 'auto'.
mask	A value that represents mask pattern for QRCode. It has 'auto and 0-7' eight mask pattern.
connection	A value that represents whether the symbol is part of a structured append message. The default value is false.
connectionNo	Specifies which block the symbol is in the structured append message. It has '0-15' values. The default value is '0'.



charCode	A value that represents the collection of characters of QRCode.
charset	A value that represents which charset to use. It has 'UTF-8 and Shift-JIS'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

### Using Code

This example code sets a QRCode in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:K"].ColumnWidth = 15;
worksheet.Range["4:6"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 2;
worksheet.Range["B2"].Value = "QR Code";
worksheet.Range["B2:K2"].Merge(true);
worksheet.Range["I3:J3"].Merge(true);
worksheet.Range["B3:H3"].Value = new object[,] {
    {"Server", "Data", "Default", "Change errorCorrectionLevel", "Change model", "Change version", "Change mask"}
};
worksheet.Range["I3"].Value = "Change connection and connectionNo";
worksheet.Range["K3:K5"].Value = new object[,] {
    {
        {"Explain" },
        {"No QR Code generated, barcode data is too short to create connection symbol."},
        {"No QR Code generated, barcode data is too short to create connection symbol."}
    };
};
worksheet.PageSetup.PrintTitleColumns = "$A:$C";
worksheet.PageSetup.Orientation = PageOrientation.Landscape;
worksheet.PageSetup.PrintGridlines = true;
worksheet.Range["K4:K5"].Font.Color = Color.Red;
worksheet.Range["K4:K5"].WrapText = true;
worksheet.Range["B4:C6"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C6"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:K3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:K3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C6"].Value = new object[,] {
    {
```

```

    {"Police", "911"},
    {"Travel Info Call 511", "511"},
    { "", "www.grapecity.com"},
};

// Set formula
for (var i = 4; i < 7; i++)
{
    worksheet.Range["D" + i].Formula = "=BC_QRCODE" + "(C" + i + ")";
    worksheet.Range["E" + i].Formula = "=BC_QRCODE" + "(C" + i + ",,,,\"H\")";
    worksheet.Range["F" + i].Formula = "=BC_QRCODE" + "(C" + i + ",,,,1)";
    worksheet.Range["G" + i].Formula = "=BC_QRCODE" + "(C" + i + ",,,,8)";
    worksheet.Range["H" + i].Formula = "=BC_QRCODE" + "(C" + i + ",,,,3)";
    worksheet.Range["I" + i].Formula = "=BC_QRCODE" + "(C" + i + ",,,,,,\"true\",0)";
    worksheet.Range["J" + i].Formula = "=BC_QRCODE" + "(C" + i + ",,,,,,\"true\",1)";
}


// Save to a pdf file
workbook.Save("qrcode.pdf");

```

## EAN-13

EAN-13 barcode makes use of numeric characters (twelve numbers) and a check digit. This barcode accepts only twelve numbers as a string to calculate a check digit (Checksum) and adds it to the thirteenth position. The check digit is an additional digit that can be used to verify that the barcode has been scanned accurately. When the CheckSum property is set to True, the check digit is automatically added. This is mainly used in supermarkets and other retail businesses.


The below image displays EAN-13 barcode in a PDF document.

EAN-13						
Name	Number	Default	Change addOn	Change addOnLabelPosition	Explain	
Medicine	692031229621					
Pen	6945091701532					
value length is 13	8142486545683	#VALUE!	#VALUE!	#VALUE!	No EAN-13 generated, because the last digit is check-sum digit and it is invalid	

### Formula definition

You can set EAN-13 barcode in a worksheet using the following formula:

=BC\_EAN13(value, color, backgroundColor, showLabel, labelPosition, addOn, addOnLabelPosition, fontFamily, fontStyle, fontWeight, fontTextDecoration, fontTextAlign, fontSize, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

 **Note:** The 'labelPosition' parameter can only be set to top or bottom. This holds true for all the barcodes that support 'labelPosition' parameter in GcExcel.

### Parameter

Name	Description
value	Specifies that the value length must be 12 or 13.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
showLabel	Specifies whether to show label text when the barcode has label.
labelPosition	A value that represents the label position when the label is shown.
addOn	A string that represents the add text of EAN-13. Specifies that value length must be 2 or 5.
addOnLabelPosition	The position to add the text when text is shown.
fontFamily	A string that represents the label text fontFamily. The default value is 'sans-serif'.
fontStyle	A string that represents the label text fontStyle. The default value is 'normal'.
fontWeight	A string that represents the label text fontWeight. The default value is 'normal'.
fontTextDecoration	A string that represents the label text fontTextDecoration. The default value is 'none'.
fontTextAlign	A string that represents the label text fontTextAlign. The default value is 'center'.
fontSize	A string that represents the label text fontSize. The default value is '12px'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

### Using Code

This example code sets EAN13 in the worksheet.

```
C#  
  
// Create a new workbook  
var workbook = new GrapeCity.Documents.Excel.Workbook();  
  
// Set worksheet layout and data  
IWorksheet worksheet = workbook.Worksheets[0];  
worksheet.Range["B:C"].ColumnWidth = 15;  
worksheet.Range["D:G"].ColumnWidth = 20;  
worksheet.Range["4:7"].RowHeight = 60;
```

```

worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "EAN-13";
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:G3"].Value = new object[,]{
    {"Name", "Number", "Default", "Change addOn", "Change addOnLabelPosition", "Explain"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C6"].Value = new object[,]{
    {
        {"Medicine", "692031229621"},
        {"Pen", "6945091701532"},
        {"value length is 13", "8142486545683"}
    }
};
worksheet.Range["G6"].Value = "No EAN-13 generated, because the last digit is check-sum digit and it is invalid";
worksheet.Range["G6"].Font.Color = Color.Red;
worksheet.Range["B4:C6"].WrapText = true;
worksheet.Range["G6"].WrapText = true;
worksheet.PageSetup.Orientation = PageOrientation.Landscape;
worksheet.PageSetup.PrintGridlines = true;

// Set formula
for (var i = 4; i < 7; i++)
{
    worksheet.Range["D" + i].Formula = "=BC_EAN13" + "(C" + i + ")";
    worksheet.Range["E" + i].Formula = "=BC_EAN13" + "(C" + i + ",,22)";
    worksheet.Range["F" + i].Formula = "=BC_EAN13" + "(C" + i + ",,22,\"bottom\")";
}







// Save to a pdf file
workbook.Save("ean13.pdf");

```

## EAN-8

EAN-8 barcode is used on small packages where an EAN-13 barcode would be too large. Similar to EAN-13, EAN-8 uses only numeric characters and a check digit. This barcode accepts only seven numbers as a string to calculate a check digit (Checksum) and add it to the eighth position. The check digit is an additional digit that can be used to verify that the barcode has been scanned accurately. When the CheckSum property is set to True, the check digit is automatically added.

The below image displays EAN-8 barcode in a PDF document.

EAN-8						
	Name	Number	Default	Change showLabel	Change labelPosition	Explain
	Value length is 7	4137962				
	Value length is 8	81424863				
	value length is 8	81424865	#VALUE!	#VALUE!	#VALUE!	No EAN-8 generated, because the last digit is check-sum digit and it is invalid

### Formula definition

You can set EAN-8 barcode in a worksheet using the following formula:

=BC\_EAN8(value, color, backgroundColor, showLabel, labelPosition, fontFamily, fontStyle, fontWeight, fontTextDecoration, fontTextAlign, fontSize, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

### Parameter

Name	Description
value	Specifies that the value length must be 7 or 8.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
showLabel	Specifies whether to show label text when the barcode has label.
labelPosition	A value that represents the label position when the label is shown.
fontFamily	A string that represents the label text fontFamily. The default value is 'sans-serif'.
fontStyle	A string that represents the label text fontStyle. The default value is 'normal'.
fontWeight	A string that represents the label text fontWeight. The default value is 'normal'.
fontTextDecoration	A string that represents the label text fontTextDecoration. The default value is 'none'.
fontTextAlign	A string that represents the label text fontTextAlign. The default value is 'center'.
fontSize	A string that represents the label text fontSize. The default value is '12px'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

## Using Code

This example code sets EAN-8 in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:G"].ColumnWidth = 17;
worksheet.Range["4:7"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "EAN-8";
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:G3"].Value = new object[,] {
    {"Name", "Number", "Default", "Change showLabel", "Change labelPosition", "Explain"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C6"].Value = new object[,] {
    {
        {"Value length is 7", "4137962"},
        {"Value length is 8", "81424863"},
        {"value length is 8", "81424865"}
    }
};
worksheet.Range["G6"].Value = "No EAN-8 generated, because the last digit is check-sum digit and it is invalid";
worksheet.Range["G6"].Font.Color = Color.Red;
worksheet.Range["B4:C6"].WrapText = true;
worksheet.Range["G6"].WrapText = true;
worksheet.PageSetup.Orientation = PageOrientation.Landscape;
worksheet.PageSetup.PrintGridlines = true;










// Set formula
for (var i = 4; i < 7; i++)
{
    worksheet.Range["D" + i].Formula = "=BC_EAN8" + "(C" + i + ")";
    worksheet.Range["E" + i].Formula = "=BC_EAN8" + "(C" + i + ",,,0)";
    worksheet.Range["F" + i].Formula = "=BC_EAN8" + "(C" + i + ",,,,\"top\")";
}

// Save to a pdf file
workbook.Save("ean8.pdf");
```

## Codabar

Codabar is a barcode that uses alphanumeric characters including, A B C D + - : . / \$ and all numbers. This is widely used in sectors where serial numbers are required, such as blood Banks, door-to-door delivery service orders, and membership card management.

The below image displays Codabar barcode in a PDF document.

Codabar					
	Name	Number	Default	Change checkDigit	Change nwRatio
	Notebook	6935205311092			
	Paper	6922266446146			
	Value can contain letters and some symbol	A1234+-. \$A			

### Formula definition

You can set codabar in a worksheet using the following formula:

=BC\_CODABAR(value, color, backgroudColor, showLabel, labelPosition, checkDigit, nwRatio, fontFamily, fontStyle, fontWeight, fontTextDecoration, fontTextAlign, fontSize, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

### Parameter

Name	Description
value	A string that represents encode on the symbol of Codabar.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
showLabel	Specifies whether to show label text when the barcode has label.
labelPosition	A value that represents the label position when the label is shown.
checkDigit	Specifies whether the symbol needs a check digit. The default value is 'false'.
nwRatio	A value that represents the wide and narrow bar ratio. It has values 2 3. The default value is '3'.
fontFamily	A string that represents the label text fontFamily. The default value is 'sans-serif'.
fontStyle	A string that represents the label text fontStyle. The default value is 'normal'.
fontWeight	A string that represents the label text fontWeight. The default value is 'normal'.

fontTextDecoration	A string that represents the label text fontTextDecoration. The default value is 'none'.
fontTextAlign	A string that represents the label text fontTextAlign. The default value is 'center'.
fontSize	A string that represents the label text fontSize. The default value is '12px'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

### Using Code

This example code sets Codabar in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:F"].ColumnWidth = 20;
worksheet.Range["4:7"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "Codabar";
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:G3"].Value = new object[,] {
    {"Name", "Number", "Default", "Change checkDigit", "Change nwRatio"}
};

worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C6"].Value = new object[,] {
    {
        {"Notebook", "6935205311092"},
        {"Paper", "6922266446146"},
        {"Value can contain letters and some symbol", "A1234+-.$.A"}
    }
};
worksheet.Range["B4:C6"].WrapText = true;
worksheet.Range["G6"].WrapText = true;
worksheet.PageSetup.Orientation = PageOrientation.Landscape;
worksheet.PageSetup.PrintGridlines = true;

// Set formula
for (var i = 4; i < 7; i++)
```



```

{
    worksheet.Range["D" + i].Formula = "=BC_CODABAR" + "(C" + i + ")";
    worksheet.Range["E" + i].Formula = "=BC_CODABAR" + "(C" + i + ",,,,,,\"true\")";
    worksheet.Range["F" + i].Formula = "=BC_CODABAR" + "(C" + i + ",,,,,,\"2\")";
}

// Save to a pdf file
workbook.Save("codabar.pdf");

```







### Limitation

- The "checkDigit" parameter takes effect only when the 'value' parameter's length is 13 and the barcode's label text does not change.

## Code39

Code 39 is a linear barcode that uses a total of nine bars to represent each symbol which includes numeric characters, upper case characters and some special characters ("% ", "\*", "\$", "/", ".", "-", "+").

The below image displays Code39 barcode in a PDF document.

Code39					
	Name	Number	Default	Change labelWithStartAndStopCharacter	Change checkDigit
	Paper	6922266446146			
	Book	9787560044231			
	Value can contain some symbol	1234+-*	#VALUE!	#VALUE!	#VALUE!

### Formula definition

You can set Code39 in a worksheet using the following formula:

=BC\_CODE39(value, color, backgroundColor, showLabel, labelPosition, labelWithStartAndStopCharacter, checkDigit, nwRatio, fullASCII, fontFamily, fontStyle, fontWeight, fontTextDecoration, fontTextAlign, fontSize, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

### Parameter

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

value	A string that represents encode on the symbol of Code39.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
showLabel	Specifies whether to show label text when the barcode has label.
labelPosition	A value that represents the label position when the label is shown.
labelWithStartAndStopCharacter	Specifies whether to show the start and stop character in the label. The default value is 'false'.
checkDigit	Specifies whether the symbol needs a check digit. The default value is 'false'.
nwRatio	A value that represents the wide and narrow bar ratio. It has values 2 3. The default value is '3'.
fullASCII	Specifies whether to support full ASCII for Code39. The default value is 'false'.
fontFamily	A string that represents the label text fontFamily. The default value is 'sans-serif'.
fontStyle	A string that represents the label text fontStyle. The default value is 'normal'.
fontWeight	A string that represents the label text fontWeight. The default value is 'normal'.
fontTextDecoration	A string that represents the label text fontTextDecoration. The default value is 'none'.
fontTextAlign	A string that represents the label text fontTextAlign. The default value is 'center'.
fontSize	A string that represents the label text fontSize. The default value is '12px'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

## Using Code

This example code sets Code39 in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:C"].ColumnWidth = 15;
worksheet.Range["D:H"].ColumnWidth = 25;
worksheet.Range["4:6"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "Code39";
```

```
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:H3"].Value = new object[,]{
    {"Name", "Number", "Default", "Change labelWithStartAndStopCharacter", "Change
checkDigit", "Change checkDigit", "Change nwRatio", "Change fullASCII"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C6"].Value = new object[,]{
    {
        {"Paper", "6922266446146"},
        {"Book", "9787560044231"},
        {"Value can contain some symbol", "1234+-#*"}
    }
};
worksheet.Range["B4:C6"].WrapText = true;
worksheet.Range["G6"].WrapText = true;
worksheet.PageSetup.PrintTitleColumns = "$A:$C";
worksheet.PageSetup.Orientation = PageOrientation.Landscape;
worksheet.PageSetup.PrintGridlines = true;








// Set formula
for (var i = 4; i < 7; i++)
{
    worksheet.Range["D" + i].Formula = "=BC_CODE39" + "(C" + i + ")";
    worksheet.Range["E" + i].Formula = "=BC_CODE39" + "(C" + i + ",,,,,,\"true\")";
    worksheet.Range["F" + i].Formula = "=BC_CODE39" + "(C" + i + ",,,,,,\"true\")";
    worksheet.Range["G" + i].Formula = "=BC_CODE39" + "(C" + i + ",,,,,,2)";
    worksheet.Range["H" + i].Formula = "=BC_CODE39" + "(C" + i + ",,,,,,\"true\")";
}

// Save to a pdf file
workbook.Save("code39.pdf");
```

## Code93

Code93 barcode is a barcode that uses uppercase characters and numeric characters along with some special characters ("% ", "\*", "\$", "/", ".", "-", "+"). It is used primarily by Canada Post to encode supplementary delivery information.

The below image displays Code93 barcode in a PDF document.

Code93					
Name	Number	Default	Change checkDigit	Change fullASCII	
Pen	6945091701532				
Book	9787560044231				
Value can contain letters	123abc	#VALUE!	#VALUE!		

### Formula definition

You can set Code93 in a worksheet using the following formula:

=BC\_CODE93(value, color, backgroundColor, showLabel, labelPosition, checkDigit, fullASCII, fontFamily, fontStyle, fontWeight, fontTextDecoration, fontTextAlign, fontSize, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

### Parameter

Name	Description
value	A string that represents encode on the symbol of Code93.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
showLabel	Specifies whether to show label text when the barcode has label.
labelPosition	A value that represents the label position when the label is shown.
checkDigit	Specifies whether the symbol needs a check digit. The default value is 'false'.
fullASCII	Specifies whether to support full ASCII for Code93. The default value is 'false'.
fontFamily	A string that represents the label text fontFamily. The default value is 'sans-serif'.
fontStyle	A string that represents the label text fontStyle. The default value is 'normal'.
fontWeight	A string that represents the label text fontWeight. The default value is 'normal'.
fontTextDecoration	A string that represents the label text fontTextDecoration. The default value is 'none'.
fontTextAlign	A string that represents the label text fontTextAlign. The default value is 'center'.
fontSize	A string that represents the label text fontSize. The default value is '12px'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.

quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

### Using Code

This example code sets Code93 in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:F"].ColumnWidth = 20;
worksheet.Range["4:6"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "Code93";
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:G3"].Value = new object[,]{
    {"Name", "Number", "Default", "Change checkDigit", "Change fullASCII"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C6"].Value = new object[,]{
    {
        {"Pen", "6945091701532"},
        {"Book", "9787560044231"},
        {"Value can contain letters", "123abc"}
    }
};
worksheet.Range["B4:C6"].WrapText = true;
worksheet.Range["G6"].WrapText = true;
worksheet.PageSetup.Orientation = PageOrientation.Landscape;
worksheet.PageSetup.PrintGridlines = true;













// Set formula
for (var i = 4; i < 7; i++)
{
    worksheet.Range["D" + i].Formula = "=BC_CODE93" + "(C" + i + ")";
    worksheet.Range["E" + i].Formula = "=BC_CODE93" + "(C" + i + ",,,,,,\"true\")";
    worksheet.Range["F" + i].Formula = "=BC_CODE93" + "(C" + i + ",,,,,,\"true\")";
}

// Save to a pdf file
workbook.Save("code93.pdf");
```

## Code128

The Code 128 barcode is a linear barcode that represents high-density linear symbology to encode text, numbers, various functions and the entire 128 ASCII character set (from ASCII 0 to ASCII 128). It is widely used in enterprise internal management, production process, logistics control system of the bar code system.

The below image displays Code128 barcode in a PDF document.

	Code128				
	Name	Number	Default	Hidden Label	Custom Label Font
	Police	911			
	Telephone Directory Assistance	411			
	Non-emergency Municipal Services	311			
	Travel Info Call 511	511			

### Formula definition

You can set Code128 in a worksheet using the following formula:

=BC\_CODE128(value, color, backgroundColor, showLabel, labelPosition, codeSet, fontFamily, fontStyle, fontWeight, fontTextDecoration, fontTextAlign, fontSize, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

### Parameter

Name	Description
value	A string that represents encode on the symbol of Code128.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
showLabel	Specifies whether to show label text when the barcode has label.
labelPosition	A value that represents the label position when the label is shown.
codeSet	A value that represents which code is set to use for QRCode. It has 'auto A B C' values. The default value is 'auto'.
fontFamily	A string that represents the label text fontFamily. The default value is 'sans-serif'.

fontStyle	A string that represents the label text fontStyle. The default value is 'normal'.
fontWeight	A string that represents the label text fontWeight. The default value is 'normal'.
fontTextDecoration	A string that represents the label text fontTextDecoration. The default value is 'none'.
fontTextAlign	A string that represents the label text fontTextAlign. The default value is 'center'.
fontSize	A string that represents the label text fontSize. The default value is '12px'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

### Using Code

This example code sets Code128 in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:F"].ColumnWidth = 20;
worksheet.Range["4:7"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "Code128";
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:F3"].Value = new object[,]{
    {"Name", "Number", "Default", "Hidden Label", "Custom Label Font"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C7"].Value = new object[,]{
    {
        {"Police", 911},
        {"Telephone Directory Assistance", 411},
        {"Non-emergency Municipal Services", 311},
        {"Travel Info Call 511", 511}
    }
};
worksheet.Range["B4:C6"].WrapText = true;
worksheet.Range["G6"].WrapText = true;
worksheet.PageSetup.Orientation = PageOrientation.Landscape;
worksheet.PageSetup.PrintGridlines = true;
```









```
// Set formula
for (var i = 4; i < 8; i++)
{
    worksheet.Range["D" + i].Formula = "=BC_CODE128" + "(C" + i + ")";
    worksheet.Range["E" + i].Formula = "=BC_CODE128" + "(C" + i + ", , , false)";
    worksheet.Range["F" + i].Formula = "=BC_CODE128" + "(C" + i + ", , , true, \"top\", \"B\", \"Arial\", \"normal\")";
}

// Save to a pdf file
workbook.Save("code128.pdf");
```

## GS1-128

GS1-128 is a barcode that uses a series of application Identifiers in order to encode data. It makes use of the complete ASCII character set while also using FNC1 character as the first character position. This barcode is especially used for dates, batch numbers, weights and HIBC applications etc.

The below image displays GS1-128 barcode in a PDF document.

GS1128					
	Name	Number	Default	Hidden Label	Custom Label Font
	Police	911			
	Telephone Directory Assistance	411			
	Non-emergency Municipal Services	311			
	Travel Info Call 511	511			

### Formula definition

You can set GS1-128 in a worksheet using the following formula:

=BC\_GS1\_128(value, color, backgroudColor, showLabel, labelPosition, fontFamily, fontStyle, fontWeight, fontTextDecoration, fontTextAlign, fontSize, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

### Parameter



Name	Description
value	A string that represents encode on the symbol of GS1-128.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
showLabel	Specifies whether to show label text when the barcode has label.
labelPosition	A value that represents the label position when the label is shown.
fontFamily	A string that represents the label text fontFamily. The default value is 'sans-serif'.
fontStyle	A string that represents the label text fontStyle. The default value is 'normal'.
fontWeight	A string that represents the label text fontWeight. The default value is 'normal'.
fontTextDecoration	A string that represents the label text fontTextDecoration. The default value is 'none'.
fontTextAlign	A string that represents the label text fontTextAlign. The default value is 'center'.
fontSize	A string that represents the label text fontSize. The default value is '12px'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

### Using Code

This example code sets GS1\_128 in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:F"].ColumnWidth = 20;
worksheet.Range["4:7"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "GS1128";
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:F3"].Value = new object[,]{
    {"Name", "Number", "Default", "Hidden Label", "Custom Label Font"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
```

```

worksheet.Range["B4:C7"].Value = new object[,]
{
    {"Police", 911},
    {"Telephone Directory Assistance", 411},
    {"Non-emergency Municipal Services", 311},
    {"Travel Info Call 511", 511}
};
worksheet.Range["B4:C6"].WrapText = true;
worksheet.Range["G6"].WrapText = true;
worksheet.PageSetup.PrintGridlines = true;
worksheet.PageSetup.Orientation = PageOrientation.Landscape;

// Set formula
for (var i = 4; i < 8; i++)
{
    worksheet.Range["D" + i].Formula = "=BC_CODE128" + "(C" + i + ")";
    worksheet.Range["E" + i].Formula = "=BC_CODE128" + "(C" + i + ", , , false)";
    worksheet.Range["F" + i].Formula = "=BC_CODE128" + "(C" + i + ", , , true, \"top\", \"Arial\", \"normal\")";
}







// Save to a pdf file
workbook.Save("gs1128.pdf");

```

## Code49

Code 49 is a two dimensional, high-density stacked barcode with two to eight rows (having eight characters each). Each row has a start code and a stop code. This barcode is especially used to encode the complete ASCII character set.

The below image displays Code49 barcode in a PDF document.

Code49					
	Name	Number	Default	Customer Label Font	Line Through Label
	Police	911	 Police: 911	 Police: 911	 Police: 911
	Travel Info Call 511	511	 Travel Info Call 511: 511	 Travel Info Call 511: 511	 Travel Info Call 511: 511

### Formula definition

You can set Code49 in a worksheet using the following formula:

=BC\_CODE49(value, color, backgroundcolor, showLabel, labelPosition, grouping, groupNo, fontFamily, fontStyle,

fontWeight, fontTextDecoration, fontTextAlign, fontSize, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

#### Parameter

Name	Description
value	A string that represents encode on the symbol of Code49.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
showLabel	Specifies whether to show label text when the barcode has label.
labelPosition	A value that represents the label position when the label is shown.
grouping	Specifies whether the symbol mode is Group Alphanumeric Mode. The default value is 'false'.
groupNo	A value that represents the index of symbol in the group. The default value is '0'
fontFamily	A string that represents the label text fontFamily. The default value is 'sans-serif'.
fontStyle	A string that represents the label text fontStyle. The default value is 'normal'.
fontWeight	A string that represents the label text fontWeight. The default value is 'normal'.
fontTextDecoration	A string that represents the label text fontTextDecoration. The default value is 'none'.
fontTextAlign	A string that represents the label text fontTextAlign. The default value is 'center'.
fontSize	A string that represents the label text fontSize. The default value is '12px'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

#### Using Code

This example code sets Code49 in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:C"].ColumnWidth = 10;
worksheet.Range["D:F"].ColumnWidth = 30;
worksheet.Range["4:5"].RowHeight = 80;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "Code49";
```

```

worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:G3"].Value = new object[,]{
    {"Name", "Number", "Default", "Customer Label Font", "Line Through Label"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C5"].Value = new object[,]{
    {
        {"Police", "911"},
        {"Travel Info Call 511", "511"},
    };
};
worksheet.Range["B4:C6"].WrapText = true;
worksheet.Range["G6"].WrapText = true;
worksheet.PageSetup.PrintGridlines = true;
worksheet.PageSetup.Orientation = PageOrientation.Landscape;

// Set formula
for (var i = 4; i < 6; i++)
{
    var value = "CONCAT(B" + i + ", \": \",C" + i + ")";
    worksheet.Range["D" + i].Formula = "=BC_CODE49" + "(" + value + ")";
    worksheet.Range["E" + i].Formula = "=BC_CODE49" + "(" + value + ", , , true, \"top\", false, 0, \"Arial\", \"normal\", 700)";
    worksheet.Range["F" + i].Formula = "=BC_CODE49" + "(" + value + ", , , , , , , , , 700, \"line - through\", \"left\", \"24px\")";
}

// Save to a pdf file
workbook.Save("code49.pdf");

```

## PDF417

PDF417 barcode is a popular high-density, two-dimensional barcode with symbology that possesses the capability to encode up to 1108 bytes of information. This barcode comprises a stacked set of small barcodes and can encode up to 35 alphanumeric characters and 2,710 numeric characters. It is a stacked linear barcode format which is used in a variety of applications such as transport, identification cards, and inventory management.

The below image displays PDF417 barcode in a PDF document.

PDF417					
Server	Data	Default	Customer Padding	Customer Columns Count	
Police	911				
Telephone Directory Assistance	411				
Non-emergency Municipal Services	311				
Travel Info Call 511	511				

### Formula definition

You can set PDF417 in a worksheet using the following formula:

=BC\_PDF417(value, color, backgroudColor, errorCorrectionLevel, rows, columns, compact, quietZoneLeft, quietZoneRight, quietZoneTop, quietZoneBottom)

### Parameter

Name	Description
value	A string that represents encode on the symbol of PDF417.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
errorCorrectionLevel	A string that represents the error correction level of PDF417. It has 'auto 0-8' values. The default value is 'auto'.
rows	A value that specifies the number of rows in the symbol. It has 'auto 3-90' values.The default value is 'auto'.
columns	A value that specifies the number of columns in the symbol. It has 'auto 1-30' values. The default value is 'auto'.
compact	Specifies whether it is a compact PDF417. The default value is 'false'.
quietZoneLeft	A value that represents the size of left quiet zone.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

### Using Code

This example code sets PDF417 in the worksheet.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:C"].ColumnWidth = 12;
worksheet.Range["D:F"].ColumnWidth = 30;
worksheet.Range["4:7"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "PDF417";
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:F3"].Value = new object[,]{
    {"Server", "Data", "Defult", "Customer Padding", "Customer Columns Count"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C7"].Value = new object[,]{
    {
        {"Police", "911"},
        {"Telephone Directory Assistance", "411"},
        {"Non-emergency Municipal Services", "311"},
        {"Travel Info Call 511", "511"}
    }
};
worksheet.Range["B4:B7"].WrapText = true;
worksheet.PageSetup.Orientation = PageOrientation.Landscape;
worksheet.PageSetup.PrintGridlines = true;













// Set formula
for (var i = 4; i < 8; i++)
{
    var value = "CONCAT(B" + i + ", \":\", C" + i + ")";
    worksheet.Range["D" + i].Formula = "=BC_PDF417" + "(" + value + ")";
    worksheet.Range["E" + i].Formula = "=BC_PDF417" + "(" + value + ", , , , , , 0, 10, 5, 5)";
    worksheet.Range["F" + i].Formula = "=BC_PDF417" + "(" + value + ", , , , , 5)";
}

// Save to a pdf file
workbook.Save("pdf417.pdf");
```

## Data Matrix

DataMatrix barcode is a high density, two-dimensional barcode with square modules typically arranged in a square or a rectangular matrix pattern. The most popular application for Data Matrix is marking small items, due to the code's ability to encode fifty characters in a symbol that is readable at 2 or 3 mm<sup>2</sup> and the fact that the code can be read with only a 20% contrast ratio.

The below image displays DataMatrix barcode in a PDF document.

	Data Matrix				
	Server	Data	Default	ECC100	ECC200
	Police	911			
	Telephone Directory Assistance	411			
	Non-emergency Municipal Services	311			
	Travel Info Call 511	511			

### Formula definition

You can set Datamatrix in a worksheet using the following formula:

=BC\_DataMatrix(value, color, backgroundColor, eccMode, ecc200SymbolSize, ecc200EncodingMode, ecc00\_140Symbole, structureAppend, structureNumber, fileIdIdentifier, quietZoneRight, quietZoneTop, quietZoneBottom)

### Parameter

Name	Description
value	A string that represents encode on the symbol of Datamatrix.
color	A color that represents the barcode color. The default value is 'rgb(0,0,0)'.
backgroundColor	A color that represents the barcode backgroundcolor. The default value is 'rgb(255, 255, 255)'
eccMode	A value that represents which ecc mode to use. It has the following values : 'ECC000, ECC050, ECC080, ECC100, ECC140, ECC200'.
ecc200SymbolSize	A value that specifies the size of the ECC200 symbol only. The default value is 'squareAuto'.

ecc200EncodingMode	A value that specifies which encoding mode to use for the symbol. The default value is 'auto'.
ecc00_140Symbole	A value that specifies the size of the ECC000-140 symbol only. The default value is 'auto'.
structureAppend	Specifies whether the symbol is part of a structured append message ECC200 only. The default value is 'false'.
structureNumber	A value that represents which block the symbol is in the structured append message. It has the value '0-15', only for ECC200. The default value is '0'.
fileIdentifier	A value that specifies the file identification. It has values '1-254', only for ECC200. The default value is '0'.
quietZoneRight	A value that represents the size of right quiet zone.
quietZoneTop	A value that represents the size of top quiet zone.
quietZoneBottom	A value that represents the size of bottom quiet zone.

### Using Code

This example creates a DataMatrix barcode.

C#

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

// Set worksheet layout and data
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B:F"].ColumnWidth = 15;
worksheet.Range["4:7"].RowHeight = 60;
worksheet.Range["A:A"].ColumnWidth = 5;
worksheet.Range["B2"].Value = "Data Matrix";
worksheet.Range["B2:F2"].Merge(true);
worksheet.Range["B3:F3"].Value = new object[,]{
    {"Server", "Data", "Default", "ECC100", "ECC200"}
};
worksheet.Range["B4:C7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C7"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:F3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:F3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C7"].Value = new object[,]
{
    {"Police", "911"},
    {"Telephone Directory Assistance", "411"},
    {"Non-emergency Municipal Services", "311"},
    {"Travel Info Call 511", "511"}
};
worksheet.Range["B4:B7"].WrapText = true;
worksheet.PageSetup.PrintGridlines = true;
```



```
// Set formula
for (var i = 4; i < 8; i++)
{
    var value = "CONCAT(B" + i + ", \":\", C" + i + ")";
    worksheet.Range["D" + i].Formula = "=BC_DataMatrix" + "(" + value + ")";
    worksheet.Range["E" + i].Formula = "=BC_DataMatrix" + "(" + value + ", ,
, \"ECC000\")";
    worksheet.Range["F" + i].Formula = "=BC_DataMatrix" + "(" + value + ", ,
, \"ECC200\")";
}

// Save to a pdf file
workbook.Save("datamatrix.pdf");
```

### Limitation

- Datamatrix ECC (000-140) barcodes are obsolete. Hence, barcode generation with these specifications is not scanned.

## 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.

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**.

To change the current theme of the workbook, you need to first get the existing theme using the indexer notation of the **Themes class**.

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

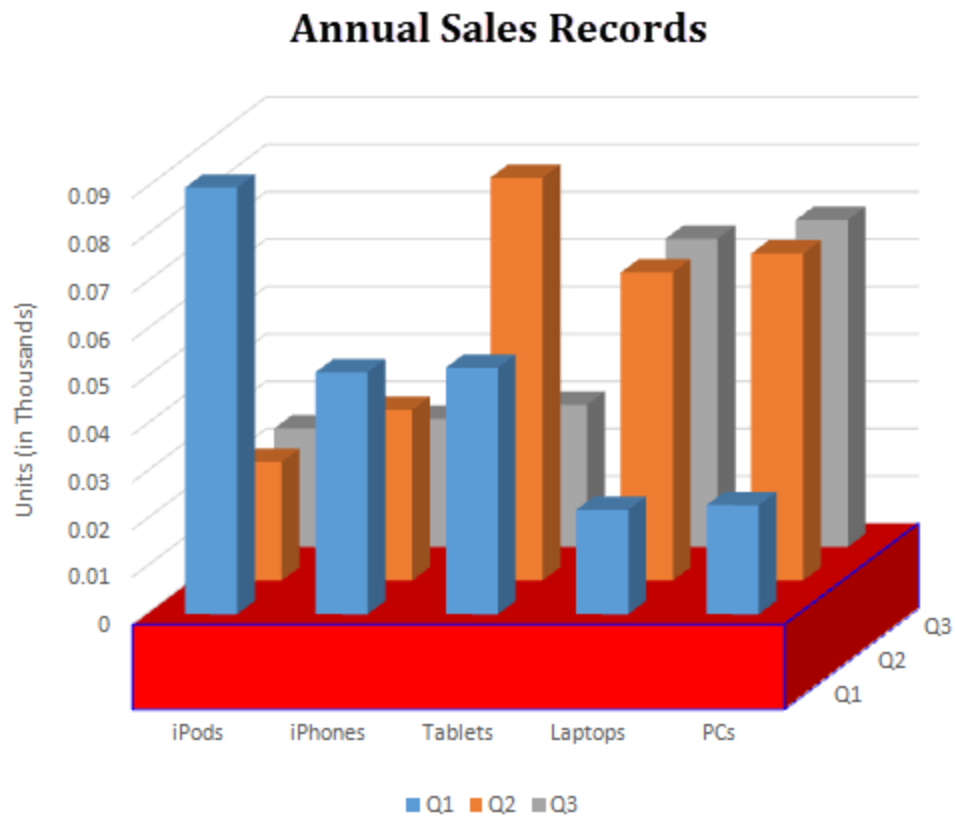
Theme theme = new Theme("testtheme");    // Base theme is office theme, if parameters are
not given

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;
```

## Chart

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



Working with charts involves the following tasks:

- [Create and Delete Chart](#)
- [Configure Chart](#)
- [Customize Chart Objects](#)
- [Chart Types](#)
- [Chart Sheet](#)

Charts can also be exported to PDF documents. For more information, refer to [Export Charts](#)

## 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** and the **ICChart interface**

### Create Chart

You can create chart in a worksheet by using the **AddChart method** 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}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);
```

## Delete Chart

You can delete an existing chart by using **Delete method** 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** 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 text angle for chart title**

### 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;
```

### Set text angle for chart title

You can also configure the text angle for chart title by using the **Orientation** property of **IChartTitle** interface. The text angle can also be exported or imported to JSON.

Refer to the following example code to set text angle for chart title.

```
C#  
  
//add chart title  
shape.Chart.HasTitle = true;  
shape.Chart.ChartTitle.Text = "MyChartTitle";  
  
//config chart title angle  
shape.Chart.ChartTitle.Orientation = 30;
```

## Chart Area

In GcExcel .NET, you can use the properties of the **IChartArea interface** to set up the chart area as per your preferences.

### Configure chart area style

You can configure the chart area style by changing its font, format and other attributes using the **Font property**, **Format property** and **RoundedCorners property** 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;
```

## Plot Area

In GcExcel .NET, you can use the properties of the **IPlotArea Interface** 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** 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;
```

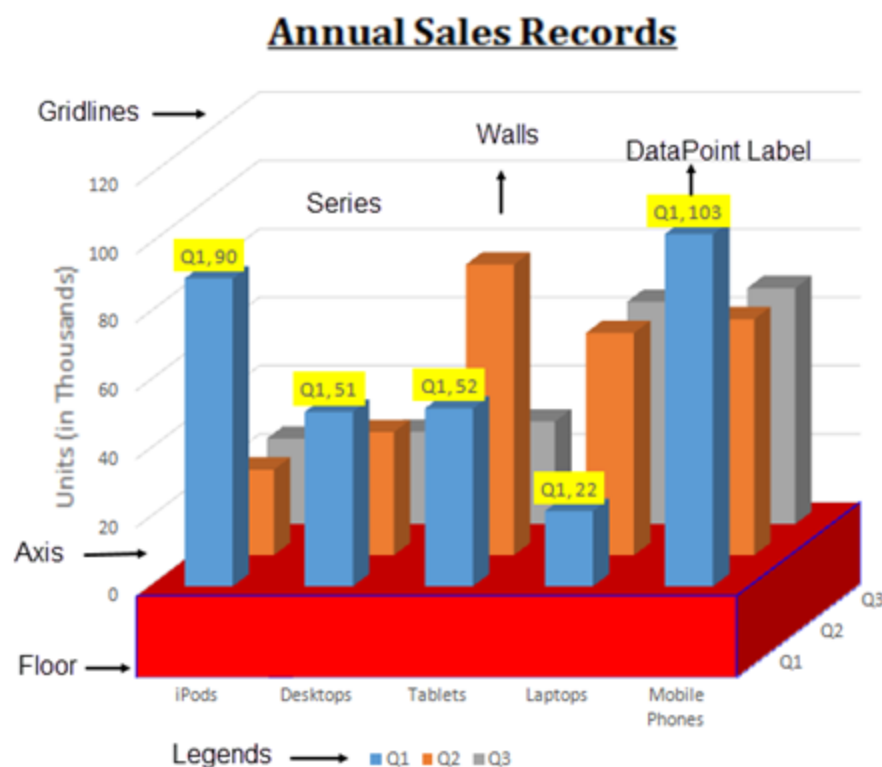
## Customize Chart Objects

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** and the **ISeriesCollection interface** 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 shape1 = 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.
shape1.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** to get a specific point in the series. Also, you can use the **DataLabel property** of the **IPoint interface** 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 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** and **Format property** of the **IDataLabel interface** 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 lables' 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 of the **ITrendlines** interface 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;
```

### Set trendline's name

You can also set the trendline's name in GcExcel using the **Name** property of **ITrendline** interface. The trendline's name can also be exported to a PDF document.

Refer to the following example code to add trendline's name in GcExcel.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

// Add a chart
IShape columnChart =
worksheet.Shapes.AddChart(GrapeCity.Documents.Excel.Drawing.ChartType.ColumnClustered,
300, 10, 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}
```



```
};

// Add series
columnChart.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);

// Get first series
ISeries series1 = columnChart.Chart.SeriesCollection[0];

// Add a trend line
ITrendline trendline = series1.Trendlines.Add();

// Set trend line's name
trendline.Name = "Theoretical data";

// Save to an excel file
workbook.Save("TrendLineName.xlsx");
```

## 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, 250, 20, 360, 230);
worksheet.Range["A1:C10"].Value = new object[,]
{
    { "Blue", null, null },
    { 125, 750, 3 },
    { 25, 625, 7 },
    { 75, 875, 5 },
    { 175, 625, 6 },
    { "Red", null, null },
    { 125, 500, 10 },
    { 25, 250, 1 },
    { 75, 125, 5 },
    { 175, 250, 8 },
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A2:C5"], RowCol.Columns);
shape.Chart.SeriesCollection.Add(worksheet.Range["A7:C10"], RowCol.Columns);

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, **HasHiLoLines** property, **HasSeriesLines** property, **DropLines** property, **HiLoLines** property, **SeriesLines** property of the **ICartGroup** interface 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**, **DownBars property** and **UpBars property** 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;

```

## Error Bars

Error bars are used in charts to indicate the error or uncertainty of data. They act as an extremely useful tool for scientists, statisticians, and research analysts to showcase data variability and measurement accuracy.


GcExcel allows you to configure error bars in charts using **IErrorBar** interface. The interface represents error bars in a chart series and provides properties to configure various types, end styles and value types of error bars. The error bars can also be exported or imported to JSON or a PDF document.



### Supported Chart Types

The following chart types are supported while adding error bars in charts:

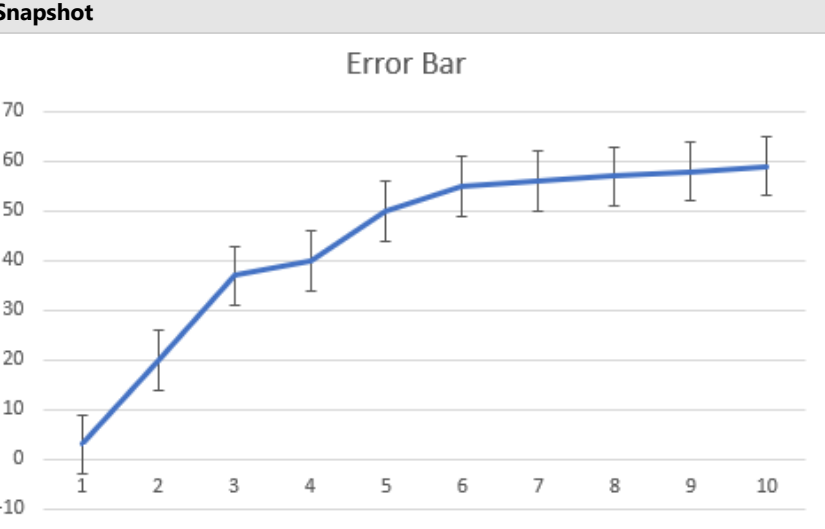
- Area Charts
- Bar Charts
- Column charts
- Line Charts
- xyScatter Charts

### Error Bar Types

Type	Snapshot	Description
Plus		<p>Error bar depicts only the positive values.</p> <pre> C# series.YErrorBar.Type = ErrorBarInclude.Plus; </pre>


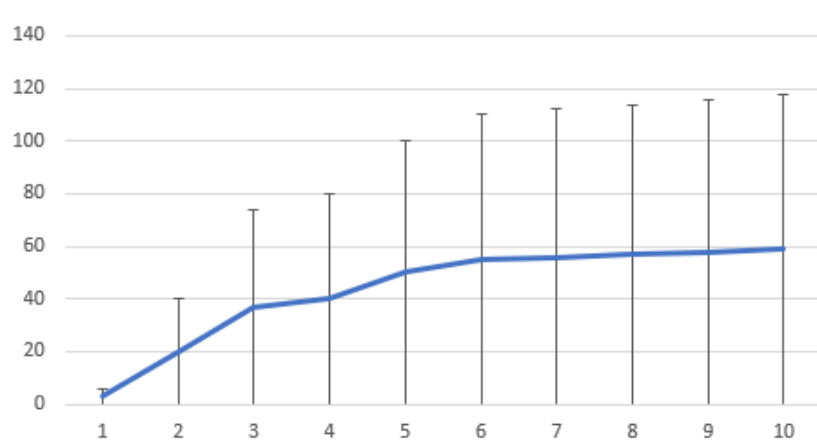
Minus		Error bar depicts only the negative values.
Both		Error bar depicts positive and negative values at the same time

### Error Bar End Styles

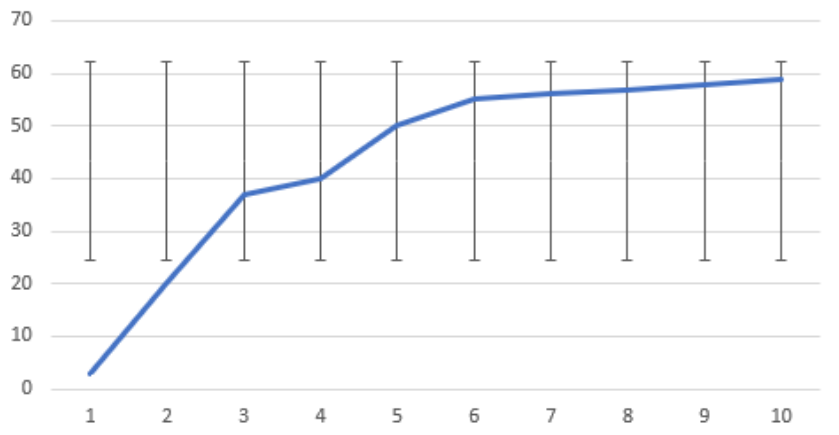
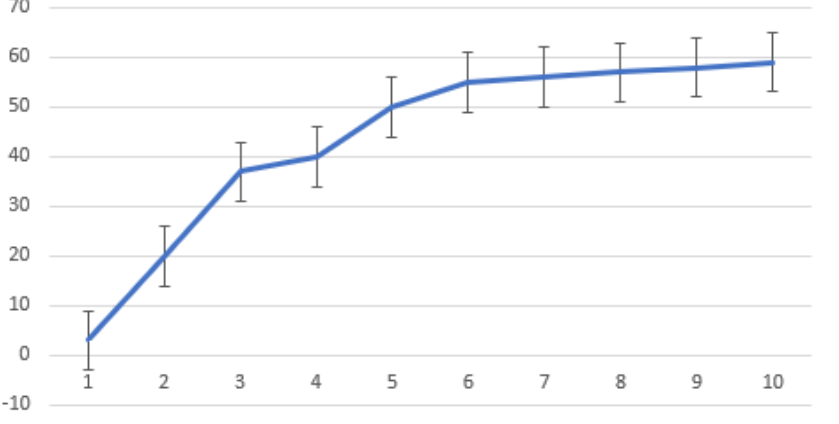
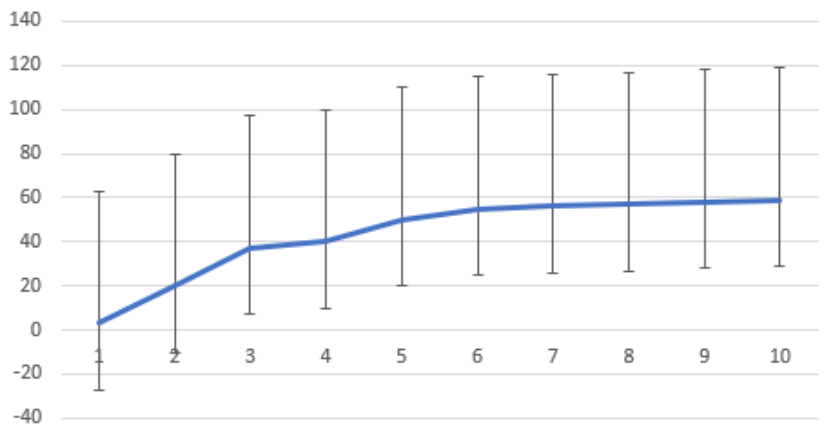
Type	Snapshot	Description
Cap		Error bar displays caps at the end of error bar lines.



### Error Bar Value Types

Type	Snapshot	Description
Fixed Value		Error bar represents the error as an absolute value. <pre>C# series1.YErrorBar.ValueType = ErrorBarType.FixedValue;</pre>
Percentage		Error bar represents the error as a percentage of data value in the same direction axis. <pre>C# series1.YErrorBar.ValueType = ErrorBarType.Percentage;</pre>



Standard Deviation	<p style="text-align: center;">Error Bar</p> 	<p>Error bar represents the error as a calculating value which depends on the set deviation and chart data values.</p> <pre>C# series1.YErrorBar.ValueType = ErrorBarType.StDev;</pre>
Standard Error	<p style="text-align: center;">Error Bar</p> 	<p>Error bar represents the error as a calculating value which only depends on the chart data values.</p> <pre>C# series1.YErrorBar.ValueType = ErrorBarType.StError;</pre>
Custom	<p style="text-align: center;">Error Bar</p> 	<p>Error bar represents the error values that are set with positive and negative values respectively by formulas or fixed values.</p> <pre>C# series1.YErrorBar.ValueType = ErrorBarType.Custom;</pre>

**Note:** In Custom value type, the array and reference formula string for plus or minus is supported. The final count of error bar values is evaluated by the formula string (for example, "`=Sheet1!$B$2:$D$2`" or "`={1,2,3}`"). The error bar values are displayed based on the total count of values:

If count = 1: all error bars have the same value.

If count < number of data points: the value of rest of the error bars is zero.

If count > number of data points: the remaining values will do nothing.

## Using Code

Refer to the following example code to add error bars using various properties.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];
// Prepare data for chart
worksheet.Range["A1:D4"].Value = new object[,]
{
    {null, "Q1", "Q2", "Q3"},
    {"Mobile Phones", 1330, 2345, 3493},
    {"Laptops", 2032, 3632, 2197},
    {"Tablets", 6233, 3270, 2030}
};
worksheet.Range["A:D"].Columns.AutoFit();
// Add Column Chart
IShape columnChartshape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 250, 20, 360,
230);

// Adding series to SeriesCollection
columnChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"], RowCol.Columns, true,
true);

// Get first series
ISeries series1 = columnChartshape.Chart.SeriesCollection[0];

//Config first series' properties
series1.HasErrorBars = true;
series1.YErrorBar.Type = ErrorBarInclude.Both;
series1.YErrorBar.ValueType = ErrorBarType.Custom;
series1.YErrorBar.EndStyle = EndStyleCap.Cap;
series1.YErrorBar.Plus = "{200,400,600}";
series1.YErrorBar.Minus = "{600,400,200}";

// Get second series
ISeries series2 = columnChartshape.Chart.SeriesCollection[1];

//Config second series' properties
series2.HasErrorBars = true;
series2.YErrorBar.Type = ErrorBarInclude.Plus;
series2.YErrorBar.ValueType = ErrorBarType.FixedValue;
series2.YErrorBar.EndStyle = EndStyleCap.Cap;
series2.YErrorBar.Amount = 1000;
series2.YErrorBar.Format.Line.Color.RGB = Color.Red;
series2.YErrorBar.Format.Line.Weight = 2;
```

```
// Get last series
ISeries series3 = columnChartshape.Chart.SeriesCollection[2];

//Config last series' properties
series3.HasErrorBars = true;
series3.YErrorBar.Type = ErrorBarInclude.Both;
series3.YErrorBar.ValueType = ErrorBarType.StError;
series3.YErrorBar.EndStyle = EndStyleCap.NoCap;

//save to an excel file
workbook.Save("ErrorBar.xlsx");
```

### Important Points

- Only series in scatter chart groups can have x and y error bars. Otherwise, an exception would be thrown.
- ISeries.HasErrorBars must be set as "true" to display error bar.
- IErrorBar.Amount only takes effect when IErrorBar.ValueType is FixedValue or Percentage or Standard Deviation.
- IErrorBar.Plus or IErrorBar.Minus only takes effect when IErrorBar.ValueType is Custom.
- IErrorBar.Plus or IErrorBar.Minus accepts a formula string like "=Sheet1!\$B\$2:\$D\$2" or "{1,2,3}".

### Limitation

There can be some difference between the exported PDF and Excel containing error bars. It is caused due to different ways of calculating error bar value between GcExcel and Excel.

## 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** and the **IChart Interface**.

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;
```

```
{
    {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 of the **IAxis** interface.

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, **HasMinorGridlines** property, **MajorGridlines** property and **MinorGridlines** property of the **IAxis** interface.

Refer to the following example code to set major and minor gridlines' style.

### 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, **DisplayUnitLabel** property and **HasDisplayUnitLabel** property 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 set the position and layout of the tick-mark labels as per your choice using the **TickLabelPosition property**, **TickLabels property**, **TickLabelSpacing property**, **TickLabelSpacingIsAuto property** and **TickMarkSpacing property** 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;
```

You can also configure the text angle of tick-mark labels by using the **Orientation** property of **ITickLabels** interface. The text angle can also be exported or imported to JSON or PDF document.

Refer to the following example code to set the text angle of tick mark label.

C#

```
//create a new workbook
var workbook = new Workbook();

IWorksheet worksheet = workbook.Worksheets[0];

//add chart
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {1, -25, 25, 25},
    {2, 51, 36, 27},
    {3, 52, 80, 30},
    {4, 22, -20, 65},
    {5, 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);
IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);

//config tick label's angle
category_axis.TickLabels.Orientation = 45;

//save to an excel file
workbook.Save("configtickmarklabelangle.xlsx");
```

## 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** 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** and the **ISeries interface**.

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 shape1 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 50, 300, 300);
shape1.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);
ISeries series1 = shape1.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 labels' text.
series2.DataLabels.ShowCategoryName = true;
series2.DataLabels.ShowSeriesName = true;
series2.DataLabels.ShowLegendKey = true;
```

You can also configure the text angle for data labels by using the **Orientation** property of **IDataLabel** interface. The text angle can also be exported or imported to JSON.

Refer to the following example code to set text angle for data label.

C#

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

IWorksheet worksheet = workbook.Worksheets[0];

//add chart
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.Range["A1:B5"].Value = new object[,]
{
    {null, "S1"},
    {"Item1", -20},
    {"Item2", 30},
    {"Item3", 50 },
    {"Item3", 40 }
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:B5"], RowCol.Columns, true, true);
ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

//set series1's all data labels' angle
series1.DataLabels.Orientation = 45;

//set series1's specific data label's angle
series1.DataLabels[2].Orientation = -45;
```

```
//save to an excel file
workbook.Save("configdatalabelangle.xlsx");
```

## 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** and the **IChart interface**.

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 shape1 = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 450, 300, 300);  
shape1.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);  
shape1.Chart.HasLegend = true;  
ILegend legend1 = shape1.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();
```

## Chart Types

**GcExcel** supports a wide range of chart types such as Area, Column, Line, Pie, Bar, Combo, Stock, Surface, Scatter, Radar, Statistical and Specialized charts. It also supports new Excel 2016 statistical and specialized chart types like Sunburst, Pareto, Treemap, Histogram, WaterFall, Box and Whisker, and Funnel. The new chart types represent and analyze hierarchical data better than conventional charts.

This topic gives a quick snapshot of all major chart types and their use cases.


Chart Type	Chart Snapshot	Use Case
<b>Area Charts</b> <ul style="list-style-type: none"><li>Area</li><li>Area3D</li><li>AreaStacked</li><li>AreaStacked100</li><li>AreaStacked1003D</li><li>AreaStacked3D</li></ul>		An Area chart is used to represent data that follows a time-series relationship. This type of chart is ideal when you need to show the plot change over time and depict the total value across a trend by showing the sum of the plotted values.


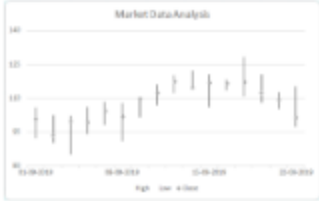
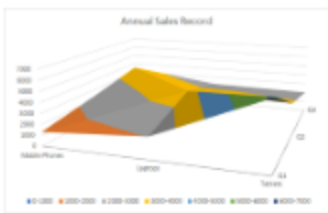




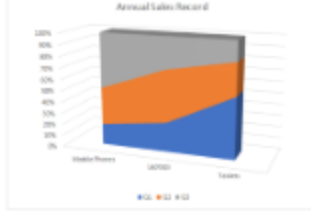
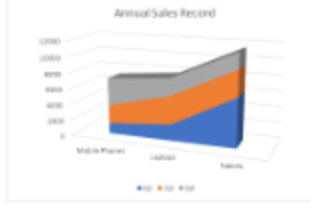
Chart Type	Chart Snapshot	Use Case
<b>Bar Charts</b> <ul style="list-style-type: none"> <li>BarClustered</li> <li>BarClustered3D</li> <li>BarStacked</li> <li>BarStacked100</li> <li>BarStacked1003D</li> <li>BarStacked3D</li> </ul>		<p>Bar charts are used for showing patterns and trends across different categories. In these charts, each horizontal bar corresponds to a category and its length corresponds to the value or measure of that category.</p>
<b>Column Charts</b> <ul style="list-style-type: none"> <li>Column3D</li> <li>ColumnClustered</li> <li>ColumnClustered3D</li> <li>ColumnStacked</li> <li>ColumnStacked100</li> <li>ColumnStacked1003D</li> <li>ColumnStacked3D</li> </ul>		<p>Unlike bar charts, Column charts use vertical columns/bars for representing data. These charts are generally used to plot data easily on X-axis.</p>
<b>Combo Chart</b> <ul style="list-style-type: none"> <li>Combo</li> </ul>		<p>The combo of two or more different charts can be used in the same plot area to compare the different data sets that are related to each other.</p>
<b>Line Charts</b> <ul style="list-style-type: none"> <li>Line</li> <li>Line3D</li> <li>LineMarkers</li> <li>LineMarkersStacked</li> <li>LineMarkersStacked100</li> <li>LineStacked</li> <li>LineStacked100</li> </ul>		<p>Line charts are used to plot continuously changing data against an interval of time. They can also be used to plot data against other continuous periodic values such as temperature, distance, humidity, share price, earnings per share etc.)</p>
<b>Pie Charts</b> <ul style="list-style-type: none"> <li>Pie</li> <li>Pie3D</li> <li>PieExploded</li> <li>PieExploded3D</li> <li>PieOfPie</li> <li>BarOfPie</li> <li>Doughnut</li> <li>DoughnutExploded</li> </ul>		<p>Pie charts are used to represent the relative contribution of various categories. It is one of the most commonly used charts and makes it easy to compare proportions by displaying the contribution of each value (slice) to a total (pie).</p>

Chart Type	Chart Snapshot	Use Case
<b>Stock Charts</b> <ul style="list-style-type: none"> <li>StockHLC</li> <li>StockOHLC</li> <li>StockVHLC</li> <li>StockVOHLC</li> </ul>		A Stock chart is used to illustrate fluctuations in data. It can represent fluctuations for stock, daily rainfall, or annual temperatures. Typically, this chart is ideal for analyzing financial data and visualizing stock information.
<b>Surface Charts</b> <ul style="list-style-type: none"> <li>Surface</li> <li>SurfaceTopView</li> <li>SurfaceTopViewWireframe</li> <li>SurfaceWireframe</li> </ul>		Surface charts are used to find the optimum combinations between two sets of data. As in a topographic map, the colors and patterns indicate the areas that are in the same range of values.
<b>XY (Scatter) Charts</b> <ul style="list-style-type: none"> <li>XYScatter</li> <li>XYScatterLines</li> <li>XYScatterLinesNoMarkers</li> <li>XYScatterSmooth</li> <li>XYScatterSmoothNoMarkers</li> <li>Bubble</li> <li>Bubble3DEffect</li> </ul>		An XY chart (also called scatter diagram) is a two-dimensional chart that shows the relationship between two variables. In a scatter graph, both horizontal and vertical axes are value axes that plot numeric data to show the correlation between two variables.
<b>Radar Charts</b> <ul style="list-style-type: none"> <li>Radar</li> <li>Radar Filled</li> <li>Radar Markers</li> </ul>		Radar charts are radial charts that help in visualizing comparison of two or more groups of values against various features or characteristics. These charts represent each variable on a separate axis, which are arranged radially at equal distances from each other.
<b>Statistical Charts</b> <ul style="list-style-type: none"> <li>Box and Whisker</li> <li>Histogram</li> <li>Waterfall</li> <li>Pareto</li> </ul>		Statistical charts help summarize and add visual meaning to key characteristics of data, including range, distribution, mean and median. It can also be used to in present and interpret statistical data in graphical format.
<b>Specialized Charts</b> <ul style="list-style-type: none"> <li>Sunburst</li> <li>Treemap</li> <li>Funnel</li> </ul>		Specialized chart types provided by GcExcel have unique data representation to show hierarchies and relationships. Such visual comparisons allow users to analyze the data thoroughly.

## Area Chart

An **Area Chart** can be used to represent the change in one or more data quantities over time. It is similar to a line graph. In area charts, the data points are plotted and connected by line segments. This helps in showing the magnitude of the value at different times. Unlike in line charts, the area between the line and x-axis is filled with color or shading in area charts.

GcExcel supports the following types of area charts.

Chart Type	Chart Snapshot	Use Case
Area		Area chart is used to depict the data series as colored regions that help in comparing the values of multiple series for the same data point. This chart shows trends over time.
Area3D		Area3D chart is used to represent the chart demonstration in 3D, which is a modification of 2D Area chart. It does not have a third dimension, it only looks volumetric in appearance.
AreaStacked		AreaStacked chart is used to depict data series as stacked regions with different colors that help in performing comparisons between multiple series for the same data point. This chart shows the trend of the contribution of each value over time or other categorical data.
AreaStacked100		AreaStacked100 chart is used to depict the series of data points with positive and negative values shown over time to reveal values of multiple series for the same data point. This chart shows the percentage that each value contributes over time or other categorical data.
AreaStacked1003D		AreaStacked1003D is used to represent the AreaStacked100 chart in 3D, which looks volumetric in appearance.
AreaStacked3D		AreaStacked3D chart is used to represent AreaStacked chart in 3D, which is a modification of the 2D Area chart.

## Using Code

Refer to the following example code to add Area Stacked Chart:

C#

```
public void AreaCharts()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];
    // Prepare data for chart
    worksheet.Range["A1:D4"].Value = new object[,]
    {
        {null, "Q1", "Q2", "Q3"},
        {"Mobile Phones", 1330, 2345, 3493},
        {"Laptops", 2032, 3632, 2197},
        {"Tablets", 6233, 3270, 2030}
    };

    worksheet.Range["A:D"].Columns.AutoFit();
    // Add Area Chart
    IShape areaChartShape = worksheet.Shapes.AddChart(ChartType.AreaStacked, 250, 20,
    360, 230);

    // Adding series to SeriesCollection
    areaChartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"],
    RowCol.Columns, true, true);

    // Configure Chart Title
    areaChartShape.Chart.ChartTitle.TextFrame.TextRange.Paragraphs.Add("Annual Sales
    Record");



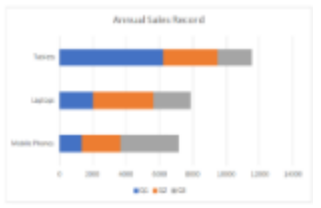



    // Saving workbook to Xlsx
    workbook.Save(@"18-AreaChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Bar Chart

**Bar charts** compare categorical data through horizontal bars, where length of each bar represents the value of the corresponding category. In bar charts, categories are organized along the vertical axis and data values along the horizontal axis. For example, sales of various product categories can be presented through a bar chart.

GcExcel supports the following types of bar charts.



Chart Type	Chart Snapshot	Use Case
BarClustered		BarClustered Chart can be used to display the comparisons of values across different categories.
BarClustered3D		BarClustered3D chart is used to display the chart demonstration in 3D, which is a modification of 2D BarClustered chart. It does not have a third dimension, it only looks volumetric in appearance.
BarStacked		BarStacked chart is used to display the relationship of each item/category to the whole in two-dimensional and three-dimensional rectangles.
BarStacked3D		BarStacked3D chart is used to represent the BarStacked chart demonstration in 3D, which looks volumetric in appearance.
BarStacked100		BarStacked100 chart is used to display the comparisons of percentage that each of the values contribute to the total across different categories.
BarStacked1003D		BarStacked1003D chart is used to represent the BarStacked100 chart demonstration in 3D, which is a modification of 2D chart in appearance.

## Using Code

Refer to the following example code to add Bar Stacked Chart:

C#

```
public void BarCharts()
```

```
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];
    // Prepare data for chart
    worksheet.Range["A1:D4"].Value = new object[,]
{
    {null, "Q1", "Q2", "Q3"},
    {"Mobile Phones", 1330, 2345, 3493},
    {"Laptops", 2032, 3632, 2197},
    {"Tablets", 6233, 3270, 2030}
};

    worksheet.Range["A:D"].Columns.AutoFit();
    // Add BarStacked Chart
    IShape barChartshape = worksheet.Shapes.AddChart(ChartType.BarStacked, 250, 20,
360, 230);

    // Adding series to SeriesCollection
    barChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"], RowCol.Columns,
true, true);

    // Configure Chart Title
    barChartshape.Chart.ChartTitle.TextFrame.TextRange.Paragraphs.Add("Annual Sales
Record");

    // Saving workbook to Xlsx
    workbook.Save(@"19-BarChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Column Chart

**Column charts** are vertical versions of bar charts and use x-axis as a category axis. Column charts are preferred where number of values is too large to be used on an x-axis, while bar charts are preferred where long category titles are difficult to fit on an x-axis. For example, population share of different countries across the globe can be represented using a column chart.

GcExcel supports the following types of column charts.







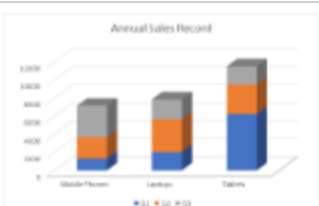
Chart Type	Chart Snapshot	Use Case
Column3D		Column3D chart is used to display the chart demonstration in 3D which is a modification of 2D Column chart. It does not have a third dimension, it only looks volumetric in appearance.

Chart Type	Chart Snapshot	Use Case
ColumnClustered		Column clustered chart is used to compare different values across different categories and show them in two-dimensional or three-dimensional vertical rectangles. This chart can be stacked normally in a regular way just like any other chart.
ColumnClustered3D		Column clustered chart to represent the ColumnClustered chart demonstration in 3D, which looks volumetric in appearance.
ColumnStacked		ColumnStacked chart is used to display the relationship of specific items to the whole across different categories and plot values in two-dimensional or three-dimensional vertical rectangles. This chart stacks the data series vertically (in a vertical direction).
ColumnStacked100		ColumnStacked100 chart is used to perform comparisons of percentages that each of the values are contributing to the total, across all your categories in the spreadsheet. This chart stacks the data series vertically and also equalizes the plotted values to meet 100%. The plotted values are displayed in two-dimensional and three-dimensional rectangles.
ColumnStacked1003D		ColumnStacked1003D is used to represent the ColumnStacked100 chart demonstration in 3D, which is a modification of 2D chart in appearance.
ColumnStacked3D		ColumnStacked3D chart is used to represent the ColumnStacked chart demonstration in 3D, which looks volumetric in appearance.

## Using Code

Refer to the following example code to add Column Stacked 3D Chart:

C#

```
public void ColumnCharts()
```

```

{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];
    // Prepare data for chart
    worksheet.Range["A1:D4"].Value = new object[,]
{
    {null, "Q1", "Q2", "Q3"},
    {"Mobile Phones", 1330, 2345, 3493},
    {"Laptops", 2032, 3632, 2197},
    {"Tablets", 6233, 3270, 2030}
};

    worksheet.Range["A:D"].Columns.AutoFit();
    // Add Column Chart
    IShape columnChartshape = worksheet.Shapes.AddChart(ChartType.ColumnStacked3D,
250, 20, 360, 230);

    // Adding series to SeriesCollection
    columnChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"],
RowCol.Columns, true, true);


    // Configure Chart Title
    columnChartshape.Chart.ChartTitle.TextFrame.TextRange.Paragraphs.Add("Annual Sales
Record");

    // Saving workbook to Xlsx
    workbook.Save(@"20-ColumnChart.xlsx", SaveFileFormat.Xlsx);
}

```

## Combo Chart

**Combo chart** is a combination of two or more chart types in a single plot area. For instance, a bar and line chart in a single plot. Combination charts are best used to compare the different data sets that are related to each other, such as actual and target values, total revenue and profit, temperature and precipitation etc. Note that these charts may require multiple axes to cater different scales.

Chart Type	Chart Snapshot	Use Case
Combo		Combo chart can be used to interpret and understand different type of data that is completely unrelated (for instance: price and volume) or to plot one or more data series on the secondary axis.

## Using Code

Refer to the following example code to add Combo Chart:

C#

```
public void ComboCharts()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:C17"].Value = new object[,] {
    { "Mobile Phones", "Laptops", "Tablets" },
    { 1350, 120, 75 },
    { 1500, 90, 35 },
    { 1200, 80, 50 },
    { 1300, 80, 80 },
    { 1750, 90, 100 },
    { 1640, 120, 130 },
    { 1700, 120, 95 },
    { 1100, 90, 80 },
    { 1350, 120, 75 },
    { 1500, 90, 35 },
    { 1200, 80, 50 },
    };

    worksheet.Range["A:C"].Columns.AutoFit();

    // Add Combination Chart
    IShape comboChartShape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 250,
20, 360, 230);
    // Adding series to SeriesCollection
    comboChartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:C17"],
RowCol.Columns);

    // Configure Chart Title
    comboChartShape.Chart.ChartTitle.Text = "Annual Sales Record-Combination Chart";
    ISeries series1 = comboChartShape.Chart.SeriesCollection[0];
    ISeries series2 = comboChartShape.Chart.SeriesCollection[1];
    ISeries series3 = comboChartShape.Chart.SeriesCollection[2];

    //Change series type to make it Combination chart of different ChartTypes
    series1.ChartType = ChartType.Area;
    series2.ChartType = ChartType.ColumnStacked;
    series3.ChartType = ChartType.Line;
```

```

//Set axis group
series2.AxisGroup = AxisGroup.Secondary;
series3.AxisGroup = AxisGroup.Secondary;

//Configure axis scale and unit
IAxis value_axis = comboChartShape.Chart.Axes.Item(AxisType.Value);
IAxis value_second_axis = comboChartShape.Chart.Axes.Item(AxisType.Value,
AxisGroup.Secondary);
value_axis.MaximumScale = 1800;
value_axis.MajorUnit = 450;
value_second_axis.MaximumScale = 300;
value_second_axis.MajorUnit = 75;

// Saving workbook to Xlsx
workbook.Save("24-ComboChart.xlsx", SaveFileFormat.Xlsx);
}

```

## Line Chart

**Line charts** are the most basic charts that are created by connecting the data points with straight lines. These charts are used to visualize a trend in data by comparing values against periodic intervals such as time, temperature etc. Some examples that can be well depicted using line charts are closing prices of a stock in a given time frame and monthly average sale of a product.

GcExcel supports the following types of line charts.


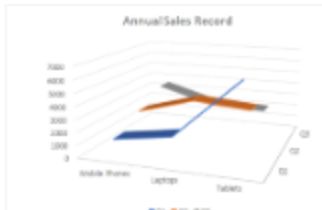
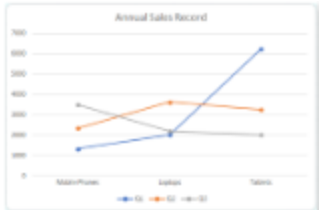




Chart Type	Chart Snapshot	Use Case
Line		Line chart is used to depict the data values plotted over time to display the trends. It shows continuous data over time on an evenly scaled Axis.
Line3D		Line3D chart is used to display the chart demonstration in 3D, which is a modification of 2D Line chart.
LineMarkers		LineMarkers chart is used to display data values shown with markers. It is ideal to use this chart when there are many categories or approximate values.

Chart Type	Chart Snapshot	Use Case
LineMarkersStacked		LineMarkersStacked is used to display data values with markers, typically showing the trend of contribution of each value over time or evenly spaced categories.
LineMarkersStacked100		LineMarkersStacked100 chart is used to display individual data values with markers, typically showing the trend of the percentage each value that has been contributed over time or evenly spaced categories. It is ideal to use this chart when there are many categories or approximate values.
LineStacked		LineStacked chart is used to display stacked line to depict the trend of contribution of each data value or ordered category over different time intervals.
LineStacked100		LineStacked100 chart is used to display displays trends in terms of the percentage that each data value or ordered category has contributed (to the whole) over different time intervals.

## Using Code

Refer to the following example code to add LineStacked100 chart:

C#

```
public void LineCharts()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];
    // Prepare data for chart
    worksheet.Range["A1:D4"].Value = new object[,]
    {
        {null, "Q1", "Q2", "Q3"},
        {"Mobile Phones", 1330, 2345, 3493},
        {"Laptops", 2032, 3632, 2197},
        {"Tablets", 6233, 3270, 2030}
    }
}
```

```

};
    worksheet.Range["A:D"].Columns.AutoFit();
    // Add Line Chart
    IShape lineChartshape = worksheet.Shapes.AddChart(ChartType.LineStacked100, 250,
20, 360, 230);

    // Adding series to SeriesCollection
    lineChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"],
RowCol.Columns, true, true);

    // Configure Chart Title
    lineChartshape.Chart.ChartTitle.TextFrame.TextRange.Paragraphs.Add("Annual Sales
Record");

    // Saving workbook to Xlsx
    workbook.Save(@"21-LineChart.xlsx", SaveFileFormat.Xlsx);
}

```

## Pie Chart

**Pie charts**, the most common tools used for data visualization, are circular graphs that display the proportionate contribution of each category, which is represented by a pie or a slice. The magnitude of the dependent variable is proportional to the angle of the slice. These charts can be used for plotting just one series with non-zero and positive values.

GcExcel supports the following types of pie charts.




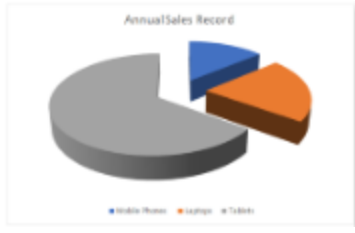
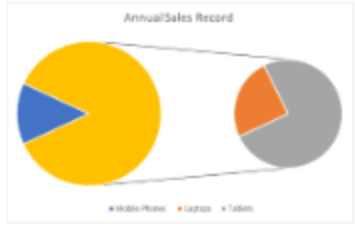
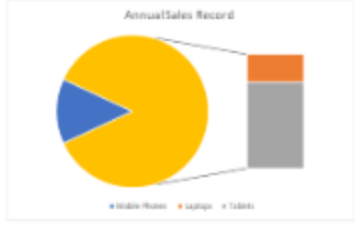


Chart Type	Chart Snapshot	Use Case
Pie		Pie chart is used to display a single data series in a circle-type structure, with each sector representing a different category.
Pie3D		Pie3D chart is used to display the chart demonstration in 3D which is a modification of 2DPie chart in terms of appearance.



Chart Type	Chart Snapshot	Use Case
PieExploded		PieExploded chart is used to pull all of the slices out of a pie chart and view the sectors separately in pieces.
PieExploded3D		PieExploded 3D chart is used display the chart demonstration in 3D which is a modification of 2DPieExploded chart.
PieOfPie		PieofPie chart is used to separate the slices from the main pie chart and display them in an additional pie chart.
BarOfPie		BarofPie chart is used to separate the slices from the main pie chart and display them in an additional stacked bar chart.
Doughnut		Doughnut chart is used to display multiple data series concurrently, with each ring depicting a single data series.
DoughnutExploded		DoughnutExploded is used to pull all slices out of a DoughnutExploded chart and view the sectors separately in pieces.

## Using Code

Refer to the following code to add Doughnut Exploded chart:

C#

```
public void PieCharts()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:D4"].Value = new object[,]
    {
        {null, "Q1", "Q2", "Q3"},
        {"Mobile Phones", 1330, 2345, 3493},
        {"Laptops", 2032, 3632, 2197},
        {"Tablets", 6233, 3270, 2030}
    };

    worksheet.Range["A:D"].Columns.AutoFit();
    // Add Pie Chart
    IShape pieChartshape = worksheet.Shapes.AddChart(ChartType.DoughnutExploded, 250,
20, 360, 230);

    // Adding series to SeriesCollection
    pieChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"], RowCol.Columns,
true, true);

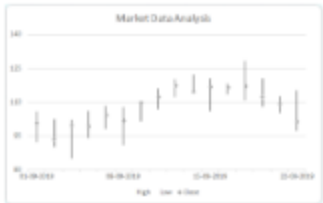



    // Configure Chart Title
    pieChartshape.Chart.ChartTitle.TextFrame.TextRange.Paragraphs.Add("Annual Sales
Record");

    // Saving workbook to Xlsx
    workbook.Save(@"22-PieChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Stock Chart

**Stock chart** is used to illustrate fluctuations in data over a time. It can represent fluctuations in stock, rainfall, or annual temperatures. The data arranged in columns or rows of a worksheet can be plotted in a Stock chart.

GcExcel supports the following types of Stock charts.

Chart Type	Chart Snapshot	Use Case
StockHLC		A high-low-close chart displays the data values organized in the order: high, low, close with the close value lying in between the high and low values.
StockOHLC		An open-high-low-close chart displays the data values organized in the order: open, high, low and close.
StockVHLC		A volume-high-low-close chart displays the data values organized in the order: volume, high, low and close.
StockVOHLC		A volume-open-high-low-close chart displays the data values organized in the order : volume, open, high, low and close.

## Using Code

Refer the following code to add StockVOHLC chart:

C#

```
public void StockCharts()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:D17"].Value = new object[,] {
    { null, "High", "Low", "Close" },
    { new DateTime(2019, 9, 1), 105.76, 92.38, 100.94 },
    { new DateTime(2019, 9, 2), 102.45, 90.14, 93.45 },
    { new DateTime(2019, 9, 3), 102.11, 85.01, 99.89 },
```

```
{ new DateTime(2019, 9, 4), 106.01, 94.04, 99.45 },
{ new DateTime(2019, 9, 5), 108.23, 98.16, 104.33 },
{ new DateTime(2019, 9, 8), 107.7, 91.02, 102.17 },
{ new DateTime(2019, 9, 9), 110.36, 101.62, 110.07 },
{ new DateTime(2019, 9, 10), 115.97, 106.89, 112.39 },
{ new DateTime(2019, 9, 11), 120.32, 112.15, 117.52 },
{ new DateTime(2019, 9, 12), 122.03, 114.67, 114.75 },
{ new DateTime(2019, 9, 15), 120.46, 106.21, 116.85 },
{ new DateTime(2019, 9, 16), 118.08, 113.55, 116.69 },
{ new DateTime(2019, 9, 17), 128.23, 110.91, 117.25 },
{ new DateTime(2019, 9, 18), 120.55, 108.09, 112.52 },
{ new DateTime(2019, 9, 19), 112.58, 105.42, 109.12 },
{ new DateTime(2019, 9, 22), 115.23, 97.25, 101.56 },

};

worksheet.Range["A:D"].Columns.AutoFit();

// Add Stock Chart
IShape stockChartshape = worksheet.Shapes.AddChart(ChartType.StockVOHLC, 350, 20,
360, 230);

// Adding series to SeriesCollection
stockChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:D17"],
RowCol.Columns);

// Configure Chart Title
stockChartshape.Chart.ChartTitle.Text = "Market Data Analysis";

// Configure value axis
IAxis valueAxis = stockChartshape.Chart.Axes.Item(AxisType.Value);
valueAxis.MinimumScale = 80;
valueAxis.MaximumScale = 140;
valueAxis.MajorUnit = 15;

// Configure category axis
IAxis categoryAxis = stockChartshape.Chart.Axes.Item(AxisType.Category);
categoryAxis.CategoryType = CategoryType.CategoryScale;
categoryAxis.MajorTickMark = TickMark.Outside;
categoryAxis.TickLabelSpacingIsAuto = false;
categoryAxis.TickLabelSpacing = 5;

// Configure Close Series Style
ISeries series_close = stockChartshape.Chart.SeriesCollection[2];
series_close.MarkerStyle = MarkerStyle.Diamond;
series_close.Has3DEffect = true;

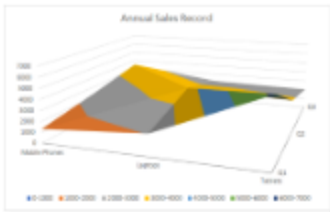



// Saving workbook to Xlsx
workbook.Save("23-StockChart.xlsx", SaveFileFormat.Xlsx);
```

}

## Surface Chart

**Surface charts** are useful when you want to find the optimum combinations between two data sets. As in a topographic map, the colors and patterns indicate the areas that are in the same range of values. A surface chart plots data on a three-dimensional surface, in a similar way that topographic maps plots elevation. The colors and patterns represent values within the same range. This chart type is especially useful for finding the optimum results when comparing two or more sets of data.

GcExcel supports the following types of Surface charts.

Chart Type	Chart Snapshot	Purpose
Surface		Surface chart is a chart with a 3-D visual effect.
SurfaceTopView		SurfaceTopView chart depicts surface chart viewed from above.
SurfaceTopViewWireframe		SurfaceTopViewWireframe chart depicts surface chart viewed from above with no fill color.
SurfaceWireframe		SurfaceWireframe chart depicts surface chart with a 3-D visual effect and no fill color.

### Using Code

Refer to the following code to add SurfaceWireframe chart.

C#

```
public void SurfaceCharts()
```

```
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:D4"].Value = new object[,]{
    {
        {null, "Q1", "Q2", "Q3"},
        {"Mobile Phones", 1330, 2345, 3493},
        {"Laptops", 2032, 3632, 2197},
        {"Tablets", 6233, 3270, 2030}
    };

    worksheet.Range["A:D"].Columns.AutoFit();
    // Add Surface Chart
    IShape surfaceChartShape = worksheet.Shapes.AddChart(ChartType.SurfaceWireframe,
250, 20, 360, 230);

    // Adding series to SeriesCollection
    surfaceChartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"],
RowCol.Columns, true, true);

    // Configure Chart Title
    surfaceChartShape.Chart.ChartTitle.TextFrame.TextRange.Paragraphs.Add("Annual
Sales Record");

    // Saving workbook to Xlsx
    workbook.Save(@"25-SurfaceChart.xlsx", SaveFileFormat.Xlsx);
}
```

## XY (Scatter) Chart

**Scatter chart** is used to illustrate relationships between individual items or categories. This chart is ideal for showing comparisons for scientific, statistical and engineering data. The data arranged in columns or rows of a worksheet can be plotted in a Scatter chart.

Unlike other charts, a scatter chart displays the actual values of the x and y variables in horizontal axis and vertical axis in the plot area. Typically, this chart combines the x and y values into single data points and displays them at irregular intervals. Also, this chart does not make use of the category axis because both horizontal axis (primary axis) and vertical axes (secondary axis) are value axes.

GcExcel supports the following types of Scatter charts.

Chart Type	Chart Snapshot	Use Case
XYScatter		A clustered Scatter chart displays the data points based on a selected data range. This helps the users to analyze and determine the relationship between x and y variables.
XYScatterLines		A scatter chart with straight lines displays a straight connecting line between data points in a particular series without showing the individual points.
XYScatterLinesNoMarkers		A scatter chart with straight lines and no data markers displays a smooth curve that connects all the data points in a particular series.
XYScatterSmooth		A scatter chart with smooth lines displays a connecting line between data points in a particular series without showing the individual points.
XYScatterSmoothNoMarkers		A scatter chart with smooth lines and no data markers displays a smooth curve that connects all the data points in a particular series.
Bubble		A bubble chart is ideal for financial data analysis. It displays the variations of a scatter chart where data points are replaced with bubbles and a third dimension is represented (Z axis) in the size of the bubbles. This chart plots z(size) values as well as x values and y values. Typically, this chart can be used when you want to plot three data series. The size of the bubbles is determined by the values in the third data series.
Bubble3DEffect		Bubble3DEffect chart can be used to display the chart demonstration in 3D, which is a modification of 2D Bubble chart. It does not have a third dimension, it only looks volumetric in appearance.

## Using Code

Refer to the following code to add a XY Scatter chart:

C#

```
public void ScatterCharts()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:C22"].Value = new object[,] {
    { "Index", "Zantedeschia", "Celosia" },
    { 0, 0, 0 }, {1, 30, 10 }, {2, 50, 21 }, {3, 90, 35 },
    {4, 67, 7 }, {5, 59, 24 }, {6, 28, 6 }, {7, 78, 50 },
    {8, 90, 20 }, {9, 100, 40 }, {10, 45, 6 }, {11, 87, 20 },
    {12, 21, 5 }, {13, 45, 25 }, {14, 68, 63 }, {15, 98, 40 },
    {16, 23, 6 }, {17, 89, 20 }, {18, 13, 6 }, {19, 90, 9 }, {20, 100, 80 }
    };
    worksheet.Range["A:C"].Columns.AutoFit();
    // Add XYScatter Chart
    IShape xyScatterChartshape = worksheet.Shapes.AddChart(ChartType.XYScatter, 250, 20,
    360, 230);

    // Adding series to SeriesCollection
    xyScatterChartshape.Chart.SeriesCollection.Add(worksheet.Range["B1:B22"],
    RowCol.Columns);
    xyScatterChartshape.Chart.SeriesCollection.Add(worksheet.Range["C1:C22"],
    RowCol.Columns);

    // Configure Chart Title
    xyScatterChartshape.Chart.ChartTitle.Text = "The Influence of Greenhouse Cultivation
on Different Flowers";

    // Configure Markers style
    ISeries series1 = xyScatterChartshape.Chart.SeriesCollection[0];
    series1.MarkerStyle = MarkerStyle.Diamond;
    series1.MarkerSize = 7;
    ISeries series2 = xyScatterChartshape.Chart.SeriesCollection[1];
    series2.MarkerStyle = MarkerStyle.Star;
    series2.MarkerSize = 7;

    // Saving workbook to Xlsx
    workbook.Save(@"26-ScatterChart.xlsx", SaveFileFormat.Xlsx);
}
```






}

## Radar Chart

A Radar chart is used to display circular visual representation of a 2-dimensional data. One can think of it as a circular XY chart. These charts represent each variable on a separate axis, which are arranged radially at equal distances from each other. Each of these axes share the same tick marks and scale. The data for each observation is plotted along these axis and then joined to form a polygon. Radar charts are generally used for analyzing performance or comparing values such as revenue and expense.

GcExcel supports the following types of radar charts.

Chart Type	Chart Snapshot	Use Case
Radar		Radar chart type can be used to represent multivariate data plotted in rows and columns in the graphical format.
RadarFilled		RadarFilled chart type can be used to display radar chart with areas highlighted by different colored regions for each value.
RadarMarkers		RadarMarkers chart can be used to display radar chart with markers representing data for each value along with areas highlighted by different line colors.

### Using Code

Refer to the following code to add RadarMarkers chart:

C#

```
public void RadarCharts()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
```

```

        worksheet.Range["A1:D7"].Value = new object[,]
    {
        {null, "Lisa", "Tim", "Jim"},
        {"Mathematics", 87, 64, 79},
        {"English", 79, 58, 78},
        {"History", 62, 70, 82},
        {"Biology", 85, 63, 54},
        {"Geography", 64, 85, 75},
        {"Zoology", 62, 79, 94}
    };

    worksheet.Range["A:D"].Columns.AutoFit();
    // Add Radar Chart
    IShape radarChartShape = worksheet.Shapes.AddChart(ChartType.RadarMarkers, 250,
    20, 360, 230);

    // Adding series to SeriesCollection
    radarChartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:D7"],
    RowCol.Columns, true, true);

    // Configure Chart Title
    radarChartShape.Chart.ChartTitle.TextFrame.TextRange.Paragraphs.Add("Test Score
    Analysis");

    // Saving workbook to Xlsx
    workbook.Save(@"27-RadarChart.xlsx", SaveFileFormat.Xlsx);
}

```

## Statistical Chart

**Statistical charts** can be used to present and interpret statistical data in graphical format. GcExcel supports statistical chart types like Box and Whisker, Histogram, Waterfall and Pareto. Such chart types add visual meaning to the represented data.

GcExcel supports the following types of Statistical chart types:


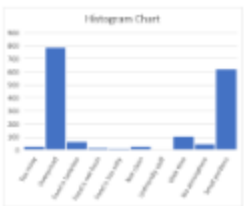
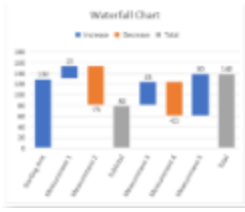
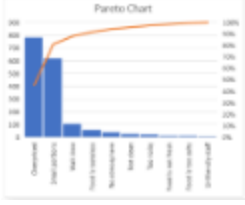
Chart Type	Chart Snapshot	Use Case
Box&Whisker		Box and Whisker charts are often used in Marketing Analysis, Statistical Analysis and General Analysis.
Histogram		Histogram is a common chart used in statistics. It can be used in scenarios, such as analysis of distribution/sales of books in a book store.

Chart Type	Chart Snapshot	Use Case
Waterfall Chart		Waterfall charts finds application in analyzing project gains including the number of contracts carried forwarded each year, contracts cancelled, tasks completed etc.
Pareto Chart		Pareto charts graphically summarize the process problems in ranking order from the most frequent to the least one.

## Box Whisker

**BoxWhisker** charts are statistical charts that display the distribution of numerical data through quartiles, means and outliers. As the name suggests, these values are represented using boxes and whiskers, where boxes show the range of quartiles (lower quartile, upper quartile and median), while whiskers indicate the variability outside the upper and lower quartiles. Any point outside the whiskers is said to be an outlier. These charts are useful for comparing distributions between many groups or data sets. For instance, you can easily display the variation in monthly temperature of two cities.

### Using Code

Refer to the following code to add Box and Whisker chart:

C#

```
public void BoxWhiskerChart()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:D16"].Value = new object[,]
    {
        {"Course", "SchoolA", "SchoolB", "SchoolC"},
        {"English", 78, 72, 45},
        {"Physics", 61, 55, 65},
        {"English", 63, 50, 65},
        {"Math", 62, 73, 83},
        {"English", 46, 64, 75},
        {"English", 58, 56, 67},
        {"Math", 60, 51, 67},
    }
}
```

```
        {"Math", 62, 53, 66},
        {"English", 63, 54, 64},
        {"English", 90, 52, 67},
        {"Physics", 70, 82, 64},
        {"English", 60, 56, 67},
        {"Math", 73, 56, 75},
        {"Math", 63, 58, 74},
        {"English", 73, 84, 45}
    };

    worksheet.Range["A:D"].Columns.AutoFit();
    //Add BoxWhisker chart
    IShape boxWhiskerChartshape = worksheet.Shapes.AddChart(ChartType.BoxWhisker,
300, 20, 300, 200);
    boxWhiskerChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:D16"]);

    // Configure Chart Title
    boxWhiskerChartshape.Chart.ChartTitle.Text = "Box & Whisker Chart";

    //Config value axis's scale
    IAxis value_axis = boxWhiskerChartshape.Chart.Axes.Item(AxisType.Value,
AxisGroup.Primary);
    value_axis.MinimumScale = 40;
    value_axis.MaximumScale = 70;

    //Configure the display of box&whisker plot
    ISeries series = boxWhiskerChartshape.Chart.SeriesCollection[0];
    series.ShowInnerPoints = true;
    series.ShowOutlierPoints = false;
    series.ShowMeanMarkers = false;
    series.ShowMeanLine = true;
    series.QuartileCalculationInclusiveMedian = true;

    // Saving workbook to Xlsx
    workbook.Save(@"28-BoxWhiskerChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Histogram

**Histograms** are visual representation of data distribution over a continuous interval or certain time period. These charts comprise vertical bars to indicate the frequency in each interval or bin created by dividing the raw data values into a series of consecutive and non-overlapping intervals. Hence, histograms help in estimating the range where maximum values fall as well as in knowing the extremes and gaps in data values, if there are any. For instance, histogram can help you find the range of height in which maximum students of a particular age group fall.

### Using Code

Refer to the following code to add a Histogram chart.

C#

```
public void HistogramChart()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:B11"].Value = new object[,]
    {
        {"Complaint", "Count"},
        {"Too noisy", 27},
        {"Overpriced", 789},
        {"Food is tasteless", 65},
        {"Food is not fresh", 19},
        {"Food is too salty", 15},
        {"Not clean", 30},
        {"Unfriendly staff", 12},
        {"Wait time", 109},
        {"No atmosphere", 45},
        {"Small portions", 621 }
    };

    worksheet.Range["A:B"].Columns.AutoFit();
    // Add Histogram Chart
    IShape histogramchartShape = worksheet.Shapes.AddChart(ChartType.Histogram, 300,
30, 300, 250);

    // Set range"A1:B11" as the histogram chart series
    histogramchartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:B11"]);

    // Sets bins type by category
    histogramchartShape.Chart.ChartGroups[0].BinsType =
BinsType.BinsTypeCategorical;

    // Configure Chart Title
    histogramchartShape.Chart.ChartTitle.Text = "Histogram Chart";

    // Saving workbook to Xlsx
    workbook.Save(@"29-HistogramChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Waterfall Chart

A **waterfall chart** shows the aggregate of values as they are added or subtracted. This type of chart is useful to understand how the initial value is affected by a series of positive and negative values. Waterfall charts can be used for

viewing fluctuations in product earnings, net income or profit analysis.

### Using Code

Refer the following code for adding Waterfall chart.

C#

```
public void WaterfallChart()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:B8"].Value = new object[,]
    {
        {"Starting Amt", 130},
        {"Measurement 1", 25},
        {"Measurement 2", -75},
        {"Subtotal", 80},
        {"Measurement 3", 45},
        {"Measurement 4", -65},
        {"Measurement 5", 80},
        {"Total", 140}
    };

    worksheet.Range["A:A"].Columns.AutoFit();

    // Add Waterfall Chart
    IShape waterfallChartShape = worksheet.Shapes.AddChart(ChartType.Waterfall, 300,
20, 300, 250);
    waterfallChartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:B8"]);

    //Set subtotal & total points
    IPoints points = waterfallChartShape.Chart.SeriesCollection[0].Points;
    points[3].IsTotal = true;
    points[7].IsTotal = true;

    //Connector lines are not shown
    ISeries series = waterfallChartShape.Chart.SeriesCollection[0];
    series.ShowConnectorLines = false;

    // Configure Chart Title
    waterfallChartShape.Chart.ChartTitle.Text = "Waterfall Chart";

    // Saving workbook to Xlsx
    workbook.Save(@"30-WaterfallChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Pareto Chart

GcExcel supports Pareto chart, also known as Pareto distribution diagram. It is a vertical bar graph in which values are plotted left to right, in decreasing order of relative frequency. Pareto charts are useful for task prioritizing. The chart gives a hint about the variables that have the greatest effect on a given system.

Pareto chart can be used to highlight the most important factor from a given set of factors. For example, quality control, inventory control, and customer grievance handling are some areas where Pareto chart analysis can be used.

### Using code

Refer the following code to add Pareto chart:

C#

```
public void ParetoChart()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:B11"].Value = new object[,]
    {
        {"Complaint", "Count"},
        {"Too noisy", 27},
        {"Overpriced", 789},
        {"Food is tasteless", 65},
        {"Food is not fresh", 19},
        {"Food is too salty", 15},
        {"Not clean", 30},
        {"Unfriendly staff", 12},
        {"Wait time", 109},
        {"No atmosphere", 45},
        {"Small portions", 621 }
    };

    worksheet.Range["A:B"].Columns.AutoFit();
    // Add Pareto Chart
    IShape paretochartShape = worksheet.Shapes.AddChart(ChartType.Pareto, 300, 30,
300, 250);




    // Set range"A1:B11" as the pareto chart series
    paretochartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:B11"]);

    // Configure Chart Title
    paretochartShape.Chart.ChartTitle.Text = "Pareto Chart";
}
```

```
// Saving workbook to Xlsx
workbook.Save(@"31-ParetoChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Specialized Chart

GcExcel supports the following types of Specialized chart types such as Sunburst, Treemap and Funnel. The following table covers the different specialized chart types, chart snapshots and their use-cases.

Chart Type	Chart Snapshot	Use Case
Sunburst		Sunburst charts can be used to break down data into different entities for identifying and visualizing multilevel parent child relationships in different business scenarios quickly and efficiently.
Treemap		Treemap charts can be used to display large amount of hierarchical data without any space constraints. You can plot more than tens of thousands of data points.
Funnel		Funnel charts help in visualizing the sequential stages in a linear process such as recruitment process, order fulfillment cycles and promotional campaigns.

## Sunburst

**Sunburst**, also known as a multi-level pie chart, is ideal for visualizing multi-level hierarchical data depicted by concentric circles. The circle in the center represents the root node surrounded by the rings representing different levels of hierarchy. Rings are divided based on their relationship with the parent slice with each of them either divided equally or proportional to a value. This type of chart helps users in breaking down data into different entities for identifying and visualizing multilevel parent child relationships in different business scenarios quickly and efficiently.

### Using Code

Refer to the following code to add a Sunburst chart:

```
C#
```



```
public void SunburstChart()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:D16"].Value = new object[, ]
    {
        {"Region", "Subregion", "Country", "Population"},
        {"Asia", "Southern", "India", 1354051854},
        {null, null, "Pakistan", 200813818},
        {null, null, "Bangladesh", 166368149},
        {null, null, "Others", 170220300},
        {null, "Eastern", "China", 1415045928},
        {null, null, "Japan", 127185332},
        {null, null, "Others", 111652273},
        {null, "South-Eastern", null, 655636576},
        {null, "Western", null, 272298399},
        {null, "Central", null, 71860465},
        {"Africa", "Eastern", null, 433643132},
        {null, "Western", null, 381980688},
        {null, "Northern", null, 237784677},
        {null, "Others", null, 234512021},
        {"Europe", null, null, 742648010},
        {"Others", null, null, 1057117703}
    };

    worksheet.Range["A:D"].Columns.AutoFit();
    // Add Sunburst Chart
    IShape sunburstChartShape = worksheet.Shapes.AddChart(ChartType.Sunburst, 250, 20,
    360, 330);

    // Adding series to SeriesCollection
    sunburstChartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:D16"],
    RowCol.Columns, true, true);

    // Configure Chart Title
    sunburstChartShape.Chart.ChartTitle.Text = "World Population";

    // Saving workbook to Xlsx
    workbook.Save(@"32-SunburstChart.xlsx", SaveFileFormat.Xlsx);
}
```

## TreeMap

**TreeMap** is a chart type used to display hierarchical data as a set of nested rectangles. Treemap charts are used to

represent hierarchical data in a tree-like structure. Data, organized as branches and sub-branches, is depicted with the help of rectangles. With Treemap charts, you can easily drill down huge data to an unlimited number of levels.

### Using Code

Refer to the following code to add Treemap chart:

C#

```
public void TreemapChart()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:D16"].Value = new object[,]
    {
        {"Region", "Subregion", "Country", "Population"},
        {"Asia", "Southern", "India", 1354051854},
        {null, null, "Pakistan", 200813818},
        {null, null, "Bangladesh", 166368149},
        {null, null, "Others", 170220300},
        {null, "Eastern", "China", 1415045928},
        {null, null, "Japan", 127185332},
        {null, null, "Others", 111652273},
        {null, "South-Eastern", null, 655636576},
        {null, "Western", null, 272298399},
        {null, "Central", null, 71860465},
        {"Africa", "Eastern", null, 433643132},
        {null, "Western", null, 381980688},
        {null, "Northern", null, 237784677},
        {null, "Others", null, 234512021},
        {"Europe", null, null, 742648010},
        {"Others", null, null, 1057117703}
    };

    worksheet.Range["A:D"].Columns.AutoFit();
    // Add Treemap Chart
    IShape treeMapChartShape = worksheet.Shapes.AddChart(ChartType.Treemap, 250, 20,
    360, 330);

    // Adding series to SeriesCollection
    treeMapChartShape.Chart.SeriesCollection.Add(worksheet.Range["A1:D16"],
    RowCol.Columns, true, true);

    // Configure Chart Title
    treeMapChartShape.Chart.ChartTitle.Text = "World Population";
}
```

```
// Saving workbook to Xlsx
workbook.Save(@"33-TreemapChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Funnel

**Funnel** charts help in visualizing sequential stages in a linear process such as order fulfillment. In such processes, each stage represents a proportion (percentage) of the total. Therefore, the chart takes the funnel shape with the first stage being the largest and each subsequent stage smaller than the predecessor. The Funnel charts can be used to represent stages in a sales process and represent the amount of potential revenue for each stage. This type of chart is useful in finding potential problem areas in an organization's sales processes. For instance, with Funnel charts, a user can plot the order fulfillment process that tracks number of orders getting across a stage.

### Using Code

Refer to the following code to add a Funnel chart:

C#

```
public void FunnelChart()
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // Prepare data for chart
    worksheet.Range["A1:B9"].Value = new object[,]
    {
        {null, "Sales"},
        {"Consultation", 140000},
        {"Prospects", 120000},
        {"Qualified", 100000},
        {"Negotiations", 80000},
        {"Prototype", 60000},
        {"Closing", 40000},
        {"Won", 20000},
        {"Finalized", 10000}
    };

    worksheet.Range["A:B"].Columns.AutoFit();

    // Add Funnel Chart
    IShape funnelChartshape = worksheet.Shapes.AddChart(ChartType.Funnel, 300, 20,
    300, 200);
    funnelChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:B9"]);

    // Configure Chart Title
    funnelChartshape.Chart.ChartTitle.Text = "Sales Pipeline";
}
```

```
// Configure Axis
IAxis axis = funnelChartshape.Chart.Axes.Item(AxisType.Category,
AxisGroup.Primary);
axis.Visible = true;


// Saving workbook to Xlsx
workbook.Save(@"34-FunnelChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Chart Sheet

Sometimes, users find it difficult to accommodate both data and charts in the same worksheet. For this reason, GcExcel now lets users add the chart to a separate sheet, called the 'Chart sheet'. Unlike Worksheets, Chart sheets can contain only the chart. This helps avoid the usual clutter of data and embedded charts in the same worksheet. Also, using chart sheets, users will be able to read the chart in detail and change the sheet page orientation while printing.

A Chart sheet can be created in a Workbook using the **IWorksheets.Add(SheetType.Chart)** method. Further, you can add a chart to the Chart sheet by using **IShapes.AddChart** method. The user may note that each chart sheet should have a chart, else it can throw an exception while saving the file.

The methods and properties associated with chart sheets in GcExcel are listed in the table below:

Methods/Properties	Description
<b>Add(SheetType.Chart)</b>	The <b>Add</b> method in <b>IWorksheets</b> interface has an overload with ' <b>SheetType</b> '. Hence, for adding a Chart sheet, you need to use <a href="#">SheetType.Chart</a> .
<b>AddChart</b>	<p>The <b>AddChart</b> method in <b>IShapes</b> interface adds a chart for the Chart sheet.</p> <p> <b>Note:</b> Each Chart sheet should have a chart. Otherwise, it will throw an exception while saving the file.</p>
<b>AddShape</b>	The <b>AddShape</b> method in <b>IChart</b> interface adds multiple shapes for the Chart sheet. Supported shapes are chart, picture, connector etc. In this case, the first chart is the main Chart, and the other shapes will be discarded when saving the file.
<b>SheetType</b>	The <b>SheetType</b> Property in <b>IWorksheet</b> interface gets the type of current sheet (Worksheet or Chart sheet).
<b>Delete</b>	The <b>Delete</b> method in <b>IShape</b> interface deletes the Chart from the Chart sheet, or deletes the shape from the Chart.
<b>Copy</b>	The <b>Copy</b> method in <b>IWorksheet</b> interface copies a new Chart sheet.
<b>Move</b>	The <b>Move</b> method in <b>IWorksheet</b> interface moves the chart sheet to a new location in the current workbook or a new workbook.

The following sections discuss in detail about chart sheet operations in a workbook.

### Add Chart Sheet

To add a chart sheet, refer the following code:

```
C#  
  
public Workbook AddChartSheet()  
{  
    Workbook workbook = new Workbook();  
    IWorksheet worksheet = workbook.Worksheets[0];  
  
    worksheet.Range["A1:E5"].Value = new object[,]  
    {  
        {"Region", "Q1", "Q2", "Q3", "Q4"},  
        {"North", 100, 300, 200, 600},  
        {"East", 400, 200, 500, 800},  
        {"South", 300, 500, 100, 400},  
        {"West", 400, 200, 600, 100},  
    };  
  
    //Add a Chart Sheet  
    IWorksheet chartSheet = workbook.Worksheets.Add(SheetType.Chart);  
  
    //Add the main chart for the chart sheet  
    IShape mainChart = chartSheet.Shapes.AddChart(ChartType.ColumnClustered, 100, 100,  
200, 200);  
    mainChart.Chart.ChartTitle.Text = "Sales 2018-2019";  
    mainChart.Chart.SeriesCollection.Add(worksheet.Range["A1:E5"]);  
  
    //Add a user shape for the main chart.  
    IShape shape = mainChart.Chart.AddShape(AutoShapeType.Rectangle, 50, 20, 100, 100);  
    shape.TextFrame.TextRange.Add("This chart displays the regional quarterly sales for  
the year 2018-2019");  
  
    //Save Workbook  
    workbook.Save("Chartsheet.xlsx");  
  
    return workbook;  
}
```

### Copy and Move Chart Sheet

To copy and move a chart sheet, refer the following example code:

```
C#  
  
public void CopyMoveChartSheet()  
{  
    Workbook workbook = AddChartSheet();  
  
    //Add additional worksheets
```

```
workbook.Worksheets.Add(SheetType.Worksheet);
workbook.Worksheets.Add(SheetType.Worksheet);

//Access ChartSheet
IWorksheet chartSheet = workbook.Worksheets[1];

//Copies the chart sheet to the end of the workbook and save it.
chartSheet.Copy();
workbook.Save("CopyChartsheet.xlsx");

//Moves the chart sheet to the end of the workbook and save it.
chartSheet.Move();
workbook.Save("MoveChartsheet.xlsx");
}
```

## Delete Chart Sheet

To delete a chart sheet, refer the following example code:

```
C#
public void DeleteChartSheet()
{
    Workbook workbook = AddChartSheet();

    //Access ChartSheet
    IWorksheet chartSheet = workbook.Worksheets[1];

    //Deletes the chart sheet
    chartSheet.Delete();

    //Save Workbook
    workbook.Save("NoChartsheet.xlsx");
}
```

## 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. Typically, a table consists of rows and columns that can be formatted and managed independently in a worksheet.

In GcExcel .NET, you can use table in the following ways:

- [Create and Delete Tables](#)
- [Modify Tables](#)
- [Table Sort](#)
- [Table Filters](#)

- [Add and Delete Table Columns and Rows](#)
- [Table Style](#)

## Create and Delete Tables

In GcExcel .NET, you can create and delete tables in spreadsheets using the **Add method** of the **ITables interface** and the **Delete Method** of the **ITable Interface**, 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**.

- **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** 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**, **DataRange property** and **TotalsRange property** 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 of the **ITableColumn** interface.

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;
```

## 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** of the **ITable interface**. The **Apply** 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();
```

## 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** of the **IRange interface** 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** - Represents the table columns collection.
- **ITableRows Interface** - Represents the table rows collection.
- **ITableColumn Interface** - Represents an individual table column.
- **ITableRow Interface** - Represents an individual table row.

### Add and Delete Single Column

To add and delete a table column, you can use the **Add method** of the **ITableColumns interface** and the **Delete method** of the **ITableColumn interface** respectively.

Refer to the following example code in order to add and delete a table column.

```
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 Multiple Columns

To add and delete multiple columns, you can use the **Add** and **Delete** methods of **ITableColumns** interface. These methods take the position of column and count of columns to be added or deleted as parameters.

Refer to the following example code in order to add and delete table columns.

```
C#

//add table
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F7"], true);

//add two columns before first column
table.Columns.Add(0, 2);

//delete three columns after second column
table.Columns.Delete(1, 3);
```

### Add and Delete Single Row

To add and delete a table row, you can use the **Add method** of the **ITableRows** interface and the **Delete method** of the **ITableRow** interface respectively.

Refer to the following example code in order to add and delete a table row.

```
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();
```

### Add and Delete Multiple Rows

To add and delete multiple rows, you can use the **Add** and **Delete** methods of **ITableRows** interface. These methods take the position of row and count of rows to be added or deleted as parameters.

Refer to the following example code in order to add and delete table rows.

```
C#  
  
//add table  
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F7"], true);  
  
//insert three rows after last row  
table.Rows.Add(-1, 3);  
  
//delete last table row  
table.Rows.Delete(table.Rows.Count - 1, 1);
```

## Table Style

In GcExcel .NET, you can create custom table style elements and apply them to your worksheet using the **ITableStyle Interface**. Also, you can format a table using any of the predefined table styles provided by GcExcel .NET.

Typically, each workbook possesses an **ITableStyle collection** 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** of the **ITables interface**, 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 =
BorderLineStyle.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 =
BorderLineStyle.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 =
BorderLineStyle.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;
```

## 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](#)
- [Pivot Table Style](#)

## 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** of the **IPivotCaches interface** to create a new pivot cache. After creating pivot cache, the next step is to create the new pivot table using **CreatePivotTable** method of the **IPivotCache interface**.

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

You can configure the fields of your pivot table using the properties and methods of the **IPivotCaches interface** and **IPivotTables interface**.

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;
```

### Manage 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;
```

### Manage Grand Total Visibility Settings

The Grand total in pivot table helps in analyzing the total sum of the data in the pivot table. You can display or hide the grand total for the row or column field by setting the visibility of **ColumnGrand** and **RowGrand** properties of the **IPivotTable** interface. These properties take boolean values and are set to true by default. For example, if you want to display the grand total only for rows, then set the RowGrand property to true and ColumnGrand to false.

Refer to the following example code to manage the visibility settings of the grand total field.


```
C#

// Set the PivotTable report to show grand totals for columns & rows
pivottable.ColumnGrand = true;
pivottable.RowGrand = true;
```

### Change Row Axis Layout

The display of pivot table can be changed to any desired layout using the **LayoutRowType** enumeration. The following options are provided by this enumeration:

- **CompactRow** (default layout)
- **OutlineRow**
- **TabularRow**

 **Note:** The **SubtotalLocationType** enumeration can only be set to **Bottom** if the **LayoutRowType** is set to **TabularRow**.

Refer to the following example code to set the row axis layout of the pivot table to TabularRow.

```
C#

// Set the PivotTable LayoutRowType to Tabular Row
pivottable.SetRowAxisLayout(LayoutRowType.TabularRow);
```

### Change Pivot Table Layout

The different layouts of a pivot table makes it more flexible and convenient to analyse its data. GcExcel supports the following pivot table layouts:

- Compact form
- Outline form
- Tabular form

In addition to these, you can also choose to insert blank rows, set the position of subtotals, show all items or to repeat any item in the pivot table layouts.

Refer to the following example code to set the layout of pivot table and additional options.

```
C#  
  
//set pivot table layout  
field_Category.LayoutForm = LayoutFormType.Tabular;  
field_Category.LayoutBlankLine = true;  
  
field_Country.LayoutForm = LayoutFormType.Outline;  
field_Country.LayoutCompactRow = false;  
  
//set subtotal location  
field_Country.LayoutSubtotalLocation = SubtotalLocationType.Bottom;  
field_Country.ShowAllItems = true;
```

### Rename Pivot Table Fields

Sometimes, the pivot table fields are not easily comprehensible and hence can be renamed to meaningful and easily understandable names.

Refer to the following example code to rename the pivot table fields.

```
C#  
  
//config pivot table's fields  
var field_Date = pivottable.PivotFields["Date"];  
field_Date.Orientation = PivotFieldOrientation.PageField;  
  
// Renaming PivotField "Category" to "Type of Category"  
var field_Category = pivottable.PivotFields["Category"];  
field_Category.Name = "Type of 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.RowField;  
  
// Renaming DataField "Sum of Amount" to "Amount Total"  
pivottable.DataFields[0].Name = "Amount Total";
```

### 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 Different Calculations on a Pivot Field

In GcExcel, you can add a pivot table field to a pivot table multiple times by applying various calculation functions on it. These functions include sum, average, min, max, count etc. The final pivot table output will contain multiple data fields based on the calculations applied over the pivot table field.

Refer to the following example code to add a pivot table field as multiple data fields by applying different calculation functions.

```
C#
//config pivot table's fields
var field_Category = pivottable.PivotFields["Category"];
field_Category.Orientation = PivotFieldOrientation.RowField;

var field_Product = pivottable.PivotFields["Product"];
field_Product.Orientation = PivotFieldOrientation.RowField;

//sum function on Amount field
var field_Amount = pivottable.PivotFields["Amount"];
pivottable.AddDataField(field_Amount, "sum amount", ConsolidationFunction.Sum);

//count function on Amount field
var field_Amount2 = pivottable.PivotFields["Amount"];
pivottable.AddDataField(field_Amount2, "count amount", ConsolidationFunction.Count);
```

The output of above example code when viewed in Excel, looks like below:

Row Labels	sum amount	count amount
<b>Fruit</b>	<b>44561</b>	<b>8</b>
Apple	16794	3
Banana	24157	4
Orange	3610	1
<b>Vegetables</b>	<b>35936</b>	<b>7</b>
Beans	2626	1
Broccoli	27137	4
Carrots	6173	2
<b>Grand Total</b>	<b>80497</b>	<b>15</b>

count amount  
Value: 15  
Row: Grand Total  
Column: count amount

### Defer Layout Update

In case of huge amount of data, the performance of a pivot table might get affected while updating its layout by adding or moving fields in the different areas of a pivot table.

GcExcel provides **DeferLayoutUpdate** property which improves the performance of a pivot table by deferring its layout updates. When set to true, the pivot table is recalculated only after all the fields are added or moved instead of getting recalculated after each change. You can choose to update the pivot table output after making all the changes by calling the **Update** method.

Refer to the following example code to defer layout updates to a pivot table.

```
C#
//defer layout update
pivottable.DeferLayoutUpdate = true;

//config pivot table's 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;

//must update the pivottable
pivottable.Update();
```

### Use Pivot Table Options

GcExcel supports the following layout and formatting options in a pivot table:

- Merging cells with outer-row item, column item, subtotal and grand total labels
- Indentation of Pivot table items when compact row layout form is set
- Ordering page fields in pivot table layout. It can be either **DownThenOver** (default value) or **OverThenDown**.
- Defining number of page fields in each column or row in the pivot table output
- Displaying custom string in cells which contain errors
- Displaying custom string in cells which contain null values

Refer to the following example code to set various layout and format options in a pivot table.

```
C#

//set layout and format options
pivottable.PageFieldOrder = Order.OverThenDown;
pivottable.PageFieldWrapCount = 2;

pivottable.CompactRowIndent = 2;

pivottable.ErrorString = "Error";
pivottable.NullString = "Empty";

pivottable.DisplayErrorString = true;
pivottable.DisplayNullString = true;
```

### Sort Pivot Table Fields

GcExcel supports sorting data fields in a pivot table by using **AutoSort** method and defining ascending or descending as its sort order.

You can also retrieve the name of data field used to sort the specified PivotTable field by using **AutoSortField** property and its sorting order by using **AutoSortOrder** property. The position of an item in its field can also be set or retrieved by using the **Position** property of **IPivotItem** interface.

Refer to the following example code to sort 'Product' field in a pivot table.

```
C#

//sort the product items
field_Product.AutoSort(SortOrder.Descending);
```

### Retrieve Pivot Table Ranges

The structure of a pivot table report is comprised of different ranges. In order to retrieve a specific range of pivot table, it is important to understand the structure of a pivot table.

Row Labels	2018	2019	Grand Total
	Jan	Jan	
<b>Consumer Electronics</b>	\$28,515.00	\$10,904.00	\$39,419.00
Bose 785593-0050	\$4,270.00	\$1,903.00	\$6,173.00
Canon EOS 1500D	\$11,063.00		\$11,063.00
Haier 394L 4Star	\$6,946.00	\$617.00	\$7,563.00
IFB 6.5 Kg FullyAuto		\$8,384.00	\$8,384.00
Mi LED 40inch	\$2,626.00		\$2,626.00
Sennheiser HD 4.40-BT	\$3,610.00		\$3,610.00
<b>Mobile</b>	\$32,016.00	\$9,062.00	\$41,078.00
Iphone XR		\$9,062.00	\$9,062.00
OnePlus 7Pro	\$15,156.00		\$15,156.00
Redmi 7	\$9,429.00		\$9,429.00
Samsung S9	\$7,431.00		\$7,431.00
<b>Grand Total</b>	\$60,531.00	\$19,966.00	\$80,497.00

As can be observed from the above screenshot, the structure of a pivot table can be explained as:

- **PivotRowAxis:** The row axis area of a pivot table contains fields which group the table's data by rows
- **PivotColumnAxis:** The column axis area of a pivot table contains fields which break the table's data into different categories by columns.
- **Pivot Cell:** Any cell in a pivot table
- **Row PivotLine:** Any row in the row axis area of a pivot table
- **Column PivotLine:** Any column in the column axis area of a pivot table

GcExcel provides API to retrieve the detailed ranges of a pivot table to apply any operation or style on them to make the result more readable and distinguishable. Detailed pivot table ranges which can be retrieved are:


- Different types of pivot cells like subtotals, grand totals, data fields, pivot fields, values, blank cells
- Different types of pivot lines like subtotal, grand total, regular or blank line
- Entire row or column axis
- Whole page area
- Entire pivot table report including page fields
- A value in any range of pivot table
- The position of any element or pivot line

Refer to the following example code to get a specific range and set its style in a pivot table report.

```
C#
//get detail range and set style
foreach (var item in pivottable.PivotRowAxis.PivotLines)
{
    if (item.LineType == PivotLineType.Subtotal)
    {
        item.PivotLineCells[0].Range.Interior.Color = Color.GreenYellow;
    }
}
```

The output of above code example when viewed in Excel, looks like below:


Row Labels	Sum of Amount
<b>Consumer Electronics</b>	
Bose 785593-0050	\$6,173.00
Canon EOS 1500D	\$11,063.00
Haier 394L4Star	\$7,563.00
IFB 6.5 Kg FullyAuto	\$8,384.00
Mi LED 40inch	\$2,626.00
Sennheiser HD 4.40-BT	\$3,610.00
<b>Consumer Electronics Total</b>	<b>\$39,419.00</b>
<b>Mobile</b>	
Iphone XR	\$9,062.00
OnePlus 7Pro	\$15,156.00
Redmi 7	\$9,429.00
Samsung S9	\$7,431.00
<b>Mobile Total</b>	<b>\$41,078.00</b>
<b>Grand Total</b>	<b>\$80,497.00</b>

 **Note:** Style applied to a pivot table is lost if the pivot table is changed in any way.

### Set Conditional Formatting

Refer to the following example code to set conditional formatting in last row of a pivot table report by setting cell color when the values are above average.

```
C#
// set conditional format to the last row
int rowCount = pivottable.DataBodyRange.RowCount;
IAboveAverage averageCondition = pivottable.DataBodyRange.Rows[rowCount - 1].FormatConditions.AddAboveAverage();
averageCondition.AboveBelow = AboveBelow.AboveAverage;
averageCondition.Interior.Color = Color.Pink;
```

 **Note:** Conditional formatting applied to a pivot table is lost if the pivot table is changed in any way.

### Disable Automatic Grouping of Date/Time Columns

The Date/Time columns in a pivot table are grouped together by default. GcExcel allows you to disable this grouping by setting **AutomaticGroupDateTimeInPivotTable** property to false before creating the pivot cache while creating a pivot table.

When AutomaticGroupDateTimeInPivotTable = False	When AutomaticGroupDateTimeInPivotTable = True (default)
Sum of Amount Column Labels Row Labels 06-01-2011 07-01-2011 11-01-2011 08-06-2011 10-01-2012	Sum of Amount Column Labels +2011 +2012 +2013 Grand Total Row Labels

Refer to the following example code to disable automatic grouping of date/time columns.

```
C#
// Set false to group date/time fields in PivotTable automatically
workbook.Options.Data.AutomaticGroupDateTimeInPivotTable = false;
```

## Pivot Table Style

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** property or use the **TableStyle** property of the **IPivotTable** interface.

The image shared below depicts a pivot table with built-in style.

Date	(All)						
Sum of Amount	Column Labels						
Row Labels	Apple	Banana	Beans	Broccoli	Carrots	Orange	Grand Total
<b>Fruit</b>	\$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
<b>Vegetables</b>			\$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
<b>Grand Total</b>	\$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** 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** property of the **IPivotTable** 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
Fruit	\$9,848.00	\$24,157.00				\$3,610.00
Canada	\$7,431.00	\$8,384.00				
France	\$2,417.00					
Germany		\$8,250.00				
New Zealand		\$6,906.00				
United States		\$617.00				\$3,610.00
Vegetables			\$2,626.00	\$24,313.00	\$6,173.00	
Australia				\$9,062.00		
Germany			\$2,626.00		\$1,903.00	
United Kingdom				\$8,239.00		
United States				\$7,012.00	\$4,270.00	
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);
```



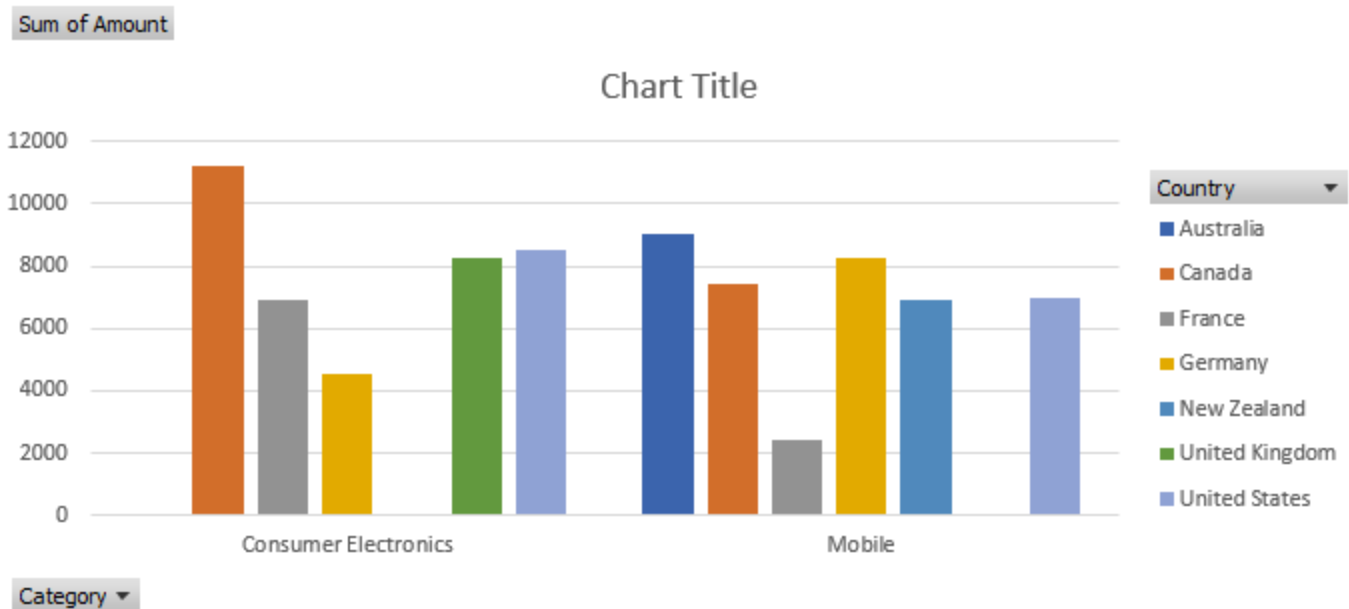
**Note:** While applying custom styles to the pivot table, it is important to note that if users apply a TableStyle whose **ShowAsAvailablePivotStyle** property is set to false, then the InvalidOperationException is thrown.

## Pivot Chart

Pivot chart represents the data of associated pivot table in a chart. Like a normal chart, the pivot chart displays data series, categories, legends, data markers and axes. You can change the titles, legend placement, data labels, chart location etc.

A pivot chart is interactive as it reflects the changes made in its associated pivot table. The pivot table fields are displayed on a pivot chart as buttons. You can configure whether to display the legend, axis, value field buttons or expanding or collapsing entire field buttons by using the **PivotOptions** property. When a field button is clicked, its filter pane appears. It helps you to sort and filter pivot chart's underlying data.

Excel files with pivot charts can be loaded, modified and saved back to Excel. The below image displays a pivot chart with legend, axis and value field buttons.



## Create Pivot Chart

The below mentioned steps explain how to create a pivot chart:

1. Create a pivot table.
2. Add a normal chart by using **AddChart** method of IShapes interface.
3. Use **SetSourceData** method of IChart interface to turn a normal chart into a PivotChart by providing the source range inside the pivot table's range.

C#

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

object[,] sourceData = new object[,] {
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Bose 785593-0050", "Consumer Electronics", 4270, new
DateTime(2018, 1, 6), "United States" },
    { 2, "Canon EOS 1500D", "Consumer Electronics", 8239, new
DateTime(2018, 1, 7), "United Kingdom" },
    { 3, "Haier 394L 4Star", "Consumer Electronics", 617, new
DateTime(2018, 1, 8), "United States" },
```

```

{ 4,          "IFB 6.5 Kg FullyAuto",  "Consumer Electronics",  8384,    new
DateTime(2018, 1, 10), "Canada" },
{ 5,          "Mi LED 40inch",          "Consumer Electronics",  2626,    new
DateTime(2018, 1, 10), "Germany" },
{ 6,          "Sennheiser HD 4.40-BT",  "Consumer Electronics",  3610,    new
DateTime(2018, 1, 11), "United States" },
{ 7,          "Iphone XR",              "Mobile",                9062,    new
DateTime(2018, 1, 11), "Australia" },
{ 8,          "OnePlus 7Pro",            "Mobile",                6906,    new
DateTime(2018, 1, 16), "New Zealand" },
{ 9,          "Redmi 7",                "Mobile",                2417,    new
DateTime(2018, 1, 16), "France" },
{ 10,         "Samsung S9",             "Mobile",                7431,    new
DateTime(2018, 1, 16), "Canada" },
{ 11,         "OnePlus 7Pro",            "Mobile",                8250,    new
DateTime(2018, 1, 16), "Germany" },
{ 12,         "Redmi 7",                "Mobile",                7012,    new
DateTime(2018, 1, 18), "United States" },
{ 13,         "Bose 785593-0050",        "Consumer Electronics",  1903,    new
DateTime(2018, 1, 20), "Germany" },
{ 14,         "Canon EOS 1500D",         "Consumer Electronics",  2824,    new
DateTime(2018, 1, 22), "Canada" },
{ 15,         "Haier 394L 4Star",        "Consumer Electronics",  6946,    new
DateTime(2018, 1, 24), "France" },
};

IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A6:F21"].Value = sourceData;
worksheet.Range["D6:D21"].NumberFormat = "$#,##0.00";
// Create pivot cache
var pivotcache = workbook.PivotCaches.Create(worksheet.Range["A6:F21"]);
// Create pivot table
var pivottable = worksheet.PivotTables.Add(pivotcache, worksheet.Range["A1"],
"pivottable1");

//config pivot table's fields
pivottable.PivotFields["Category"].Orientation = PivotFieldOrientation.RowField;
pivottable.PivotFields["Country"].Orientation = PivotFieldOrientation.ColumnField;
pivottable.PivotFields["Amount"].Orientation = PivotFieldOrientation.DataField;


worksheet.Range["A:I"].AutoFit();

// Add a column chart
IChart chart = worksheet.Shapes.AddChartInPixel(ChartType.ColumnClustered, 0, 100, 689,
320).Chart;

// Set data source(use pivot table range).
chart.SetSourceData(pivottable.TableRange1);

```

```
//save to an excel file  
workbook.Save("createpivotchart.xlsx");
```

 **Note:** To turn a normal chart into a pivot chart, add any chart from the ones listed below. A **NotSupportedException** will be thrown if any other chart is added.

- Area
- Bar
- Column
- Pie/Doughnut
- Line
- Radar
- Surface

### Configure Pivot Chart's Buttons

Refer to the following example code to configure pivot chart's buttons.

C#

```
chart.PivotOptions.ShowLegendFieldButtons = false;  
chart.PivotOptions.ShowAxisFieldButtons = false;  
  
// Set legend position to bottom  
chart.Legend.Position = LegendPosition.Bottom;
```

### Update Pivot Table to Reflect in Pivot Chart

Refer to the following example code to update pivot table to reflect in pivot chart.

C#

```
// Drag row field to hidden  
chart.PivotTable.RowFields[0].Orientation = PivotFieldOrientation.Hidden;
```

### Convert Pivot Chart to Normal Chart

Refer to the following example code to convert pivot chart to normal chart.

C#

```
// Clear pivot table to turn a PivotChart into a normal chart.  
pivottable.TableRange2.Clear();
```

### Limitations

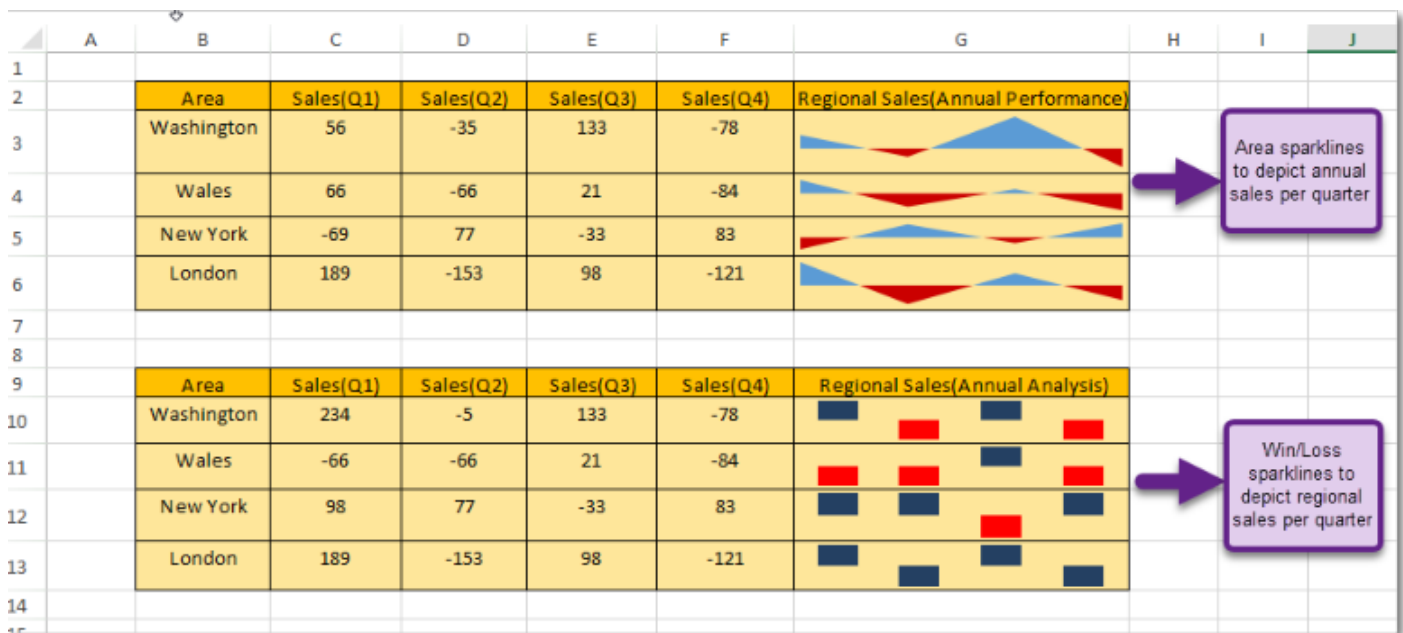
- The pivot chart is exported as a normal chart in PDF or while doing JSON I/O.
- If you add, change or delete the source range of a series, it will not reflect in pivot chart.

## 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. These 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** of the **ISparklineGroups** interface.

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** 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** 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 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 of the **ISparklineGroup** interface.

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;
```

## 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](#)
- [Slicer Style](#)
- [Export Slicers](#)

## Add Slicer in Table

In GcExcel .NET, you can use slicer in a table by accessing the properties and methods of the **ISlicer** interface, **ISlicerCache** interface, and **ISlicerCaches** interface.

To add slicer in your table, you need to first invoke the **Add method** 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**, **IPivotCaches Interface**, **IPivotField Interface**, **IPivotFields Interface**, **IPivotTable**

**Interface, IPivotTables Interface, IPivotItem Interface.**

To add slicer in a pivot table, you need to first invoke the **Add method** 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 pivotcache = workbook.PivotCaches.Create(worksheet.Range["A1:F16"]);
IPivotTable pivottable1 = worksheet.PivotTables.Add(pivotcache, worksheet.Range["K5"],
"pivottable1");
IPivotTable pivottable2 = worksheet.PivotTables.Add(pivotcache, 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();
```

## 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** of the **ISlicer interface**.

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** and **DisplayHeader property** of the **ISlicer interface**.

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;
```

## Print Settings

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.

- [Configure Page Header and Footer](#)
- [Configure Page Settings](#)
- [Configure Page Breaks](#)
- [Configure Paper Settings](#)
- [Configure Print Area](#)
- [Configure Columns to Repeat at Left and Right](#)
- [Configure Rows to Repeat at Top and Bottom](#)
- [Configure Sheet Print Settings](#)
- [Configure Paper Source](#)

## Configure Page Header and Footer

In GcExcel .NET, you can use the **LeftHeader property**, **LeftFooter property**, **CenterFooter property**, **RightHeader property**, **CenterHeader property**, and the **RightFooter property** of the **IPageSetup interface** 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 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 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 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**, **RightMargin property** and **BottomMargin property** 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** 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** 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** and the **CenterVertically property** 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 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** property and **HPageBreaks** property of the **IWorksheet** 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** 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** 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** 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** 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** in order to control how the spreadsheet is scaled; the **FitToPagesTall property** and the **FitToPagesWide property** in order to set its size; and the **Zoom property** 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** 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** of the **IPageSetup interface**.

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 and at the right by using the **PrintTailColumns** property of the **IPageSetup** interface.

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** property and at the bottom using the **PrintTailRows** property of the **IPageSetup** interface.

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**, **PrintHeadings property**, **BlackAndWhite property**, **PrintComments property** and **PrintErrors property** of the **IPageSetup interface** 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;
```

## Configure Paper Source

GcExcel allows you to configure the paper source while printing a PDF document. It lets you control whether the PDF page size should be used to select the input paper tray while printing. This can be done by using the **PickTrayByPDFSize** property of the **ViewerPreferences** class.

Refer to the following example code to configure paper source while printing a PDF document.

C#

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

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

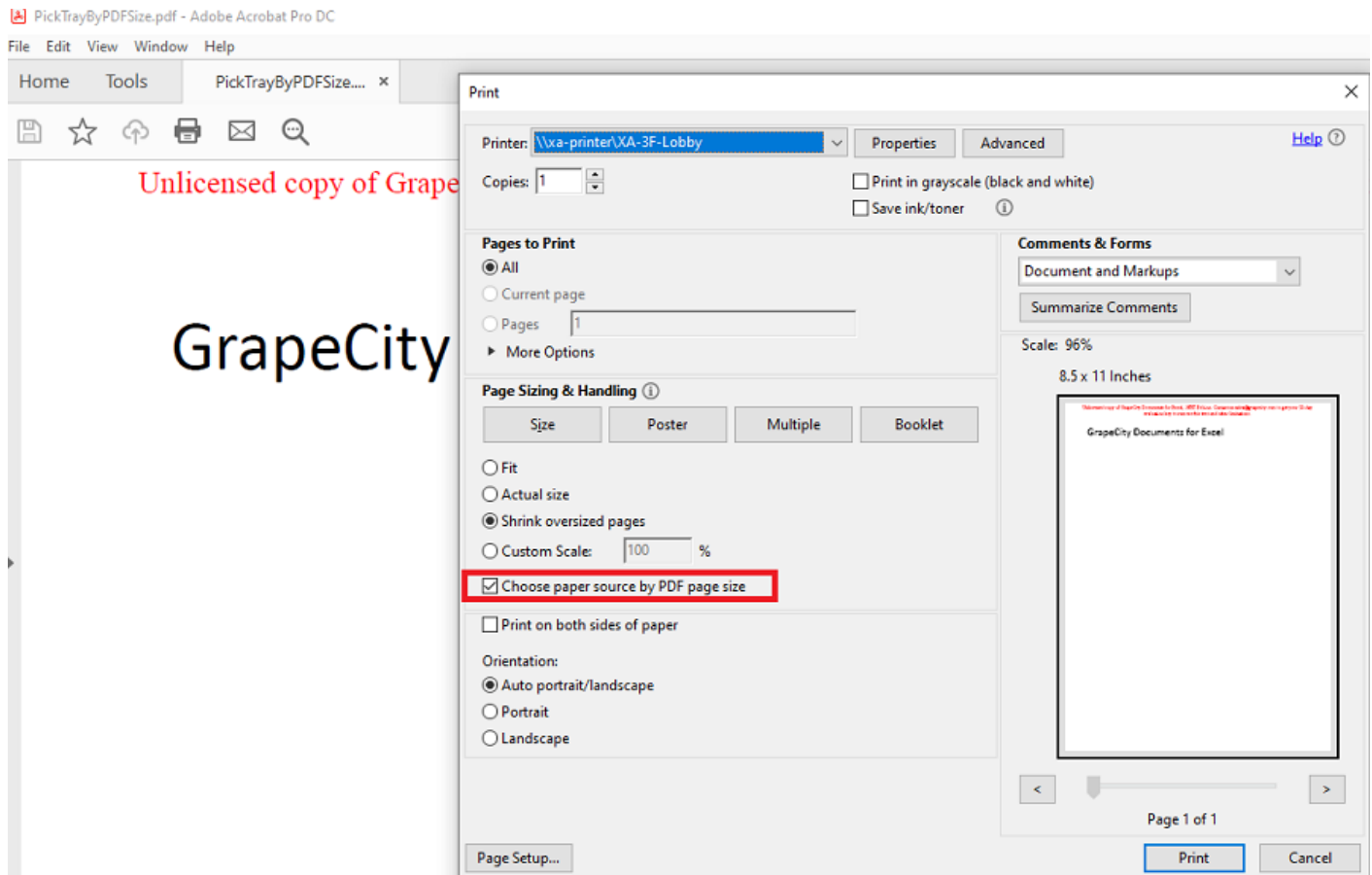
worksheet.Range["A1"].Value = "GrapeCity Documents for Excel";
worksheet.Range["A1"].Font.Size = 25;

// create a pdfSaveOptions object before using ViewerPreferences
PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();

// PDF page size is used to select the input paper tray when printing
pdfSaveOptions.ViewerPreferences.PickTrayByPDFSize = true;

// Save the workbook into pdf file
workbook.Save("PickTrayByPDFSize.pdf", pdfSaveOptions);
```

The paper source setting appears in the 'Print' dialog box of PDF document as shown below:



## Logging

GcExcel supports logging which allows you to capture logs and resolve issues by finding out their root cause. You can store different log levels like debug, error, warning or information based on the configuration in JSON file.

Along with other external libraries, GcExcel uses Microsoft.Extensions.Logging library to implement the logging system. The logging information is supported for Json and Excel I/O and PDF exporting. Logging is disabled by default. To enable it, Workbook.LoggerFactory property needs to be set while initializing the application or website which is explained in detail in the next section.

### Enable Logging

Follow the below steps to enable logging in GcExcel. The logs will be printed to console and saved to a file.

1. Create a new console app and install the following nuget packages:
  - Microsoft.Extensions.Logging.Console
  - Microsoft.Extensions.Configuration.Json
  - Serilog.Extensions.Logging.File
  - If your project targets .NET Framework, Microsoft.Extensions.Logging.Abstractions also needs be installed.
2. Write a method for creating logger factory as shown below:

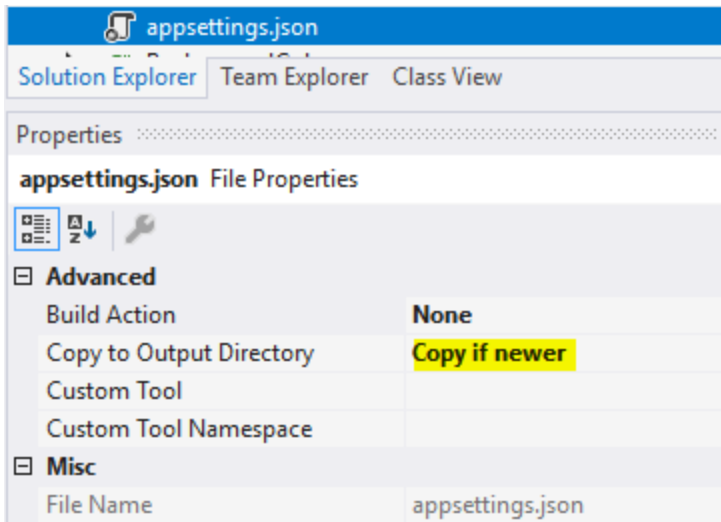
```
C#
```

```
private static ILoggerFactory CreateLoggerFactory()
{
    var builder = new ConfigurationBuilder().
        SetBasePath(Directory.GetCurrentDirectory()).
        AddJsonFile("appsettings.json", optional: true, reloadOnChange: true);
    var cfgRoot = builder.Build();
    var loggingCfg = cfgRoot.GetSection("Logging");
    var factory = LoggerFactory.Create(
        logBuilder => logBuilder.AddConfiguration(loggingCfg).AddConsole());
    factory.AddFile(loggingCfg);
    return factory;
}
```

3. Attach the logger factory to workbook as shown below:

```
C#
static void Main(string[] args)
{
    using (var loggerFactory = CreateLoggerFactory())
    {
        Workbook.LoggerFactory = loggerFactory;
        var wb = new Workbook();
        try
        {
            wb.Save("test.pdf");
        }
        catch (Exception)
        { }
    }
}
```

4. Create a new json file in your project and name it as 'appsettings.json'. Set "Copy to output directory" to "Copy if newer" in properties window. If the file exists already, merge its content with existing file and appsettings.development.json. You can also refer [Configure Serilog File Logs](#) for more information about it.



5. Add the following setting to appsettings.json file.

```
JSON
{
  "Logging": {
    "LogLevel": {
      "Default": "Information"
    },
    "PathFormat": "log-{Date}.txt"
  }
}
```

6. Run the project to observe that logs are printed to console and saved to log file 'log-yyyyMMdd.txt'.

## Set Log Level

GcExcel allows you to capture four log levels, namely, debug, info, warn and error. Their priority order is:

debug < info < warn < error

You can configure which log level needs to be printed to the log file. After setting the log level, the logs with higher or equal priority are included in the printed logs while the logs with lower priority are ignored. For example, If you set the log level to "info", the info, warn and error logs are printed, while the "debug" logs are ignored.

The following setting in appsettings.json sets the default log level to "warning" and GcExcel log level to "Information".

```
JSON
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning",
      "GrapeCity.Documents.Excel": "Information"
    }
  }
}
```

```
}  
}
```

Similarly, the following setting in appsettings.json sets the GcExcel log level of console output to "Information".

#### JSON

```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Warning"  
    },  
    "Console": {  
      "LogLevel": {  
        "GrapeCity.Documents.Excel": "Information"  
      }  
    }  
  }  
}
```

For more information, you can also refer to [Logging in .NET Core and ASP.NET Core](#).

## Save Log Records with Thread ID

You can also choose to print thread ID along with the logs in log files. This is particularly helpful to analyse the specific process or thread creating an issue. The appsettings file uses Microsoft.Extensions.Logging format to generate the logs.

 **Note:** This example does not use [serilog-settings-configuration](#) because it usually reports type load errors.

Follow the below steps to save logs with thread ID:

1. Install the following nuget packages:
  - Microsoft.Extensions.Configuration.Json
  - Microsoft.Extensions.Logging
  - Microsoft.Extensions.Logging.Configuration
  - Serilog.Enrichers.Thread
  - Serilog.Extensions.Logging
  - Serilog.Sinks.RollingFile
  - If your project targets .NET Framework, Microsoft.Extensions.Logging.Abstractions also needs to be installed
2. Write methods for creating a logger factory as shown below:

C#

```
private static ILoggerFactory CreateLoggerFactory()  
{  
    var appSettings = new ConfigurationBuilder().  
        SetBasePath(Directory.GetCurrentDirectory()).  
        AddJsonFile("appsettings.json", optional: true, reloadOnChange: true).  
        Build();  
    return new LoggerFactory().AddSerilog(appSettings, new ThreadEnricher());  
}
```

```
Build();
var loggingSection = appSettings.GetSection("Logging");
var serilogConfig = new LoggerConfiguration();
ConfigureSerilog(loggingSection, serilogConfig);
var factory = new LoggerFactory();
factory.AddSerilog(serilogConfig.CreateLogger());
return factory;
}

private static void ConfigureSerilog(IConfigurationSection loggingSection,
                                   LoggerConfiguration loggerCfg)
{
    loggerCfg.Enrich.WithThreadId();
    string pathFormat = loggingSection["PathFormat"];
    string outputTemplate = loggingSection["OutputTemplate"];
    loggerCfg.WriteTo.RollingFile(pathFormat, outputTemplate: outputTemplate);
    ConfigureMinLevel(loggerCfg.MinimumLevel, loggingSection);
}

private static void ConfigureMinLevel(LoggerMinimumLevelConfiguration minimumLevel,
                                   IConfigurationSection loggingCfgSection)
{
    var logLevelSection = loggingCfgSection.GetSection("LogLevel");
    int pathLength = logLevelSection.Path.Length;
    foreach (var logItem in logLevelSection.AsEnumerable())
    {
        if (logItem.Key.Length <= pathLength)
        {
            continue;
        }
        string name = logItem.Key.Substring(pathLength + 1);
        if (Enum.TryParse(logItem.Value, ignoreCase: true, out LogLevel level))
        {
            var serilogLevel = (LogEventLevel)level;
            if (name == "Default")
            {
                minimumLevel.Is(serilogLevel);
            }
            else
            {
                minimumLevel.Override(name, serilogLevel);
            }
        }
    }
}
```

3. Attach the logger factory to workbook as shown below:

C#

```

        static void Main(string[] args)
        {
            static void Main()
            {
                using (ILoggerFactory loggerFactory = CreateLoggerFactory())
                {
                    Workbook.LoggerFactory = loggerFactory;
                    Workbook wb = new Workbook();
                    try
                    {
                        wb.Save("test.pdf")
                    }
                    catch (Exception)
                    { }
                }
            }
        }
    }

```

4. Create a new json file in your project and name it as 'appsettings.json'. Set "Copy to output directory" to "Copy if newer" in properties window. If the file exists already, merge its content with existing file and appsettings.development.json.

C#

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Warning",
      "GrapeCity.Documents.Excel": "Debug"
    },
    "PathFormat": "log-{Date}.txt",
    "OutputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} (Thread {ThreadId}) [{Level}] {Message}{NewLine}{Exception}"
  }
}

```

Here,

"LogLevel" section has the same schema as Microsoft.Extensions.Logging.

"PathFormat" section specifies the name of log file.

"OutputTemplate" section specifies the format of each log record.

5. Run the project. Logs will be saved to log-yyyyMMdd.txt and the sample output will look like below:

Text

```

2020-09-10 11:44:30.566 +08:00 (Thread 1) [Debug] Save pdf of the workbook.
2020-09-10 11:44:30.615 +08:00 (Thread 1) [Debug] Paginate Start(Workbook)
2020-09-10 11:44:31.104 +08:00 (Thread 1) [Debug] GetDigitWidthOfDefaultStyle
GraphicsType: Pdf

```

```
2020-09-10 11:44:31.108 +08:00 (Thread 1) [Debug] GetDigitWidthOfDefaultStyle
DefaultFont: "FontName = Calibri
FontSize = 11
Bold = False
Italic = False
ScreenWidth = 0
PDFWidth = 5.5751953125
"
2020-09-10 11:44:31.161 +08:00 (Thread 1) [Debug] Count of print ranges: 0
2020-09-10 11:44:31.161 +08:00 (Thread 1) [Debug] Paginate End(Workbook)
2020-09-10 11:44:31.164 +08:00 (Thread 1) [Error] There is no content to print.
```



## Templates

Report generation is crucial for creating marketing strategies, project management, product development cycles, budgeting estimates and growth strategies. It is a common requirement of any business domain. Excel reports are periodically generated and consume a considerable amount of time and effort. However, the chances of manual error cannot be eliminated altogether. That's where the use of templates finds its place. GcExcel provides templates to create highly effective and well-designed Excel reports.

Some of the powerful features provided by GcExcel templates are as follows:

- **Flexible:** GcExcel templates have a highly flexible template syntax and API to bind Excel documents to data. It follows easy data population rules in fields.
- **Efficient:** GcExcel templates provide extended reusability. This means the templates can be used with minor modifications, or as it is time and again, saving both time and effort.
- **Multi-platform:** GcExcel templates are supported on Windows, Linux and macOS.
- **Multi-domain:** GcExcel templates cater to complex use-cases to create Excel reports for any scenario.

The GcExcel template is a pre-defined and formatted workbook. It can be used in the creation of final reports. In the following sections, you will find GcExcel templates used in three diverse use-cases.

- **Use Case 1 - Financial Statistics Report**

In this use-case, we have created a Financial dashboard template to show the Budget statistics of different countries in different seasons or quarters. Here, in the template, the cell A1 contains the title of the template, and the cell D1 contains the year gap for which the financial data has been recorded. Note that here 'ds' is the data source that will populate the country names and quarterly seasons in the Excel report. The names such as BUDGET STATISTICS, BUD and ACT are other data fields of ds. The country field is expanded horizontally to add other countries. Various function fields are used to perform calculations on the Budget and Actual columns.

You can also download the **Excel template layout** used in below example.

	A	B	C	D	E
1	FINANCIAL DASHBOARD			2016-2019	
2					
3		BUDGET STATISTICS{{{E = H}}}			
4		{{ds.country(E = H, S = None)}}			
5		BUD{{{C = B4}}}	ACT{{{C = B4}}}		
6	{{ds.expect(C = B4*A6)}}	{{ds.expect(C = B4*A6)}}	{{ds.actual(C = B4*A6)}}		
7		{{SUM(B6)(C = B4)}}	{{SUM(C6)(C = B4)}}		
8		TOTAL BUD	{{SUM(B7)}}		
9		TOTAL ACTUAL	{{SUM(C7)}}		
10					
11					

The Excel report generated from the Financial Dashboard template is given below:

	A	B	C	D	E
1	FINANCIAL DASHBOARD			2016-2019	
2					
3					
4		USA		Japan	
5		BUD	AST	BUD	AST
6	2016 Q1	\$ 2,36,047	\$ 3,28,554	\$ 3,50,156	\$ 3,70,834
7	2016 Q2	\$ 3,73,060	\$ 2,38,136	\$ 3,69,399	\$ 2,47,324
8	2016 Q3	\$ 2,24,132	\$ 3,00,822	\$ 2,78,834	\$ 2,37,385
9	2016 Q4	\$ 2,69,305	\$ 3,15,337	\$ 2,64,277	\$ 2,45,048
10	2017 Q1	\$ 2,65,397	\$ 2,79,008	\$ 2,03,006	\$ 2,95,389
11	2017 Q2	\$ 2,14,079	\$ 2,06,019	\$ 2,76,987	\$ 2,15,804
12	2017 Q3	\$ 3,70,191	\$ 2,38,294	\$ 3,30,315	\$ 3,30,443
13	2017 Q4	\$ 2,66,843	\$ 2,42,323	\$ 3,07,477	\$ 2,62,512
14		\$ 22,19,054	\$ 21,48,493	\$ 23,80,451	\$ 22,04,739
15		TOTAL BUD	\$ 1,08,60,998		
16		TOTAL ACTUAL	\$ 1,12,50,382		
17					
18					
19					

#### • Use Case 2 - Department & Budget Report

In this template, the budget of each department is depicted based on the salaries of employees in that department. Here, 'ds' is the data source and it populates data fields with the names of departments, managers and employees. The department data fields are expanded horizontally to add more departments. Note that in each department, the static text fields 'Employee' and 'Salary' remains the same. The image below shows the budget report for two departments, Marketing and Sales.

You can also download the **Excel template layout** used in below example.

B	C	D
	<b>{{ds.dpt.name(E=H, R=C2:D6, G=list)}}</b>	
	MANAGER	<b>{{ds.dpt.mgr}}</b>
	BUDGET	<b>{{ds.dpt.bud}}</b>
	<b>Employee</b>	<b>Salary</b>
	<b>{{ds.dpt.emp.name(G=list)}}</b>	<b>{{ds.dpt.emp.salary}}</b>

The Excel report generated from the Department & Budget template is given below:


C	D	E	F
<b>Marketing</b>		<b>Sales</b>	
MANAGER	Carl Sommerset	MANAGER	Kelly Johnson
BUDGET	\$ 3,54,586	BUDGET	\$ 2,37,721
<b>Employee</b>	<b>Salary</b>	<b>Employee</b>	<b>Salary</b>
JoeKline	\$ 49,402	Liam Elmerson	\$ 61,892
Lisa Crane	\$ 81,337	Angela Sanderson	\$ 38,020
John Ryes	\$ 43,503	Blake Schwarz	\$ 55,701
Elli Davidson	\$ 67,334	Linda Barataz	\$ 82,108
Jack Reaze	\$ 68,314		
Ben Lam	\$ 44,696		

#### • Use Case 3 - Sales Report


This use case depicts a template for recording the E-commerce sales of electronic goods in different areas of a country. The data source used here is ds, and it populates the data in the final Excel report with categories, names, cities, sales etc.

The Excel report generated in this case displays the sales of electronic goods individually as well as with respect to their categories. The area and cities are expanded horizontally due to their expansion property. Various function fields are used to perform calculations on the sales and revenue numbers. The value in cell D14 is calculated, first by summing up the revenue in cell C14 and then summing up the values of the whole category (as A14 is its context).

You can also download the **Excel template layout** used in below example.

	A	B	C	D
5				
6	Quarterly Results	Q4 Sales		
7				
8				
9	Business Name:	E-Commerce		
10				
11	Sales		Area {{{E=H}}}	Category's Sales
12			{{ds.Area(E=H)}}	
13			{{ds.City(E=H)}}	
14	{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C14)(C=A14)}}
15	Region's Sales		{{=Sum(C14)(C=C12)}}	{{=Sum(C15)}}
16				

The Excel report generated from the Sales template is given below:

Quarterly Results	Q4 Sales					
Business Name:	E-Commerce					
Sales		Area				Category's Sales
		North America		South America		
		Chicago	Minnesota	Medillin	Quito	
Consumer Electronics	Bose 785593-0050	\$92,800.00				\$42,06,891.00
	Canon EOS 1500D	\$98,650.00	\$89,110.00			
	Haier 394L 4Star	\$3,67,050.00			\$7,29,100.00	
	IFB 6.5 Kg FullyAuto			\$82,910.00		
	Mi LED 40Inch	\$5,50,010.00	\$17,84,702.00			
	Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00	
Mobile	Iphone XR		\$17,34,621.00			\$44,19,531.00
	OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00	
	Redmi 7		\$81,650.00		\$2,76,390.00	
	Samsung S9		\$8,96,250.00		\$7,16,520.00	
Region's Sales		\$63,72,043.00		\$22,54,379.00		\$86,26,422.00

## Template Configuration

Template configuration includes all the features, fields and properties of Templates in GcExcel.

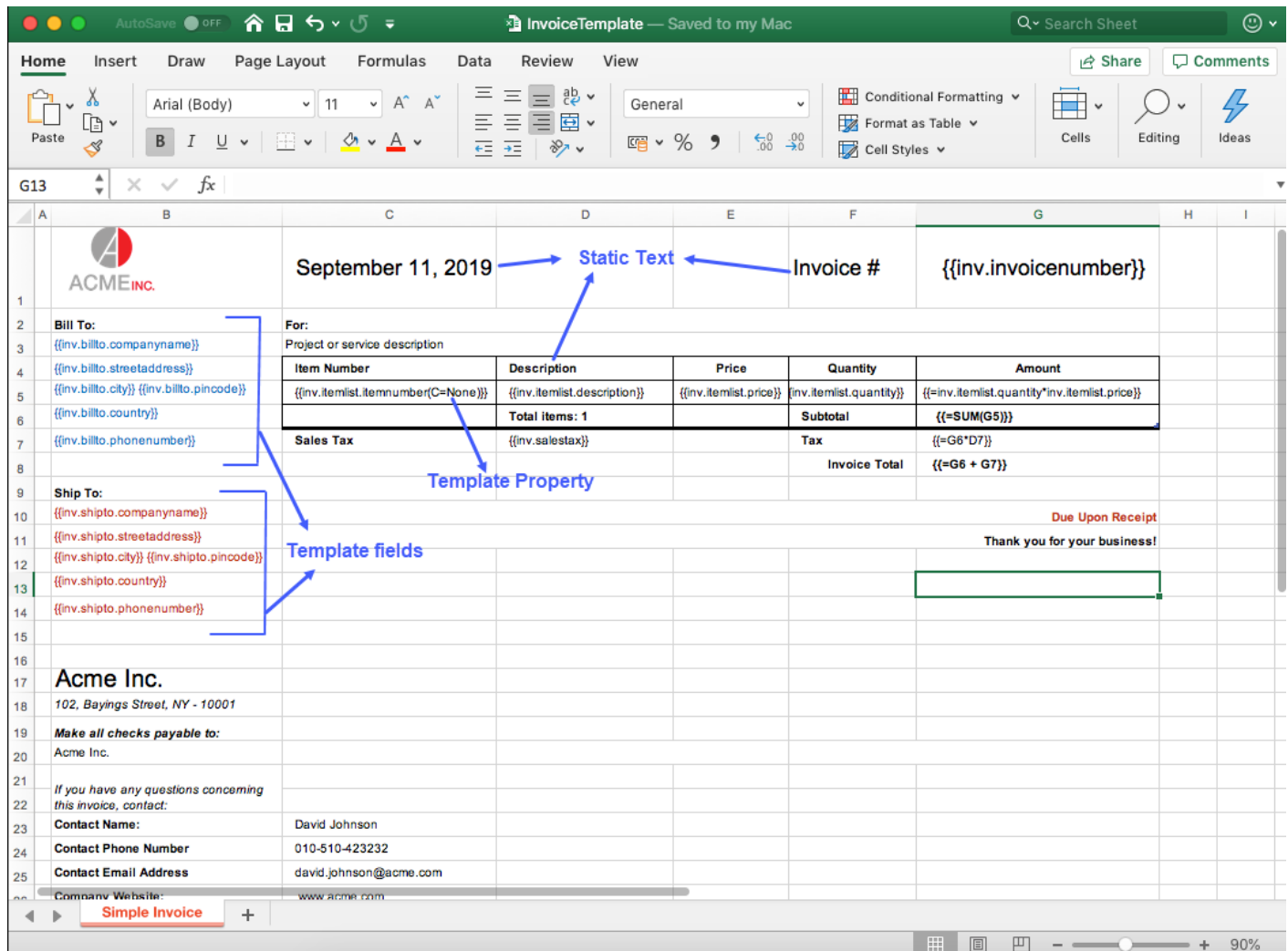
The first step to configure a template is to create a template layout in Excel, which is a pre requisite to generate Excel reports. This layout defines the outline of how the final report will look like and can include static text, data bound fields and other template properties.

Except static fields, all other fields follow syntax and are defined in mustache braces {{ }}. These fields can also include template properties like group, sort, pagebreak etc which are applied on the final Excel report when populated from

the data, in datasource.

Apart from static text, a template layout in Excel is comprised of:

- [Template Fields](#)
- [Template Properties](#)



**Note:** The Excel formulas applied in template layout can be exported as formulas in Excel reports instead of just the cell values. The formula and its range can be viewed in the formula bar of Excel report by selecting the cell to which it has been applied. The Excel formula can be exported by using this syntax in template layout: `{{= formula }}`

For example:

Lets say, `{{=SUM(A5)}}` formula is applied in a template layout. Now, in the generated Excel, formula displayed on clicking the formula cell will be `SUM(A5:A10)`, meaning that this is the range on which the formula is applied.

The formula must be syntactically correct and refer to the right range while defining in the template layout.

Templates, in GcExcel, also supports:

- [Conditional Formatting](#)

- [Global Settings](#)
- [Fixed Layout](#)
- [Default Values in Template Cells](#)
- [PDF Form Builder](#)
- [Charts](#)
- [Tables](#)
- [Sparklines](#)

## Template Fields

A template layout can contain various fields bound to the data source. The below image shows different template fields:

The screenshot shows an Excel spreadsheet titled 'ExpressionTemplate[Template].xlsx'. The layout includes a header section with 'Quarterly Results' and 'Q4 Sales', a logo for 'ACME INC.', and a table with columns for 'Sales', 'Category's Sales', and 'Tax'. Red arrows point to specific fields:

- Data Fields:** {{ds.Category}}, {{ds.Name}}, {{ds.Revenue}}, {{ds.City(E=H)}}
- Function Fields:** {{=Sum(C14)(C=C12)}}, {{=Sum(C15)}}, {{=Sum(E14)}}
- Expression Field:** {{=D14\*0.06(C=A14)}}
- Sheet Name Field:** {{ds.quarter}}

### Data Fields


Data fields are the bound fields which are populated by the data in data source. These fields can be defined in different ways as shown in examples below:

- **Data field**

The data bound fields are defined as: `{{ds.FieldName}}`, where `ds` is the alias of the datasource. For example, `{{ds.grade}}`

- **Nested data field**

If the data is nested, you might want to arrange report as per an inner object field, it can be defined using nested data fields with the following syntax by separating the nested objects with a `'.'`

<b>Student Report</b>	
Student Name	<code>{{report.student.name(R=A3:B9)}}</code>
Father's Name	<code>{{report.student.family.father.name(S=None,E=H)}}</code>
Occupation	<code>{{report.student.family.father.occupation(E=H)}}</code>
Mother's Name	<code>{{report.student.family.mother.name(E=H)}}</code>
Occupation	<code>{{report.student.family.mother.occupation(E=H)}}</code>

- **Inline data field**

It is defined along with the text in a cell. For example, Date: `{{task.dueDate}}`

Also, you can define multiple inline data fields from different data sources as shown below:

Employee Name	Employee ID	Department	Department HOD Name	Department ID	Department Location
Hello: <code>{{emp.name}}</code>	<code>{{emp.id(R=A4:F4, S=None)}}</code>	Belongs to <code>{{emp.department}}</code> of our company	<code>{{department.name.hodname}}</code>	<code>{{emp.department.id}}</code>	<code>{{emp.department.loc}}</code> in US

## Function Fields

Function fields are used to perform calculations in your reports. A function can be applied over a cell or a data field. The standard Excel functions which are supported in the function field are Sum, Count, Average, Max, Min, Product, StdDev, StdDevp, Var and Varp. For example:

`{{=SUM(F4)}}`

`{{=SUM(team.score)}}`


`{{=Count(student.name)}}`




**Note:** Function field supports only one parameter

The function fields can also be calculated over a context. for example, in the below image cell D14 contains function field as well as the context property. The resultant value will be calculated, first by summing up the revenue in cell C14 and then summing up the values of the whole category (as A14 is its context)

You can also download the **Excel template layout** used in the below example.

	A	B	C	D
5				
6	Quarterly Results	Q4 Sales		
7				
8				
9	Business Name:	E-Commerce		
10				
11	Sales		Area {{{E=H}}}	Category's Sales
12			{{ds.Area(E=H)}}	
13			{{ds.City(E=H)}}	
14	{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C14)(C=A14)}}
15	Region's Sales		{{=Sum(C14)(C=C12)}}	{{=Sum(C15)}}
16				

Quarterly Results		Q4 Sales					
Business Name:		E-Commerce					
Sales		Area				Category's Sales	
		North America		South America			
		Chicago	Minnesota	Medillin	Quito		
Consumer Electronics	Bose 785593-0050	\$92,800.00				\$42,06,891.00	
	Canon EOS 1500D	\$98,650.00	\$89,110.00				
	Haier 394L 4Star	\$3,67,050.00			\$7,29,100.00		
	IFB 6.5 Kg FullyAuto			\$82,910.00			
	Mi LED 40inch	\$5,50,010.00	\$17,84,702.00				
	Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00		
Mobile	Iphone XR		\$17,34,621.00			\$44,19,531.00	
	OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00		
	Redmi 7		\$81,650.00		\$2,76,390.00		
	Samsung S9		\$8,96,250.00		\$7,16,520.00		
Region's Sales		\$63,72,043.00		\$22,54,379.00		\$86,26,422.00	

### Expression Fields

Expression fields can be used to perform calculations using operators '+', '-', '\*', '/' and '()'. An expression can be applied over cells or data fields. For example:

```
{{= ds.count*ds.price}}
```

```
{{= ds.price + ds.tax}}
```

```
{{=A18*0.05}}
```

```
{{=A18+D18-G18}}
```

```
{{=(A18+ A20)*0.3}}
```



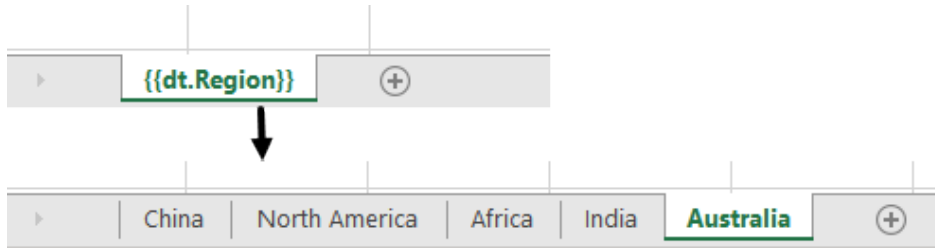
**Note: Template properties** are not supported in Expression fields. Function field and Expression field cannot be used together like {{=Sum(A5+A6)}} or {{=Sum(A5)\*0.01}}, which is not supported.



## Sheet Name

GcExcel supports using bound field in sheet name, which means, the field value of sheet name is populated by the data in data source and multiple worksheets are created. Each worksheet contains data corresponding to its value.

For example, if we specify `{{dt.Region}}` as the sheet name and the data source contains data for 5 regions, the final report will consist of 5 worksheets and the individual sheet will contain data for a specific region.



**Note:** Only the Sort and Group [template properties](#) are supported for sheet name field.

## Template Properties

The template properties are defined along with template fields in round braces ( ) as can be seen in the below image:

The screenshot displays a GcExcel report titled 'TravelTemplate'. The report is divided into several sections, each containing data fields defined by template properties in round braces. The sections include:

- CALIFORNIA** (Section Header)
- TRANSPORT** (Section Header)
- LODGING** (Section Header)
- Emergency Contacts** (Section Header)
- Travel Party** (Section Header)

Key template fields visible in the report include:

- `{{ds1.Carrier(G = list, S = None)}}`
- `{{ds1.FlightNo}}`
- `{{ds1.Date}}`
- `{{ds1.From}}`
- `{{ds1.DepartureTime}}`
- `{{ds1.To}}`
- `{{ds1.ArrivalTime}}`
- `{{ds1.ReservationNo}}`
- `{{ds2.Date}}`
- `{{ds2.Concierge}}`
- `{{ds2.Phone}}`
- `{{ds2.AddressPart1}}`
- `{{ds2.AddressPart2}}`
- `{{ds2.ConfirmNo}}`
- `{{ds2.Days}}`
- `{{ds2.TotalCost}}`
- `{{ds3.Contact(R=A21:C21, G = list, S = None)}}`
- `{{ds3.Phone}}`
- `{{ds4.Contact(G = list, S = None)}}`
- `{{ds4.Phone}}`
- `{{ds4.Note(C=F21)}}`

## Cell Context

The cell context property defines the relationship between cells depending on which the cells are grouped or filtered.

**Value:** Cell location or Data field

*Custom:* Cell context must be specified explicitly.

*Default* (default value): The adjacent cell on the left with E=V, or the adjacent cell on the top with E= H.

*None:* The cell has no context.

## Example

```
{{ds.field(C=A1, E=H)}}
```

```
Hello World! {{{C=A2}}}
```

```
{{=SUM(F4) (C=ds1.team)}}
```

```
{{=SUM(ds1.score) (C=ds1.team*ds1.season)}}
```

For more information about Cell Context, refer [Cell Context](#) topic.

## Cell Expansion

The cell expansion property describes the direction in which the cell values will expand.

**Value:** Enum

*E=N* (None)

*E=H* (Horizontal): Cell data is expanded from left to right.

*E=V* (Vertical-Default value): Cell data is expanded from top to bottom.

## Example

```
{{ds.field(C=A1, E=H)}}
```

For more information about Cell Expansion, refer [Cell Expansion](#) topic.

## Group

The group property allows you to group data in template.

**Value:** Enum

*G=Normal:* The group by field(s) value is not repeated for the corresponding records in the column; instead they are printed once per data group.

*G=Merge* (default value): The same behavior as for the normal parameter, except that it merges the cells in the group by field(s) for each group set.

*G=Repeat:* The group by field(s) value is repeated for the corresponding records.

*G=List:* The field(s) values are listed independently for the corresponding records.

## Example

```
{{ds.field(G=repeat)}}
```

```
{{ds.field(G=list)}}
```

The below image shows how to apply 'merge' grouping on repeating data. You can also download the **Excel template layout** used in below example.

Merge group(default)			Repeat group	
Team	Name		Team	Name
{{ds.Team}}	{{ds.Name}}		{{ds.Team(G=R)}}	{{ds.Name}}
Normal group			List group	
Team	Name		Team	Name
{{ds.Team(G=N)}}	{{ds.Name}}		{{ds.Team(G=L)}}	{{ds.Name}}

Merge group(default)			Repeat group	
Team	Name		Team	Name
New York	Hellen		New York	Hellen
	Hunter		New York	Hunter
	Jim		New York	Jim
	Phillip		New York	Phillip
Seattle	Bob		Seattle	Bob
	Jaguar		Seattle	Jaguar
	Tommy		Seattle	Tommy
Normal group			List group	
Team	Name		Team	Name
New York	Hellen		Seattle	Bob
	Hunter		Seattle	Tommy
	Jim		Seattle	Jaguar
	Phillip		New York	Phillip
Seattle	Bob		New York	Hunter
	Jaguar		New York	Hellen
	Tommy		New York	Jim

## Range

The range property specifies the fallback context for the fields in specified range. All the fields that are covered in the range which have no default nor explicit context, use the current cell in which the range is defined, as their context.

**Value:** Cell range

Default value: Null

### Example

```
{{ds.field(R= B3:F10)}}
```

The below image shows that the range is defined for a student name, specifying that the details will expand and group with respect to Student name. You can also download the **Excel template layout** used in below example.

Sales			
<b>Name:</b>	{{ds.Name(R=B11:F116)}}	<b>Revenue:</b>	{{ds.Revenue}}
<b>Area:</b>	{{ds.Area}}		
<b>City:</b>	{{ds.City}}		

↓

Sales			
<b>Name:</b>	Bose 785593-0050	<b>Revenue:</b>	\$ 92,800.00
<b>Area:</b>	North America		
<b>City:</b>	Chicago		

## Sort

The sort property specifies the type of sorting in template.

**Value:** Enum

*S=Asc* (default value): Ascending

*S=Desc* : Descending

*S=None*: None

### Example

```
{{ds.field(S=Desc)}}
```

The below image shows how the template fields are expanded based on their sorting type. You can also download the **Excel template layout** used in below example.

Sort Ascending(default)		Sort Descending		None sort
Name		Name		Name
{{ds.Name}}		{{ds.Name(S=desc)}}		{{ds.Name(S=none)}}

↓

Sort Ascending(default)		Sort Descending		None sort
Name		Name		Name
Bose 785593-0050		Sennheiser HD 4.40-BT		Bose 785593-0050
Canon EOS 1500D		Samsung S9		Canon EOS 1500D
Haier 394L 4Star		Redmi 7		Haier 394L 4Star
IFB 6.5 Kg FullyAuto		OnePlus 7Pro		IFB 6.5 Kg FullyAuto
Iphone XR		Mi LED 40inch		Mi LED 40inch
Mi LED 40inch		Iphone XR		Sennheiser HD 4.40-BT
OnePlus 7Pro		IFB 6.5 Kg FullyAuto		Iphone XR
Redmi 7		Haier 394L 4Star		OnePlus 7Pro
Samsung S9		Canon EOS 1500D		Redmi 7
Sennheiser HD 4.40-BT		Bose 785593-0050		Samsung S9

## Page Break

The page break property specifies whether to add a new page after a field or not.

- If the template cell is located in the first column, horizontal page break is added.
- If the template cell is located in the first row, vertical page break is added
- If the template cell is located in any other location than the first row and first column, both a horizontal and a vertical page break is added

**Value:** Boolean

*Pagebreak=True*

*Pagebreak=False* (Default value)

## Example

```
{{ds.field(Pagebreak=true)}}
```

The below image shows that a page break will be added after 'Category' field. You can also download the **Excel template layout** used in below example.

Category	Name
{{ds.Category(Pb=true)}}	{{ds.Name}}

Category	Name
Consumer Electronics	Bose 785593-0050
	Canon EOS 1500D
	Haier 394L 4Star
	IFB 6.5 Kg FullyAuto
	Mi LED 40inch
	Sennheiser HD 4.40-BT
Mobile	Iphone XR
	OnePlus 7Pro
	Redmi 7
	Samsung S9

## Image

The image property specifies whether to add an image or not and if yes, its height and width can also be specified.

The supported image data type is byte[] and base64 string.

The position of image in the cell can be controlled by setting the horizontal and vertical alignment style of cell. By default, the image is located in the center of the cell horizontally and vertically, both.


**Value:** Boolean

*Image = True*

*Image = False* (Default value)

Image.width=String value: Default value is cell width.

Image.height=String value: Default value is cell height.

 **Note:** Image property should be set to true in order to set its width or height.

## Example

```
{{ds.icon(Image=true)}}
```

```
{{ds.icon(Image=true, Image.width=150px)}}
```

```
{{ds.icon(Image=true, Image.height=150px)}}
```

The below image shows how an image can be added in the Excel report. You can also download the **Excel template layout** used in below example.



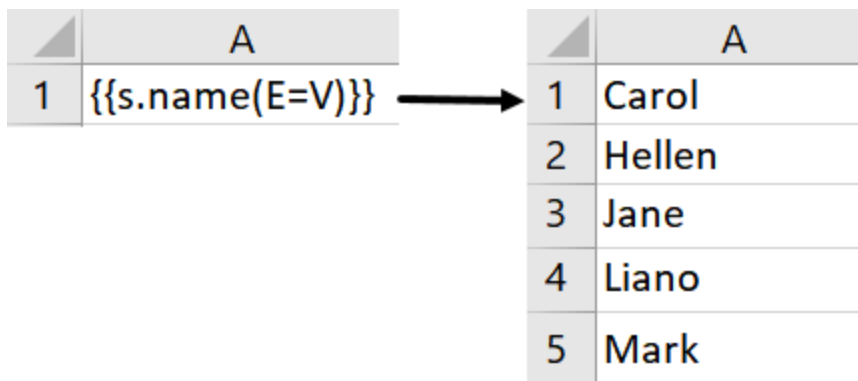
## Cell Expansion

The layout of a template in Excel consists of various fields, some of which are bound to a data source. The value of a bound field in a template, expands to several cells in report. For example, if you have created a field named 'Color' and bound it to the data source which contains 10 values for 'Color', the cell will expand to 10 values.

The expansion of a cell depends on the rules explained below:

### Vertical Expansion

The cell values will expand vertically if the expansion property of the cell is set to vertical, that is, "**E=V**", as shown below. The default expansion setting is vertical, which means if you do not specify any expansion property in the cell, the cell values will expand vertically.

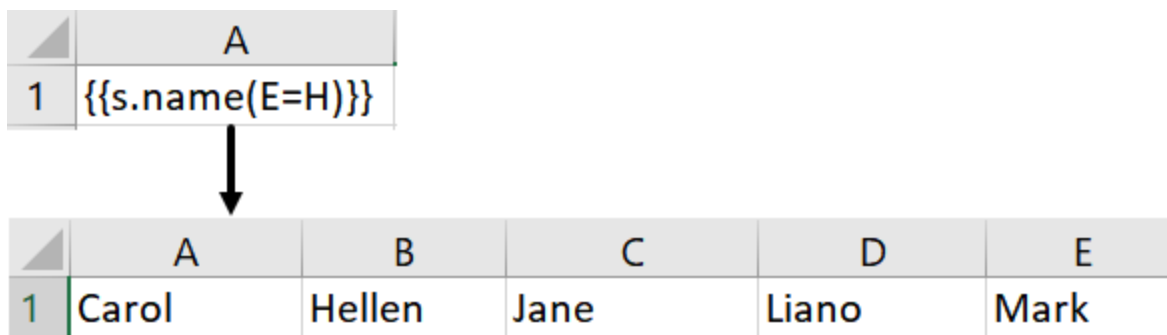


	A
1	{{s.name(E=V)}}

1	Carol
2	Hellen
3	Jane
4	Liano
5	Mark

### Horizontal Expansion

The cell values will expand horizontally if the expansion property of the cell is set to horizontal, that is, "**E=H**", as shown below:



	A
1	{{s.name(E=H)}}

1	Carol	Hellen	Jane	Liano	Mark
---	-------	--------	------	-------	------



## Cell Context

A template layout can contain multiple bound fields which depend on each other while expansion in the final Excel report.

For example, in the below image, the 'team' and 'name' are two bound fields in the template layout where team is the former cell and name is the latter cell. Now, the 'name' field will depend on the 'team' field to group or filter its values based on the team. Also, the direction of expansion of the 'name' field will be decided by the 'team' field. Here, team is the context cell of name.

	A	B
1	{{s.team}}	{{s.name}}

### Context Relationships

When multiple fields bound to a data source are defined in a template layout, a relationship is established between them which is called 'Context' relationship. The former cell is called the context cell of latter cell. Based on this relationship, the data is filtered or grouped while expansion in the final report.

There are two types of context relationships:

- Filtering Relationship:** The data in the cell is filtered using data of the context cell as the filter condition. For example, in the below image, the data in the 'name' cell is filtered corresponding to the data in its context cell:

	A	B
1	{{s.team}}	{{s.name}}

→

	A	B
1		Carol
2	Avengers	Hellen
3		Jane
4		Liano
5	Dominators	Mark

- Following Relationship:** The data in the cell is grouped according to the expansion direction of the data in context cell. For example, in the below image, the data in the 'name' cell is grouped and expanded horizontally depending on its context cell:

	A
1	{{s.team(E=H)}}
2	{{s.name}}

→

	A	B
1	Avengers	Dominators
2	Carol	Jane
3	Hellen	Liano
4		Mark
5		

## Context Cell

The context of a cell is defined using the 'C' property. The data in cells expand vertically or horizontally depending on their context. A cell's context can be set in the below ways:

- **None:** No cell context (C= None)

	A		A	B
1	{{s.team(E=H)}}	→	1	Avengers
2	{{s.name(C=None)}}		2	Carol
			3	Hellen
			4	Jane
			5	Liano
			6	Mark

- **Custom:** The cell context is specified explicitly using 'C' property

	A	B		A	B
1	{{s.team}}		→	1	Avengers
2				2	
3		{{s.name(C=A1)}}		3	Carol
				4	Hellen
				5	Dominators
				6	
				7	Jane
				8	Liano
				9	Mark

- **Default:** If no context is defined in the cell, the default context cell is the adjacent cell on the left with E=V (expanding vertically), or adjacent cell on the top with E= H (expanding horizontally)

For example, in the below image, A1 is the context cell of B1 and expands vertically.

	A	B
1	{{s.team}}	{{s.name}}

And, A1 is the context cell of A2 and expands horizontally.

	A
1	{{s.team(E=H)}}
2	{{s.name}}

### Context Precedence

The priority order in which context should be applicable on a cell is determined in the following order:

**Explicit context > Default Context > Fallback context**

- **Explicit context:** The context defined by **C** property in the cell itself
- **Default context:** If no context is defined in the cell, the default context is given priority
- **Fallback context:** If there is no adjacent cell value on the left or top, the cell looks for a cell with **R** (Range) property which covers its location, and use it as its context.

The **Fallback context** can be defined in a cell using the Range property, in case no default or explicit context is defined. The cell that defines the range is followed as a context for other cells to expand.

For example, the below template layout is created to display the sales details for different camera models, which means the data needs to expand with respect to the model of the camera. Then after a break, the sales details need to be displayed for another model of the camera. Instead of adding context to every field, we can define the range R=B11:F16 for Camera model - {{ds.Name (R=B11:F16)}}, stating that the sales details need to expand and group with respect to the camera model.

E-Commerce				
<b>Sales</b>				
<b>Name:</b>	{{ds.Name(R=B11:F16)}}		<b>Revenue:</b>	{{ds.Revenue}}
<b>Area:</b>	{{ds.Area}}			
<b>City:</b>	{{ds.City}}			

The above template layout will generate the following Excel report:

E-Commerce				
Sales				
Name:	Bose 785593-0050	Revenue:	\$	92,800.00
Area:	North America			
City:	Chicago			
Sales				
Name:	Canon EOS 1500D	Revenue:	\$	98,650.00
Area:	North America			
City:	Chicago			

## Conditional Formatting

Conditional formatting rules can be defined in Template layout which are applied to the expanded cells in Excel report.

For example, the below template layout applies a conditional formatting rule in data field: {{ds.Revenue}}. The rule specifies to show the cell value in red if it is less than or equal to 500000 and in green if it is equal to or greater than 100000. Also, the icons are displayed alongside cell values.

You can also download the **Excel template layout** used in below example.

Business Name:	E-Commerce				
Sales		Area {{{E=H}}}	Category's Sales		
		{{ds.Area(E=H)}}			
		{{ds.City(E=H)}}			
{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C14)(C=A14)}}		
Region's Sales		{{=Sum(C14)(C=C12)}}	{{=Sum(C15)}}		

Conditional Formatting Rules Manager

Show formatting rules for: Current Selection

New Rule... Edit Rule... Delete Rule

Rule (applied in order shown)	Format	Applies to	Stop If True
Icon Set		= \$C\$14	<input type="checkbox"/>
Cell Value >= 500000	AaBbCcYyZz	= \$C\$14	<input type="checkbox"/>
Cell Value <= 100000	AaBbCcYyZz	= \$C\$14	<input type="checkbox"/>

OK Close Apply

When the template is processed, the conditional formatting rule is applied to the expanded data in the final Excel report as shown below.

Business Name:		E-Commerce					
Sales		Area				Category's Sales	
		North America					
		Chicago	Minnesota	Medillin	Quito		
Consumer Electronics	Bose 785593-0050	⬇️ \$92,800.00				\$42,06,891.00	
	Canon EOS 1500D	⬇️ \$98,650.00	⬇️ \$89,110.00				
	Haier 394L 4Star	\$3,67,050.00			⬆️ \$7,29,100.00		
	IFB 6.5 Kg FullyAuto			⬇️ \$82,910.00			
	Mi LED 40inch	⬆️ \$5,50,010.00	⬆️ \$17,84,702.00				
Mobile	Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00	\$44,19,531.00	
	Iphone XR		⬆️ \$17,34,621.00				
	OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00		
	Redmi 7		⬇️ \$81,650.00		\$2,76,390.00		
	Samsung S9		⬆️ \$8,96,250.00		⬆️ \$7,16,520.00		
Region's Sales		\$63,72,043.00		\$22,54,379.00		\$86,26,422.00	

### Limitation

If the formula reference in a conditional formatting rule refers to a template cell, it is not handled correctly by GcExcel Template. The formula is not adjusted after template processing, that is, the formula will remain the same and will not get updated dynamically with the range.

For example:

If cell B5 has a formula reference in conditional formatting rule "=\$A\$5 > 100". And both A5 and B5 are template cells


then after the template processing, conditional formatting rule may be applied from B5:B10 but, it "`=$A$5 > 100`" will not change dynamically with the cell range.

## Global Settings

Global settings, in GcExcel Templates, are the settings which when defined are applied throughout the template. These settings save lots of effort when same properties need to be applied on several fields. Global settings can be applied in all the template layouts and even in multiple worksheets of a workbook.

The global settings provided by GcExcel template are explained below:

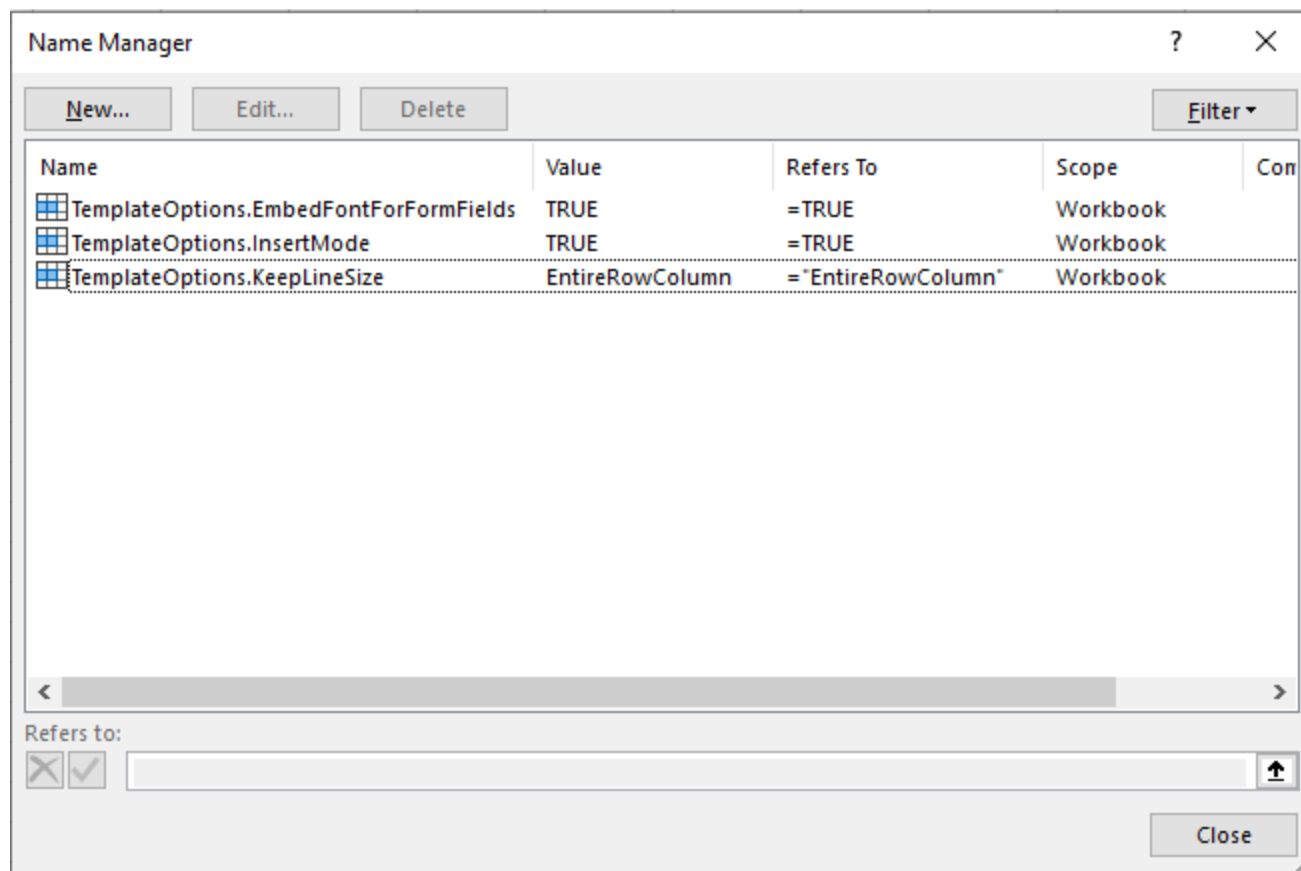
Global Settings	Description	Value
TemplateOptions.KeepLineSize	It specifies whether the row height and column width should be kept the same throughout the template	<b>Type:</b> Boolean <b>Value:</b> True False (Default Value)
TemplateOptions.InsertMode	It specifies whether to insert extra cells or entire rows and columns when extra space is needed while expanding the template	<b>Type:</b> String <b>Value:</b> Cells (Default Value) EntireRowColumn
TemplateOptions.EmbedFontForFormFields	It specifies whether the fonts used by form fields should be embedded in exported PDF file.  <b>Note:</b> This setting is only applicable for <a href="#">PDF form fields</a>	<b>Type:</b> Boolean <b>Value:</b> True (Default Value) False

 **Note:** The scope of global settings is within a workbook only, which means, that all the worksheets in a workbook will apply the global settings.

The global settings can be applied in GcExcel template by using either of the two ways explained below:

### Define Global Settings in Template Layout

Global settings can be defined in Template layout in Excel in 'Name Manager' dialog box as shown below. The 'Name Manager' can be accessed by navigating through Formulas tab > Defined Names group, and then clicking the 'Name Manager'.



### Set Global Settings using Code

The global settings can be defined in GcExcel after loading the Excel template by using built-in workbook defined names **TemplateOptions**. The **Add** method of **INames** interface can be used to apply the global settings. The method takes **Name** and **RefersTo** properties as the parameters:

The value of **Name** property in built-in defined name is taken as the template global option's name. It is case-sensitive.

The value of **RefersTo** property in built-in defined name is taken as the template global option's value. It is case-sensitive.

Refer to the below example code to specify the global settings in template:

C#

```
Workbook workbook = new Workbook();
workbook.Open("template.xlsx");

//Init template global settings
workbook.Names.Add("TemplateOptions.KeepLineSize", "true");
workbook.Names.Add("TemplateOptions.InsertMode", "EntireRowColumn");
//Global setting for PDF form fields
workbook.Names.Add("TemplateOptions.EmbedFontForFormFields", "true");

//Add data source
```

```

workbook.AddDataSource("ds", ds);

//Invoke to process the template
workbook.ProcessTemplate();


workbook.Save("report.xlsx");

```

This template example records the E-commerce sales of electronic goods in different areas of a country. You can also download the **Excel template layout**.

### KeepLineSize

The below image shows the Excel report when 'TemplateOptions.KeepLineSize' is set to true.

	A	B	C	D	E	F	G
5							
6	Quarterly Results	Q4 Sales					
7							
8							
9	Business Name:	E-Commerce					
10							
11	Sales		Area				Category's Sales
North America							
Chicago			Minnesota	Medillin	Quito		
14	Consumer Electronics	Bose 785593-0050	\$92,800.00				\$42,06,891.00
15		Canon EOS 1500D	\$98,650.00	\$89,110.00			
16		Haier 394L 4Star	\$3,67,050.00			\$7,29,100.00	
17		IFB 6.5 Kg FullyAuto			\$82,910.00		
18		Mi LED 40inch	\$5,50,010.00	\$17,84,702.00			
19		Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00	
20	Mobile	Iphone XR		\$17,34,621.00			\$44,19,531.00
21		OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00	
22		Redmi 7		\$81,650.00		\$2,76,390.00	
23		Samsung S9		\$8,96,250.00		\$7,16,520.00	
24	Region's Sales		\$63,72,043.00		\$22,54,379.00		\$86,26,422.00
25							
26							
27							
28							
29							
30							

### InsertMode

The below image shows the Excel report when 'TemplateOptions.InsertMode' is set to EntireRowColumn. By doing this, the row height and outline groups of the rows are retained when the template expands.



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

A

B

C

D


E

F

G

Quarterly Results

Q4 Sales



Business Name:

E-Commerce

Sales

Area

Category's Sales

North America

Chicago

Minnesota

Medillin

Quito

Consumer Electronics

Bose 785593-0050

Canon EOS 1500D

Haier 394L 4Star

IFB 6.5 Kg FullyAuto

Mi LED 40inch

Sennheiser HD 4.40-BT

\$92,800.00

\$98,650.00

\$3,67,050.00

\$5,50,010.00

\$1,78,100.00

\$89,110.00

\$82,910.00

\$7,29,100.00

\$2,34,459.00

\$42,06,891.00

Mobile

Iphone XR

OnePlus 7Pro

Redmi 7

Samsung S9

\$4,99,100.00

\$81,650.00

\$8,96,250.00

\$2,15,000.00

\$2,76,390.00

\$7,16,520.00

\$44,19,531.00

Region's Sales

\$63,72,043.00

\$22,54,379.00

\$86,26,422.00

### EmbedFontForFormFields

This setting allows you to embed font files used by form fields in the PDF document generated by GcExcel.

When true, any arbitrary character is displayed correctly even if your machine or browser does not have corresponding fonts installed. However, it may generate large sized PDF documents, especially when East Asian characters are used. When false, the generated PDF document is of optimal size but messy code will be displayed if your machine or browser does not have corresponding fonts installed.

The below image shows the PDF form generated by GcExcel when 'TemplateOptions.EmbedFontForFormFields' is set to True. You can also download the **Excel template layout** used in below example.

(R1.5HP用)

## 新規登録申請書

日本小型船舶検査機 殿

この申請書フォームには入力できます。

申請者（新所有者等）

〒

住 所：

（フリガナ）

氏名又は名称： 印

## Fixed Layout

When GcExcel processes a template layout, it inserts blank lines first and then sets the data and style to generate the final report. In cases where a fixed layout is defined in the template, GcExcel provides two properties you can use to properly load the data in this fixed layout area.

### 1. fillMode(FM) Property


The fillMode property can be set to 'overwrite' to set the data directly in template cells (without inserting blank rows). The style of the template layout is retained and data is filled into it.

The below template example records the E-Commerce sales of electronic goods in different areas of a country and uses 'overwrite' fillMode property. You can also download the **Excel template layout** from [here](#).

<b>Business Name:</b>	E-Commerce		
<b>Area</b>	<b>City</b>	<b>Name</b>	<b>Sales</b>
{{ds.Area(G=list, FM=overwrite)}}	{{ds.City}}	{{ds.Name}}	{{ds.Revenue}}
<b>Region's Sales</b>			<b>\$0.00</b>

The data in data source contains 8 rows. After GcExcel processes the template layout, the Excel report will look like below:


Business Name:	E-Commerce		
Area	City	Name	Sales
North America	Chicago	Bose 785593-0050	\$92,800.00
North America	Chicago	Haier 394L 4Star	\$3,67,050.00
South America	Santiago	Haier 394L 4Star	\$5,78,900.00
South America	Medillin	IFB 6.5 Kg FullyAuto	\$82,910.00
North America	Chicago	Sennheiser HD 4.40-BT	\$1,78,100.00
South America	Quito	OnePlus 7Pro	\$2,15,000.00
North America	Minnesota	Redmi 7	\$81,650.00
South America	Quito	Samsung S9	\$7,16,520.00
Region's Sales			

 **Note:** If fillMode property is not defined in the template layout, the default behavior is followed, which is to insert the blank lines first.

## 2. fillRange(FR) Property

The fillRange property should be used when fillMode is set to overwrite and the data in data source exceeds the area of fixed layout in template. So if the data overflows, the range defined by fillRange property is duplicated to fill additional data.

For example: If the data in data source contains 20 rows and fillRange property defines the cell range as A1:A8. The range will be duplicated to 8 more rows (total 16 rows) and then again to 8 more rows (total 24 rows), to fill the complete data of 20 rows.

 **Note:** When data in data source overflows and fillRange property is missing, range will not be duplicated to set data. Instead, data will be filled beneath the range area in existing rows.

The below template example records the E-Commerce sales of electronic goods in different areas of a country. It uses 'overwrite' fillMode property along with fillRange property to accommodate the additional data. You can also download the **Excel template layout** from here.

9	<b>Business Name:</b>	E-Commerce		
10				
11	<b>Area</b>	<b>City</b>	<b>Name</b>	<b>Sales</b>
12	{{ds.Area(G=list, FM=overwrite, FR=A12:D23)}}	{{ds.City}}	{{ds.Name}}	{{ds.Revenue}}
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24	<b>Region's Sales</b>			<b>\$0.00</b>

The data in data source contains 26 rows. The fillRange property defines cell range for 12 rows and it is duplicated after that twice to fill all the data. After GcExcel processes the template layout, the Excel report will look like below:

9	Business Name:	E-Commerce		
10				
11	Area	City	Name	Sales
12	North America	Chicago	Bose 785593-0050	\$92,800.00
13	North America	New York	Bose 785593-0050	\$92,800.00
14	South America	Santiago	Bose 785593-0050	\$19,550.00
15	North America	Chicago	Canon EOS 1500D	\$98,650.00
16	North America	Minnesota	Canon EOS 1500D	\$89,110.00
17	South America	Santiago	Canon EOS 1500D	\$4,59,000.00
18	North America	Chicago	Haier 394L 4Star	\$3,67,050.00
19	South America	Quito	Haier 394L 4Star	\$7,29,100.00
20	South America	Santiago	Haier 394L 4Star	\$5,78,900.00
21	North America	Fremont	IFB 6.5 Kg FullyAuto	\$9,04,930.00
22	South America	Buenos Aires	IFB 6.5 Kg FullyAuto	\$6,73,800.00
23	South America	Medillin	IFB 6.5 Kg FullyAuto	\$82,910.00
24	North America	Chicago	Mi LED 40inch	\$5,50,010.00
25	North America	Minnesota	Mi LED 40inch	\$17,84,702.00
26	South America	Santiago	Mi LED 40inch	\$1,02,905.00
27	North America	Chicago	Sennheiser HD 4.40-BT	\$1,78,100.00
28	South America	Quito	Sennheiser HD 4.40-BT	\$2,34,459.00
29	North America	Minnesota	Iphone XR	\$17,34,621.00
30	South America	Santiago	Iphone XR	\$1,09,300.00
31	North America	Chicago	OnePlus 7Pro	\$4,99,100.00
32	South America	Quito	OnePlus 7Pro	\$2,15,000.00
33	North America	Minnesota	Redmi 7	\$81,650.00
34	South America	Quito	Redmi 7	\$2,76,390.00
35	North America	Minnesota	Samsung S9	\$8,96,250.00
36	South America	Buenos Aires	Samsung S9	\$8,96,250.00
37	South America	Quito	Samsung S9	\$7,16,520.00
38				
39				
40				
41				
42				
43				
44				
45				
46				
47				
48	Region's Sales			

## Default Values in Template Cells

While working with GcExcel templates, some cells are displayed as blank in the final Excel report when they have no data or empty value in data source. To handle such cases, GcExcel provides 'defaultValue' (DV) property which sets the specified default value in template cells containing [data fields](#). The specified default value for cells containing no data or empty value in data source can be viewed in the final Excel report.

Property	Data Type	Description	Example
----------	-----------	-------------	---------

DV (defaultValue)	String or Number	The default value to show when there is no data in data source.	{{ds.BaseAmount(C=B7*C6, defaultValue = 0)}}  {{ds.BaseAmount(C=C15*D14, defaultValue = "-")}}
----------------------	---------------------	--	--

The below template is created to maintain E-commerce sales of electronic goods and their revenues in different areas of a country. The 'ds.Revenue' field will display default value when no data is present in data source. You can also download the **Excel template layout** used in below example.

<b>Business Name:</b>		E-Commerce	
<b>Sales</b>		<b>Area {{{E=H}}}</b>	<b>Category's Sales</b>
		<b>{{ds.Area(E=H)}}</b>	
		<b>{{ds.City(E=H)}}</b>	
<b>{{ds.Category}}</b>	<b>{{ds.Name}}</b>	<b>{{ds.Revenue(defaultValue=0)}}</b>	<b>{{=Sum(C14)(C=A14)}}</b>
<b>Region's Sales</b>		<b>{{=Sum(C14)(C=C12)}}</b>	<b>{{=Sum(C15)}}</b>

When the template is processed, the default values are displayed in the final Excel report as shown below:

Business Name:		E-Commerce					
Sales		Area				Category's Sales	
		North America		South America			
		Chicago	Fremont	Buenos Aires	Quito		
Consumer Electronics	Bose 785593-0050	\$92,800.00	\$0.00	\$0.00	\$0.00	\$38,28,899.00	
	Canon EOS 1500D	\$98,650.00	\$0.00	\$0.00	\$0.00		
	Haier 394L 4Star	\$3,67,050.00	\$0.00	\$0.00	\$7,29,100.00		
	IFB 6.5 Kg FullyAuto	\$0.00	\$9,04,930.00	\$6,73,800.00	\$0.00		
	Mi LED 40Inch	\$5,50,010.00	\$0.00	\$0.00	\$0.00		
	Sennheiser HD 4.40-BT	\$1,78,100.00	\$0.00	\$0.00	\$2,34,459.00		
Mobile	Iphone XR	\$0.00	\$0.00	\$0.00	\$0.00	\$26,03,260.00	
	OnePlus 7Pro	\$4,99,100.00	\$0.00	\$0.00	\$2,15,000.00		
	Redmi 7	\$0.00	\$0.00	\$0.00	\$2,76,390.00		
	Samsung S9	\$0.00	\$0.00	\$8,96,250.00	\$7,16,520.00		
Region's Sales		\$26,90,640.00		\$37,41,519.00		\$64,32,159.00	



**Note:** The default value in template cells can not be displayed for [function](#) or [expression](#) fields.


## PDF Form Builder

GcExcel Templates provide the ability to build PDF forms with various form fields using Excel as the designer. The form fields can be defined using the proper syntax while creating template layouts. After the template is processed, the result can be exported to a PDF document that includes the pre-defined form fields.

The **"form"** property can be used to define a PDF form field. The value of this property is in JSON format and a JSON string can be used to describe all settings of the form field. For example:


- `{{ds1.Name(form={"type": "textbox", "name": "username", "value": "Input your name!", "font":{"size":15, "color": "#ff0000", "bold": true}, "required": true)}}}`
- `{{(form={"type": "listbox", "name": "cities", "value": ["Xi'An", "Beijing"], "font":{"size":11, "color": "#ff00ff", "bold": true}, "required":`

```
true}}}}
```

 **Note:** The property name and enum values are case insensitive.

The following standard PDF form fields are supported:

- Check box
- Combo box
- List box
- Button
- Radio button
- Signature
- Text box

 **Note:** The form fields are visible only in PDF documents and not in Excel.

### Bound PDF Form

Consider an example for generating a bound PDF form by using GcExcel Templates. In this case, an address book is generated in PDF by defining textbox fields in template cells. The textbox fields are defined in a way that they relate to common details of an address book, like:

Name : {{ds.Name(form={"type": "textbox", "name": "name", "font":{"color": "#000000", "bold": true}})}}

Email: {{ds.Email(form={"type": "textbox", "name": "Email", "font":{"color": "#EC881D"}})}}

These textbox fields are bound fields, whose data is populated from the data source and is displayed in the PDF form after template processing. You can also download the **Excel template layout** from here.

#### The Address Book

NAME	WORK	CELL	HOME	EMAIL	BIRTHDAY	ADDRESS
{{ds.Name(form={"type": "textbox", "name": "name", "font":{"color": "#000000", "bold": true}})}}	{{ds.Work(form={"type": "textbox", "name": "Work", "font":{"color": "#000000", "bold": true}})}}	{{ds.Cell(form={"type": "textbox", "name": "Cell", "font":{"color": "#000000", "bold": true}})}}	{{ds.Home(form={"type": "textbox", "name": "Home", "font":{"color": "#000000", "bold": true}})}}	{{ds.Email(form={"type": "textbox", "name": "Email", "font":{"color": "#EC881D", "bold": true}})}}	{{ds.Birthday(form={"type": "textbox", "name": "Birthday", "font":{"color": "#000000", "bold": true}})}}	{{ds.Address(form={"type": "textbox", "name": "Address", "font":{"color": "#000000", "bold": true}})}}

After GcExcel processes the template and exports it to a PDF document, the PDF form will look like below:

#### The Address Book

NAME	WORK	CELL	HOME	EMAIL	BIRTHDAY	ADDRESS
Andrew Lepp	6235320178	6235320178	6235320178	Andrew@example.com	10/9/1996	123 N. Maple
James Williams	5235550879	5235550879	5235550879	James@example.com	4/5/1995	123 N. Maple
John Smith	3215230123	3215230123	3215230123	John@example.com	5/20/1990	4456 E. Aspen
Kim Abercrombie	1235550123	1235550123	1235550123	Kim@example.com	4/13/1991	123 N. Maple
Mark Jordan	1238640185	1238640185	1238640185	Mark@example.com	12/13/1988	123 N. Maple

## Unbound PDF Form

Consider an example for generating an unbound PDF form by using GcExcel Templates. In this case, a wage and tax statement is generated in PDF by defining textbox and checkbox fields in template cells, like:

## Textbox field:

```
{{(form={"type": "textbox", "name": "tips", "backgroundcolor": "#ffabab"})}}
```

## Checkbox fields:

```
{{(form={"type": "checkbox", "name": "Retirement", "border": {"color": "#ff0000"}})}}
```

```
{{(form={"type": "checkbox", "name": "Statutory", "border": {"color": "#ff0000"}})}}
```

These fields are unbound fields, and their data should be filled directly in the PDF form after template processing. You can also download the **Excel template layout** from here.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
2	22222	OID	{{(form={																		
3	Employee's social security number																	For Official Use Only OMB No. 1545-0008			
4	b Employer identification number (EIN)										1 Wages, tips, other compensation					2 Federal income tax withheld					
5																					
6	c Employer's name, address, and ZIP code										3 Social security wages					4 Social security tax withheld					
7																					
8											5 Medicare wages and tips					6 Medicare tax withheld					
9											7 Social security tips					8 Allocated tips					
10																					
11	d Control number										9					10 Dependent care benefits					
12											11 Nonqualified plans					12a See instructions for box 12					
13	e Employee's first name and initial										Last name					12b					
14																					
15											13 Statutory employee Retirement Third-party sick pay					14 Other					
16																					
17																					
18																					
19																					
20																					
21																					
22																					
23	f Employee's address and ZIP code																				
24	15 State Employer's state ID number					16 State wages, tips, etc.					17 State income tax					18 Local wages, tips, etc.					
25																					
26																					
27																					
28																					
29	Form <b>W-2</b> Wage and Tax Statement										2020					Department of the Treasury—Internal Revenue Service For Privacy Act and Paperwork Reduction Act Notice, see the separate instructions.					
30																					
31																					
32	Copy A—For Social Security Administration. Send this entire page with															Cat. No. 10134D					
33	Form W-3 to the Social Security Administration; photocopies are not																				
34																					
35																					

After GcExcel processes the template and exports it to a PDF document, the PDF form will look like below:



22222		VOID <input type="checkbox"/>		a Employee's social security number		For Official Use Only ► OMB No. 1545-0008	
b Employer identification number (EIN)				1 Wages, tips, other compensation		2 Federal income tax withheld	
c Employer's name, address, and ZIP code				3 Social security wages		4 Social security tax withheld	
				5 Medicare wages and tips		6 Medicare tax withheld	
				7 Social security tips		8 Allocated tips	
d Control number				9		10 Dependent care benefits	
e Employee's first name and initial		Last name		Suff		11 Nonqualified plans	
				13 Statutory employee Retirement plan Third-party sick pay <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>		12a See instructions for box 12	
						12b	
						12c	
						12d	
f Employee's address and ZIP code				14 Other			
15 State Employer's state ID number		16 State wages, tips, etc.		17 State income tax		18 Local wages, tips, etc.	
						19 Local income tax	
						20 Locality name	

Form **W-2** Wage and Tax Statement

2020

Department of the Treasury—Internal Revenue Service  
For Privacy Act and Paperwork Reduction  
Act Notice, see the separate instructions.

Cat. No. 10134D

Copy A—For Social Security Administration. Send this entire page with  
Form W-3 to the Social Security Administration; photocopies are not acceptable.**Do Not Cut, Fold, or Staple Forms on This Page**

Various settings can be applied on form fields to enhance and customize their appearance. These are explained as below:

1. **Common Settings**

Name	Value Type	Example	Description
<b>type</b>	Enum string	{"type": "listbox"}	Indicates the type of form field.
	Standard Form Fields		(Mandatory field)
	<ul style="list-style-type: none"> <li>checkbox</li> <li>textbox</li> <li>listbox</li> <li>combobox</li> <li>radiobutton</li> <li>pushbutton</li> <li>signature</li> </ul>		
	Custom Form		

		<a href="#">Fields</a> <ul style="list-style-type: none"> <li>• text</li> <li>• date</li> <li>• time</li> <li>• tel</li> <li>• email</li> <li>• url</li> <li>• password</li> <li>• month</li> <li>• week</li> <li>• number</li> <li>• search</li> </ul>		
<b>alternateName</b>		String	{ "alternateName": "The alt name" }	Displays text which is helpful for a user while filling in the form field. Tooltips appear when the pointer hovers briefly over the form field.
<b>backgroundColor</b>		String	{ "backgroundColor": "#ffff00" } { "backgroundColor": "rgb(255, 178, 0)" } { "backgroundColor": "rgba(188, 100, 0, 255)" }	Indicates background color of the form field.
<b>border</b>	<b>width</b>	Yes	{ "border": { "width": 120 } }	Indicates width, color and style settings of the border for the form field.
	<b>color</b>	String	{ "border": { "color": "#ffff00" } } { "border": { "color": "rgb(255, 178, 0)" } } { "border": { "color": "rgba(188, 100, 0, 255)" } }	
	<b>style</b>	Enum string: <ul style="list-style-type: none"> <li>◦ none</li> <li>◦ solid</li> <li>◦ dashed</li> <li>◦ beveled</li> <li>◦ inset</li> <li>◦ underline</li> <li>◦ unknown</li> </ul>	{ "border": { "style": "dashed" } }	
<b>font</b>	<b>size</b>	Yes	{ "font": { "size": 18 } }	Indicates various font settings which can be used in the form field.
	<b>color</b>	String	{ "font": { "color": "#ffff00" } } { "font": { "color": "rgb(255, 178, 0)" } } { "font": { "color": "rgba(188, 100, 0, 255)" } }	
	<b>name</b>	String	{ "font": { "name": "sans-serif" } }	
	<b>bold</b>	Boolean	{ "font": { "bold": true } }	
	<b>italic</b>	Boolean	{ "font": { "italic": true } }	
<b>locked</b>		Boolean	{ "locked": true }	Indicates whether

			the user can change the properties of field or not.
<b>name</b>	String	{"name": "The field name"}	Indicates the unique name of field.
<b>readOnly</b>	Boolean	{"readOnly": true}	Indicates whether the user can change the value of field or not
<b>required</b>	Boolean	{"required": true}	Indicates whether the field must have a value.
<b>printed</b>	Boolean	{"printed": false}	Indicates whether to print the field when page is printed.
<b>hidden</b>	Boolean	{"hidden": true}	Indicates whether to display the field or not.
<b>mouseUp</b>	JsonObject	{"mouseUp": {"script": "fBox1 = this.getField(\"checkbox\")\n;\r\nfBox1.display = display.hidden", "submit": "http://localhost:80//myscript#FDF", "reset": {"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the mouse button is released in the active area of the field.
<b>mouseDown</b>	JsonObject	{"mouseDown": {"script": "fBox1 = this.getField(\"checkbox\")\n;\r\nfBox1.display = display.hidden", "submit": "http://localhost:80//myscript#FDF", "reset": {"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the mouse button is pressed in the active area of the field.
<b>mouseEnter</b>	JsonObject	{"mouseEnter": {"script": "fBox1 = this.getField(\"checkbox\")\n;\r\nfBox1.display = display.hidden", "submit": "http://localhost:80//myscript#FDF", "reset": {"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the mouse button enters the field's active area.
<b>mouseExit</b>	JsonObject	{"mouseExit": {"script": "fBox1 = this.getField(\"checkbox\")\n;\r\nfBox1.display = display.hidden", "submit": "http://localhost:80//myscript#FDF", "reset": {"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the mouse button exits the field's active area.


		;\r\nfBox1.display = display.hidden", "submit": "http://localhost:80//myscript#FDF", "reset": {"fieldNames": ["checkbox", "textbox"]}]}	performed in sequence when the mouse button exits the field's active area.
<b>onFocus</b>	JsonObject	{"onFocus": {"script": "fBox1 = this.getField(\"checkbox\") ;\r\nfBox1.display = display.hidden", "submit": "http://localhost:80//myscript#FDF", "reset": {"fieldNames": ["checkbox", "textbox"]}]}}	Indicates the actions to be performed in sequence when the annotation receives the input focus.
<b>onBlur</b>	JsonObject	{"onBlur": {"script": "fBox1 = this.getField(\"checkbox\") ;\r\nfBox1.display = display.hidden", "submit": "http://localhost:80//myscript#FDF", "reset": {"fieldNames": ["checkbox", "textbox"]}]}}	Indicates the actions to be performed in sequence when the annotation loses the input focus.
<b>format</b>	String	{"format": "event.value = (event.value * 100) + \" % \";"}	Indicates a JavaScript action to be performed before the field is formatted to display its current value. This action can modify the field's value before formatting.
<b>validate</b>	String	{"validate": "if (event.value < 0    event.value > 100) {\r\n" + "app.beep(0);\r\n" + "app.alert(\"Invalid value for field \" + event.target.name);\r\n" + "event.rc = false;\r\n" + "}"}	Indicates a JavaScript action to be performed when the field's value is changed. This action can check the new value for validity.
<b>calculate</b>	String	{"calculate": "var oil = this.getField(\"Oil\");\r\n" + "var filter = this.getField(\"Filter\");\r\n" + "event.value (oil.value + filter.value) * 1.0825;"}	Indicates a JavaScript action to be performed to recalculate the value of this field when that of another field changes.
<b>keystroke</b>	String	{"keystroke": "if (!event.willCommit)	Indicates a

			<pre>{\r\n"+"var f =this.getField (\"myPictures\");\r\n"+"var i =this.getIcon(event.change) ;\r\n"+"f.buttonSetIcon(i);\r\n"+"};"} </pre>	JavaScript action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This action can check the keystroke for validity and reject or modify it.
<b>autofocus</b>	Boolean		{"type": "password" , "autofocus": true}	Indicates whether a date field should automatically get focus when the page loads.
<b>disabled</b>	Boolean		{"type": "password" , "disabled": true}	Indicates whether a field is disabled or not.
<b>autocomplete</b>	Enum String		{ "type": "date" , "autocomplete": "bday" }	<p>Lets web developers specify if any user needs to provide assistance for automated filling of form field values, as well as guidance to the browser about what type of information is expected in the field.</p> <p>The behavior of this property depends on the browser implementation.</p>
	<ul style="list-style-type: none"> <li>o on</li> <li>o off</li> <li>o name</li> <li>o honorific-prefix</li> <li>o given-name</li> <li>o additional-name</li> <li>o family-name</li> <li>o honorific-suffix</li> <li>o nickname</li> <li>o email</li> <li>o username</li> <li>o new-password</li> <li>o current-password</li> <li>o one-time-code</li> <li>o organization-title</li> <li>o organization</li> </ul>	<ul style="list-style-type: none"> <li>o cc-number</li> <li>o cc-exp</li> <li>o cc-exp-month</li> <li>o cc-exp-year</li> <li>o cc-csc</li> <li>o cc-type</li> <li>o transaction-currency</li> <li>o transaction-amount</li> <li>o bday-day</li> <li>o language</li> <li>o bday</li> <li>o bday-day</li> <li>o bday-month</li> <li>o bday-year</li> <li>o sex</li> <li>o tel</li> <li>o tel-country-code</li> <li>o tel-national</li> </ul>		

	<ul style="list-style-type: none"> <li>◦ street-address</li> <li>◦ address-line1</li> <li>◦ address-line2</li> <li>◦ address-line3</li> <li>◦ address-level4</li> <li>◦ country</li> <li>◦ country-name</li> <li>◦ postal-code</li> <li>◦ cc-name</li> <li>◦ cc-given-name</li> <li>◦ cc-additional-name</li> <li>◦ cc-family-name</li> </ul>	<ul style="list-style-type: none"> <li>◦ tel-area-code</li> <li>◦ tel-local</li> <li>◦ tel-local-prefix</li> <li>◦ tel-local-suffix</li> <li>◦ tel-extension</li> <li>◦ impp</li> <li>◦ url</li> <li>◦ photo</li> </ul>		
--	---	---	--	--

## 2. JsonObject Settings

Name		Value Type	Example	Description
<b>script</b>		String	<code>{"script": "fBox1 = this.getField(\"checkbox\");\r\nfBox1.display = display.hidden"}</code>	Indicates an action which causes a script to be compiled and executed by the JavaScript interpreter.
<b>submit</b>		String	<code>{"submit": "http://localhost:80//myscript#FDF"}</code>	Indicates an action to transmit the names and values of selected interactive form fields to a specified uniform resource locator (URL), presumably the address of a Web server that will process them and send back a response.
<b>reset</b>	<b>fieldNames</b>	Yes	<code>{"fieldNames": ["checkbox", "textbox"]}</code>	Indicates the list of names of fields that should be processed (or excluded from processing) by this action. If empty then all fields will be processed.
	<b>exclude</b>	Boolean	<code>{"exclude": true}</code>	Indicates whether to exclude the fields specified in fieldNames from processing (by default, this property is false and the specified fields are included).

 **Note:** Snippets of JavaScript code needs to be escaped.

## 3. Checkbox Form Field Settings

Name	Value Type	Example	Description
<b>checkStyle</b>	Enum string: <ul style="list-style-type: none"> <li>• check</li> <li>• circle</li> <li>• cross</li> <li>• diamond</li> <li>• square</li> <li>• Star</li> </ul>	{"checkStyle": "circle"}	Indicates the style of check mark.
<b>value</b>	Boolean	{"value": true}	Indicates the value of Checkbox. (If the value is missing, GcExcel automatically tries to convert the cell's value to Boolean, and then, set it to the property after processing the template.)
<b>defaultValue</b>	Boolean	{"defaultValue": false}	Indicates the default value of Checkbox.

#### 4. Textbox Form Field Settings

Name	Value Type	Example	Description
<b>value</b>	String	{"value": "Hunter"}	Indicates the value of Textbox.
<b>defaultValue</b>	String	{"defaultValue": "Input your name!"}	Indicates the default value of Textbox.
<b>combo</b>	Boolean	{"combo": true}	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
<b>password</b>	Boolean	{"password": true}	Indicates whether the field is intended for entering a secure password that should not be visible on the screen.
<b>spellcheck</b>	Boolean	{"spellcheck": false}	Indicates whether the text entered in the field is spell-checked.
<b>scrollable</b>	Boolean	{"scrollable": false}	Indicates whether the field is scrollable to accommodate more text than it fits within its annotation rectangle.
<b>maxlength</b>	Integer	{"maxlength": 10}	Indicates the maximum length of the field's text, in characters.
<b>multiline</b>	Boolean	{"multiline": true}	Indicates whether the field can contain multiple lines of text.
<b>justification</b>	Enum string: <ul style="list-style-type: none"> <li>• left</li> <li>• center</li> <li>• right</li> </ul>	{"justification": "center"}	Indicates the justification to be used while displaying the field's text.



**Note:** GcExcel also supports custom form fields like date, email, password, month etc. which are inherited from Textbox form field. To know more about these, refer [Custom Form Fields](#).

## 5. Listbox Form Field Settings

Name	Value Type	Example	Description
<b>value</b>	String Array	{"value": ["US", "UK"]}	Indicates the value of Listbox.
<b>defaultValue</b>	String Array	{"defaultValue": ["US", "UK"]}	Indicates the default value of Listbox.
<b>commitOnSelChange</b>	Boolean	{"commitOnSelChange": true}	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
<b>selectedIndex</b>	Integer	{"selectedIndex": 0}	Indicates the indexes of selected item.
<b>sort</b>	Boolean	{"sort": true}	Indicates whether the field's option items should be sorted alphabetically.
<b>spellCheck</b>	Boolean	{"spellCheck": true}	Indicates whether the text entered in the field is spell-checked.
<b>selectedIndexes</b>	Integer Array	{"selectedIndexes": [0, 2, 5]}	Indicates the indexes of selected items.
<b>multiSelect</b>	Boolean	{"multiSelect": true}	Indicates whether more than one of the field's option items may be selected simultaneously.
<b>exportValue</b>	String	{"exportValue": "TheResult"}	Indicates the export value of Listbox field.

## 6. Combobox Form Field Settings

Name	Value Type	Example	Description
<b>value</b>	String Array	{"value": ["US", "UK"]}	Indicates the value of Combobox.
<b>defaultValue</b>	String Array	{"defaultValue": ["US", "UK"]}	Indicates the default value of Combobox.
<b>commitOnSelChange</b>	Boolean	{"commitOnSelChange": true}	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
<b>selectedIndex</b>	Integer	{"selectedIndex": 0}	Indicates the indexes of selected item.
<b>sort</b>	Boolean	{"sort": true}	Indicates whether the field's option items should be sorted alphabetically.
<b>spellCheck</b>	Boolean	{"spellCheck": true}	Indicates whether the text entered in the field is spell-checked.
<b>editable</b>	Boolean	{"editable": true}	Indicates whether the Combobox includes an editable text box as well as a drop-down list.



## 7. Radiobutton Form Field Settings

Name	Value Type	Example	Description
<b>checkStyle</b>	Enum string: <ul style="list-style-type: none"> <li>• check</li> <li>• circle</li> <li>• cross</li> <li>• diamond</li> <li>• square</li> <li>• Star</li> </ul>	{"checkStyle": "circle"}	Indicates the style of check mark.
<b>groupName</b>	String	{"groupName": "Teams"}	Indicates the name of radio button group.  Radio buttons with the same group name are added in the same group.  (If the value is missing, GcExcel automatically adds radio buttons expanded from the same template cell to the same group after processing the template)
<b>radiosInUnison</b>	Boolean	{"radiosInUnison": true}	Indicates whether a group of radio buttons within a radio button field that use the same value for the on state will turn on and off in unison.  If one is checked, they are all checked. If clear, the buttons are mutually exclusive (the same behavior as HTML radio buttons).
<b>checkedChoice</b>	String	{"checkedChoice": "Team5"}	Indicates the value of checked option.
<b>defaultCheckedChoice</b>	String	{"defaultCheckedChoice": "Team1"}	Indicates the value of checked option when the user first opens the form.

## 8. Pushbutton Form Field Settings

Name	Value Type	Example	Description
<b>highlighting</b>	Enum string: <ul style="list-style-type: none"> <li>• none</li> <li>• invert</li> <li>• outline</li> <li>• push</li> </ul>	{"highlighting": "outline"}	Indicates the annotation's highlighting mode.
<b>caption</b>	String	{"caption": "Push"}	Indicates the button's caption.
<b>image</b>	Base64 String	{"image": "The base64 image data."}	Indicates the button's image.
<b>captionImageRelation</b>	Enum string: <ul style="list-style-type: none"> <li>• captionOnly</li> </ul>	{"captionImageRelation": "captionBelowIcon"}	Indicates the positioning of button's caption relative to image.

		<ul style="list-style-type: none"> <li>• imageOnly</li> <li>• captionBelowIcon</li> <li>• captionAboveIcon</li> <li>• captionAtRight</li> <li>• captionAtLeft</li> <li>• captionOverlaid</li> </ul>		
<b>downCaption</b>		String	{"downCaption": "Push Down"}	Indicates the button's caption when user presses the button.
<b>downImage</b>		Base64 String	{"downImage": "The base64 image data."}	Indicates the button's image when user presses the button.
<b>rolloverCaption</b>		String	{"rolloverCaption": "Rollover"}	Indicates the button's caption when the user rolls the cursor into its active area without pressing the mouse button.
<b>rolloverImage</b>		Base64 String	{"rolloverImage": "The base64 image data."}	Indicates the button's image when the user rolls the cursor into its active area without pressing the mouse button.
<b>imageScale</b>	mode	Yes	{"imageScale": {"mode": "bigger"}}	Indicates the scaling mode.
	proportional	Boolean	{"imageScale": {"proportional": true}}	Indicates whether an image should be scaled proportionally.
	x	Float	{"imageScale": {"proportional": true, "x": 0.6}}	Indicates the position of an image.
	y	Float	{"imageScale": {"proportional": true, "y": 0.8}}	<p>The two numbers between 0.0 and 1.0 indicates the fraction of leftover space to allocate at the left and bottom of an image. A value of (0.0, 0.0) positions the image at the bottom-left corner of the button rectangle. A value of (0.5, 0.5) centers it within the rectangle.</p> <p>This value is used only if the image is scaled proportionally.</p>
	ignoreBorder	Boolean	{"imageScale": {"ignoreBorder": true}}	Indicates whether a button's appearance should be scaled to fit fully within the bounds of the annotation without taking into consideration the line width of the border.

## 9. Signature Form Field Settings

Name	Value Type	Example	Description
<b>lockType</b>	<b>Enum string:</b> <ul style="list-style-type: none"> <li>• all</li> </ul>	{"lockType": "specifiedOnly"}	Indicates the type of locked fields.

	<ul style="list-style-type: none"><li>• specifiedOnly</li><li>• allButSpecified</li></ul>		
<b>fieldNames</b>	String Array	{"fieldNames": ["signerName", "time"]}	Indicates the list of field names which should be included or excluded from processing depending on lockType property.
<b>LockedFields</b>	Boolean	{"LockedFields": true}	Indicates whether to lock the fields when SignatureFormField is signed or not.



**Note:** GcExcel Template generates only digital signature fields in PDF documents. If you want to add signatures on signature fields, you need to use GcPDF or PDFBox to process.

Apart from the above mentioned standard PDF form fields, GcExcel also supports custom form input types to generate PDF forms. Refer to [Custom Form Input Types](#) for more information.

## Custom Form Input Types

Along with the support of [Standard PDF form fields](#) in GcExcel Templates, it also supports custom form input types in PDF forms which allow you to fill PDF forms easily and conveniently. It supports adding HTML5 custom input types to PDF documents. These custom form input types are not supported by standard PDF specification and hence these can only be opened, viewed and filled in [GcDocs Pdf Viewer](#) (not in Acrobat or other PDF viewers). These custom form input types are inherited from "textbox" field and are mentioned below:

- text
- date
- time
- tel
- email
- url
- password
- month
- week
- number
- search

You can also define validation settings for these custom form input types which provide users with feedback on their form submission before sending it to server. For example, any custom form input type is a required field, password needs to be of minimum 8 characters, email needs to follow a certain pattern etc. If a certain validation is not followed and a validation message is set, the form will display the defined validation message on submission. These validations can be defined in custom form input types by using below validation types:

- validationmessage
- validateoninput
- minlength
- maxlength
- required
- pattern
- min
- max

A few examples of custom form input types with validations are given below:

- `{{(form={"type":"url", "autocomplete":"url", "validateoninput": true, "backgroundcolor": "rgb(191, 207, 255)", "border": {"color": "#000000"}})}}`
- `{{(form={"type":"email", "autocomplete":"email", "validateoninput": true, "backgroundcolor": "rgb(191, 207, 255)", "border": {"color": "#000000"}})}}`
- `{{(form={"type":"password", "autocomplete":"off", "validateoninput": true, "placeholder": "4 to 8 characters", "pattern": "^(?=.*\\d){4,8}$", "validationmessage": "The password must be between 4 and 8 characters.", "backgroundcolor": "rgb(191, 207, 255)", "border": {"color": "#000000"}})}}`

These custom form input types are inherited from 'textbox' field, hence they inherit all the 'textbox' field settings which can be referred [here](#). Also, there are some [Common Settings](#) which can be applied to all the custom form input types. The settings specific to each field are explained below:

### Tel or Password or URL Custom Form Input Type Settings

Property	Value	Example	Description
<b>pattern</b>	String	<code>{"type": "tel", "pattern": "[0-9]{3}-[0-9]{2}-[0-9]{3}"}</code>	Pattern the value must match to be valid
<b>placeholder</b>	String	<code>{"type": "password", "placeholder": "4 to 8 characters"}</code>	Sets or returns the value of the placeholder attribute of an tel field
<b>maxlength</b>	Integer	<code>{"type": "password", "minlength": 4, "maxlength": 8}</code>	Maximum length (number of characters) of value
<b>minlength</b>	Integer	<code>{"type": "password", "minlength": 4, "maxlength": 8}</code>	Minimum length (number of characters) of value

### Email Custom Form Input Type Settings

Property	Value	Example	Description
<b>multiple</b>	Boolean	<code>{"type": "email", "multiple": true}</code>	Sets or returns whether a user is allowed to enter more than one email address in the email field.
<b>pattern</b>	String	<code>{"type": "email", "pattern": "\\S+@\\S+\\.\\S+"}</code>	Sets or returns the value of the pattern attribute of an email field.
<b>placeholder</b>	String	<code>{"type": "email", "placeholder": "example@xxx.com"}</code>	Sets or returns the value of the placeholder attribute of an email field.

### Text Custom Form Input Type Settings

Property	Value	Example	Description
<b>maxlength</b>	Integer	<code>{"type": "text", "minlength": 4, "maxlength": 8}</code>	Maximum length (number of characters) of value
<b>minlength</b>	Integer	<code>{"type": "text", "minlength": 4, "maxlength": 8}</code>	Minimum length (number of characters) of value

<b>pattern</b>	String	{"type": "text" , "pattern": "\\S+@\\S+\\.\\S+"}	Sets or returns the value of the pattern attribute of field
<b>placeholder</b>	String	{"type": "text" , "placeholder": "Input your name!"}	Sets or returns the value of the placeholder attribute of field
<b>spellcheck</b>	Boolean	{"type": "text" , "spellcheck": true}	Whether the element may be checked for spelling errors

### Search Custom Form Input Type Settings

Property	Value	Example	Description
<b>maxlength</b>	Integer	{"type": "search" , "minlength": 4, "maxlength": 8}	Maximum length (number of characters) of value
<b>minlength</b>	Integer	{"type": "search" , "minlength": 4, "maxlength": 8}	Minimum length (number of characters) of value
<b>placeholder</b>	String	{"type": "search" , "placeholder": "Search..."}	Sets or returns the value of the placeholder attribute of field
<b>spellchecker</b>	Boolean	{"type": "search" , "spellcheck": true}	Whether the element may be checked for spelling errors

### Validation Settings

The following table explains the validation settings provided for custom custom form input types.

Property	Value	Description
<b>validateonmessage</b>	String	Localized validation message
<b>validateoninput</b>	Boolean	Indicates whether validation should be performed immediately during user input
<b>maxlength</b>	Number	Maximum number of characters to be accepted
<b>minlength</b>	Number	Minimum number of characters which can be considered valid
<b>required</b>	Boolean	Indicates whether the form filling is required or not
<b>pattern</b>	String	Regular expression that must be matched by the entered value to pass constraint validation
<b>max</b>	Number	Maximum value to accept for this input
<b>min</b>	Number	Minimum value to accept for this input

### Settings Supported by Custom Form Input Types

The following table provides consolidated information about settings supported by different custom form input types.

Attribute	Input Field Type	Description
<b>autocomplete</b>	All	Input type.

<b>autofocus</b>	All	Automatically focus the form control when the page is loaded.
<b>defaultvalue</b>	All	The default value.
<b>disabled</b>	All	Whether the form control is disabled.
<b>displayname</b>	All	Text label for the input control. Applicable only if the field appears in the Form Filler dialog box.
<b>min</b>	number and date	Minimum value to accept for the input.
<b>max</b>	number and date	Maximum value to accept for the input.
<b>maxlength</b>	password, search, tel, text and url	Maximum length (number of characters) of value.
<b>minlength</b>	password, search, tel, text and url	Minimum length (number of characters) of value
<b>multiline</b>	text	Set this property to true if you want to use the textarea as a user input element.
<b>multiple</b>	email	Boolean. Whether to allow multiple values or not.
<b>pattern</b>	password, text and tel	Pattern value must match to be valid.
<b>placeholder</b>	password, search, tel, text and url	Text that appears in the form control when it has no value set.
<b>readonly</b>	All	Boolean. The value is not editable.
<b>required</b>	All	Boolean. A value is required or must be check for the form to be submittable.
<b>spellcheck</b>	search and text	Whether the element may be checked for spelling errors.
<b>type</b>	All	Type of form control.
<b>validateonmessage</b>	All	Localized validation message.
<b>validateoninput</b>	All	Indicates whether validation should be performed immediately during user input.

The following Excel template shows various input types and settings supported with GcExcel templates. As can be observed, these fields are very common and makes PDF form filling very convenient. The 'date' and 'time' fields will provide date picker and time picker dropdown UI in the generated PDF form when viewed in GcDocs Pdf Viewer. The 'tel with pattern' field defines a pattern which will be matched with the user input while filling the form. The 'password with minlength' and 'password with maxlength' defines the character limit which can be used while setting a password. The validations applied to custom form input types make the form filling more meaningful and useful.


You can also download the **Excel template layout** from [here](#).

B	C	D	E	F	G	H	I	J	K	L	M
date:	{{{form={"type":"date", "autocomplete":"bday", "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
time:	{{{form={"type":"time", "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
datetime-local*exper	{{{form={"type":"date", "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
tel:	{{{form={"type":"tel", "validateoninput": true, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
tel with pattern:	{{{form={"type":"tel", "pattern": "^[\\+]?([0-9]{3})?[-\\s\\.].?0-										
email:	{{{form={"type":"email", "autocomplete":"email", "validateoninput": true, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
email with pattern:	{{{form={"type":"email", "autocomplete":"email", "validateoninput": true, "pattern": "\\S+@\\S+\\.\\S+", "validationmessage": "The value entered is not a valid email address."}}}}}										
url:	{{{form={"type":"url", "autocomplete":"url", "validateoninput": true, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
password:	{{{form={"type":"password", "autocomplete":"new-password", "validateoninput": true, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
password with minle	{{{form={"type":"password", "autocomplete":"new-password", "validateoninput": true, "placeholder": "minlength=8", "minlength": 8, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
password with maxle	{{{form={"type":"password", "autocomplete":"off", "validateoninput": true, "placeholder": "maxlength=4", "maxlength": 4, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
password with patte	{{{form={"type":"password", "autocomplete":"off", "validateoninput": true, "placeholder": "4 to 8 characters", "pattern": "^(?=.*[\\d]).{4,8}\$", "validationmessage": "Password must be 4 to 8 characters long and contain at least one digit."}}}}}										
text with autofocus:	{{{form={"type":"text", "autofocus": true, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
text with required:	{{{form={"type":"text", "required": true, "validateoninput": true, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
text, spellcheck='true'	{{{form={"type":"text", "defaultValue": "mistaake", "spellcheck": true, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
text, spellcheck='false'	{{{form={"type":"text", "defaultValue": "mistaake", "spellcheck": false, "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
month:	{{{form={"type":"month", "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
week:	{{{form={"type":"week", "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
number:	{{{form={"type":"number", "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										
search:	{{{form={"type":"search", "backgroundcolor": "rgb(191, 207, 255)", "border":{"color": "#000000"}}}}}										

After GcExcel processes the above template and exports it to a PDF document, the PDF form will look like below in GcDocs PDF Viewer:

The screenshot displays a PDF form with a sidebar on the left containing navigation icons. The main content area shows a list of form fields with their respective labels and values:

date:	dd - mm - yyyy
time:	-- : --
datetime-local*experimental:	dd - mm - yyyy
tel:	
tel with pattern:	
email:	
email with pattern:	
url:	
password:	
password with minlength:	minlength=8
password with maxlength:	maxlength=4
password with pattern:	4 to 8 characters
text with autofocus:	
text with required:	
text, spellcheck='true':	mistaake
text, spellcheck='false':	mistaake
month:	-----, ----
week:	Week --, ----
number:	
search:	

 **Note:** The PDF form with custom input fields can only be filled in GcDocs PDF Viewer. You can also customize the custom form input type settings by using GcDocs PDF Viewer's [Form Filler](#) feature.

## Charts

Excel charts can be added in Template layout which are visible in the Excel report. It is very useful as charts are often used in Excel reports to display graphical data. Along with that, it provides an advantage that the final chart will always be updated with the latest data, when the template is processed.

The Excel charts are bound with template cells by specifying the series name, series value and axis labels in the template layout. While processing the template layout, the chart is bound to the data, and the Excel report is generated with the chart displaying final data. A chart can be placed in a worksheet with its data or in another worksheet too.

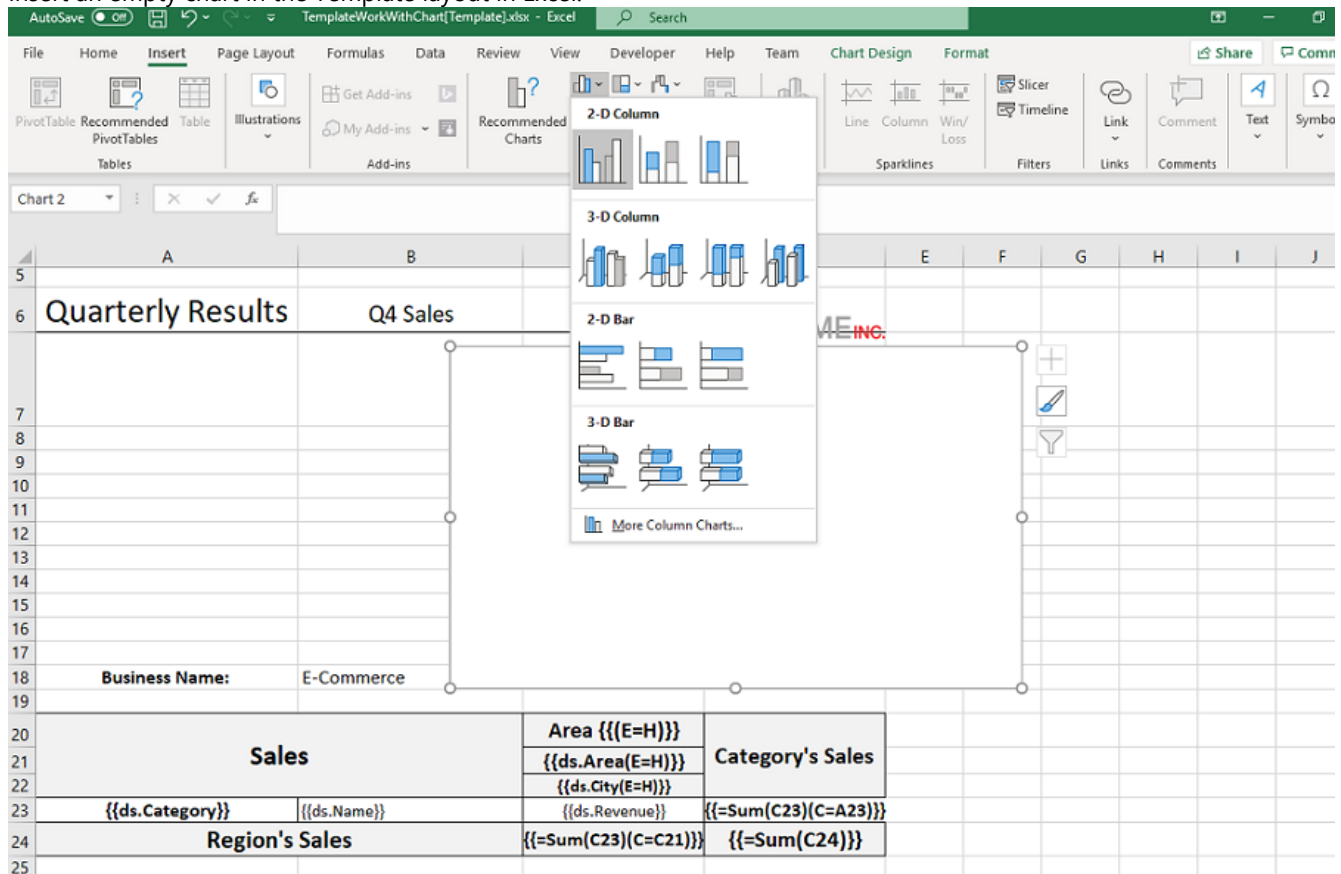


Follow the steps mentioned below to add chart in a template layout and configure its data to template cells:

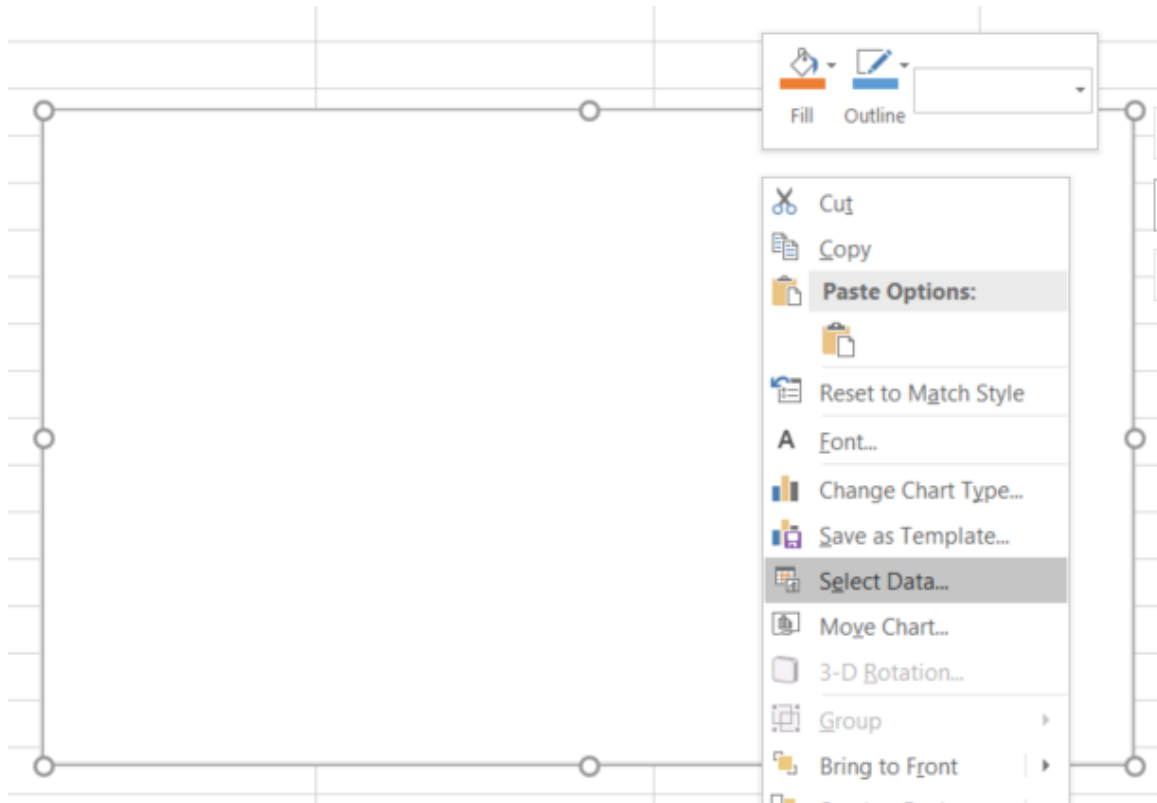
Here we are adding a chart to a 'Quarterly Sales Data' report which displays the sales of electronic goods in different areas of North and South America. The chart in the template configures the name of an electronic item as series name, revenue as series value and the city in which sales have been done as axis labels to display the sales data in a graphical manner.

You can also download the **Excel template layout** used in below example.

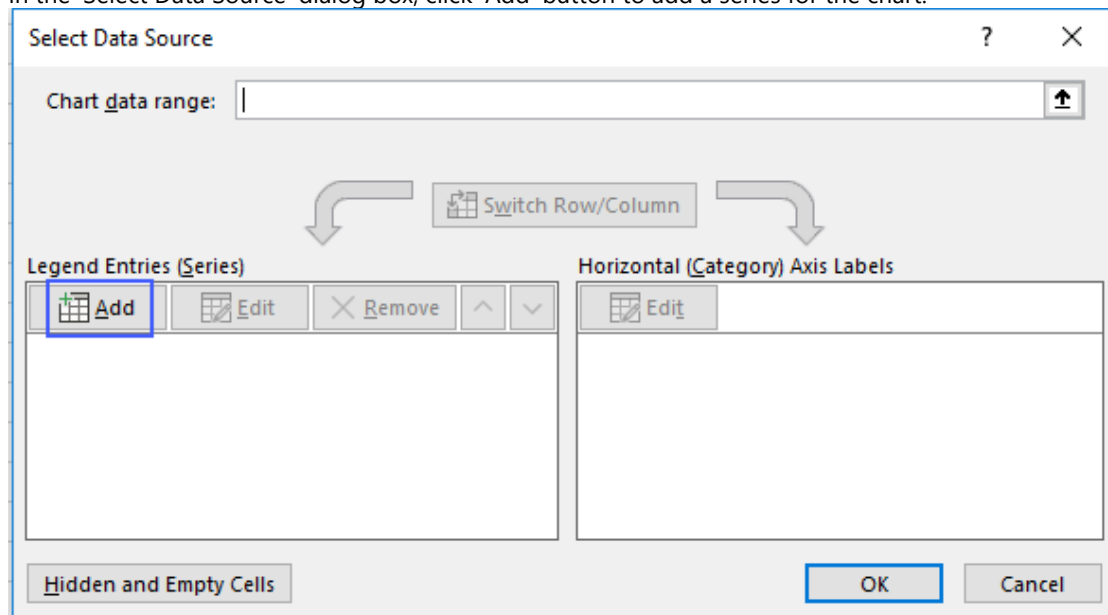
1. Insert an empty chart in the Template layout in Excel.



2. Right click on the chart and choose 'Select Data' from context menu



3. In the 'Select Data Source' dialog box, click 'Add' button to add a series for the chart.



4. In the 'Edit Series' dialog box, click in 'Series Name' and then select 'ds.Salesman' field of the template layout as salesman field is being used as series for the chart.

Sales		Area {{{E=H}}}	Category's Sales
		{{ds.Area(E=H)}}	
		{{ds.City(E=H)}}	
{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C23)(C=A23)}}
Region's Sales		{{=Sum(C23)(C=C21)}}	{{=Sum(C24)}}

Edit Series

?

×

Series name:

=Sales!\$B\$23

↑

Select Range

Series values:

= {1}

↑

= 1

OK

Cancel

5. Next, click in 'Series Values' and then select 'ds.Sales' field of the template layout as sales field is being used as the value for the series of the chart.

Sales		Area {{{E=H}}}	Category's Sales
		{{ds.Area(E=H)}}	
		{{ds.City(E=H)}}	
{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C23)(C=A23)}}
Region's Sales		{{=Sum(C23)(C=C21)}}	{{=Sum(C24)}}

Edit Series

?

×

Series name:

=Sales!\$B\$23

↑

= {{ds.Name}}

Series values:

=Sales!\$C\$23

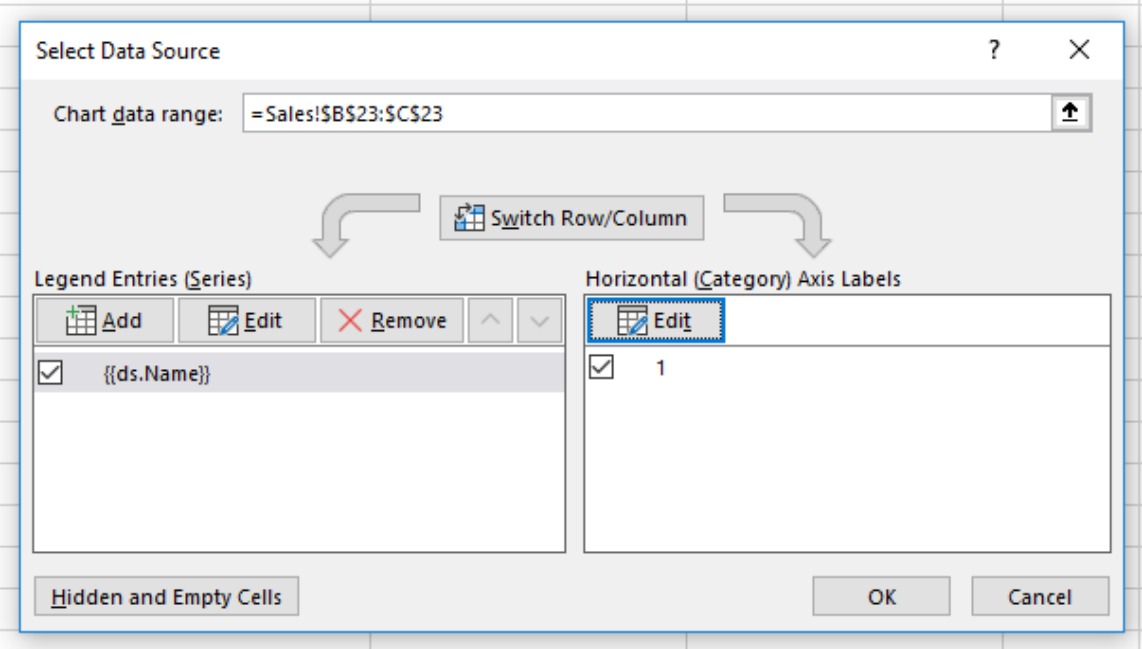
↑

= 1

OK

Cancel

6. Click on the 'Edit' button highlighted in the below screenshot.



7. In the 'Axis Labels' dialog box, click in 'Axis Label Range' and then select 'ds.Product' field of the template layout as products field is being used as axis label of the chart.

Sales		Area {{{E=H}}}	Category's Sales
		{{ds.Area(E=H)}}	
		{{ds.City(E=H)}}	
{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C23)(C=A23)}}
Region's Sales		{{=Sum(C23)(C=C21)}}	{{=Sum(C24)}}

Axis Labels

Axis label range:

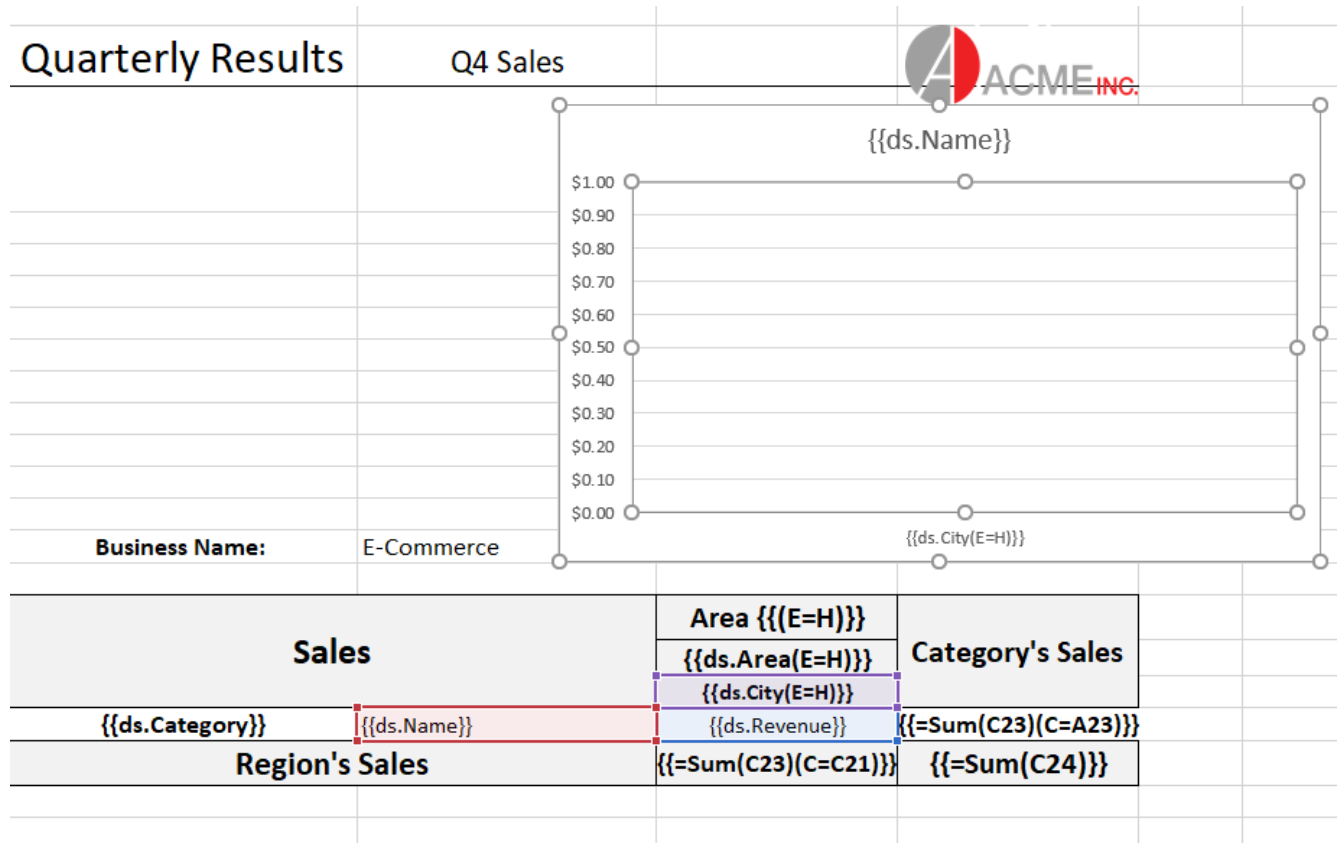
=Sales!\$C\$22

Select Range

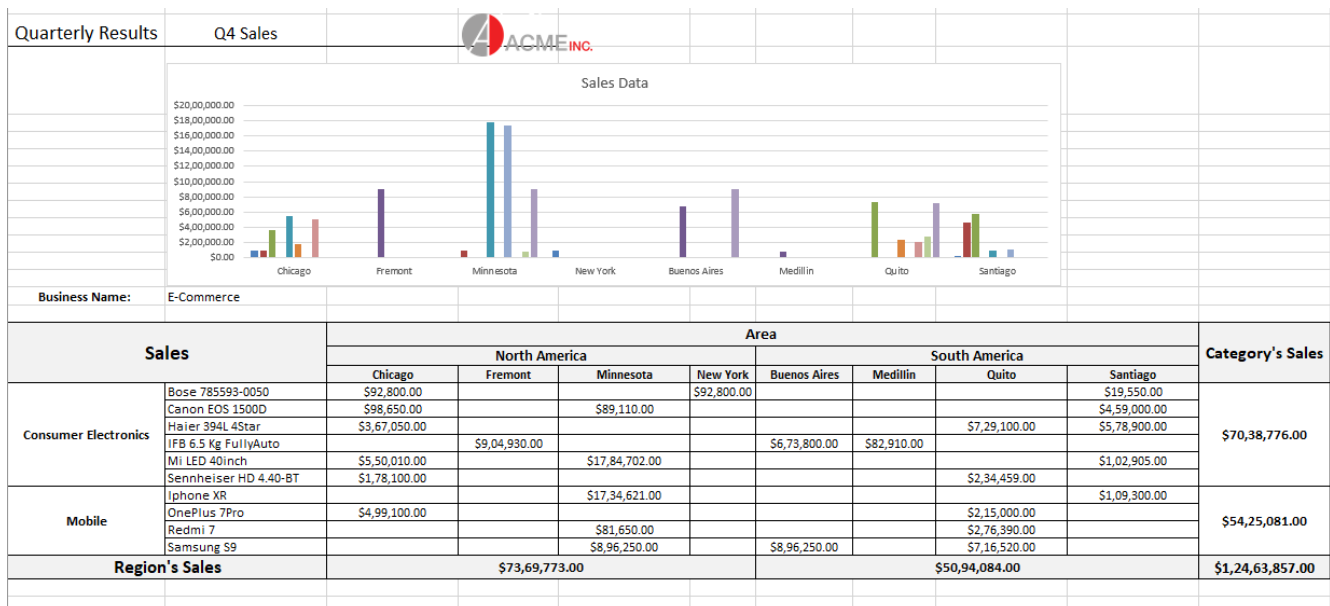
OK

Cancel

8. Click Ok to submit the data configuration.



After processing the template layout, the Excel report will look like below.

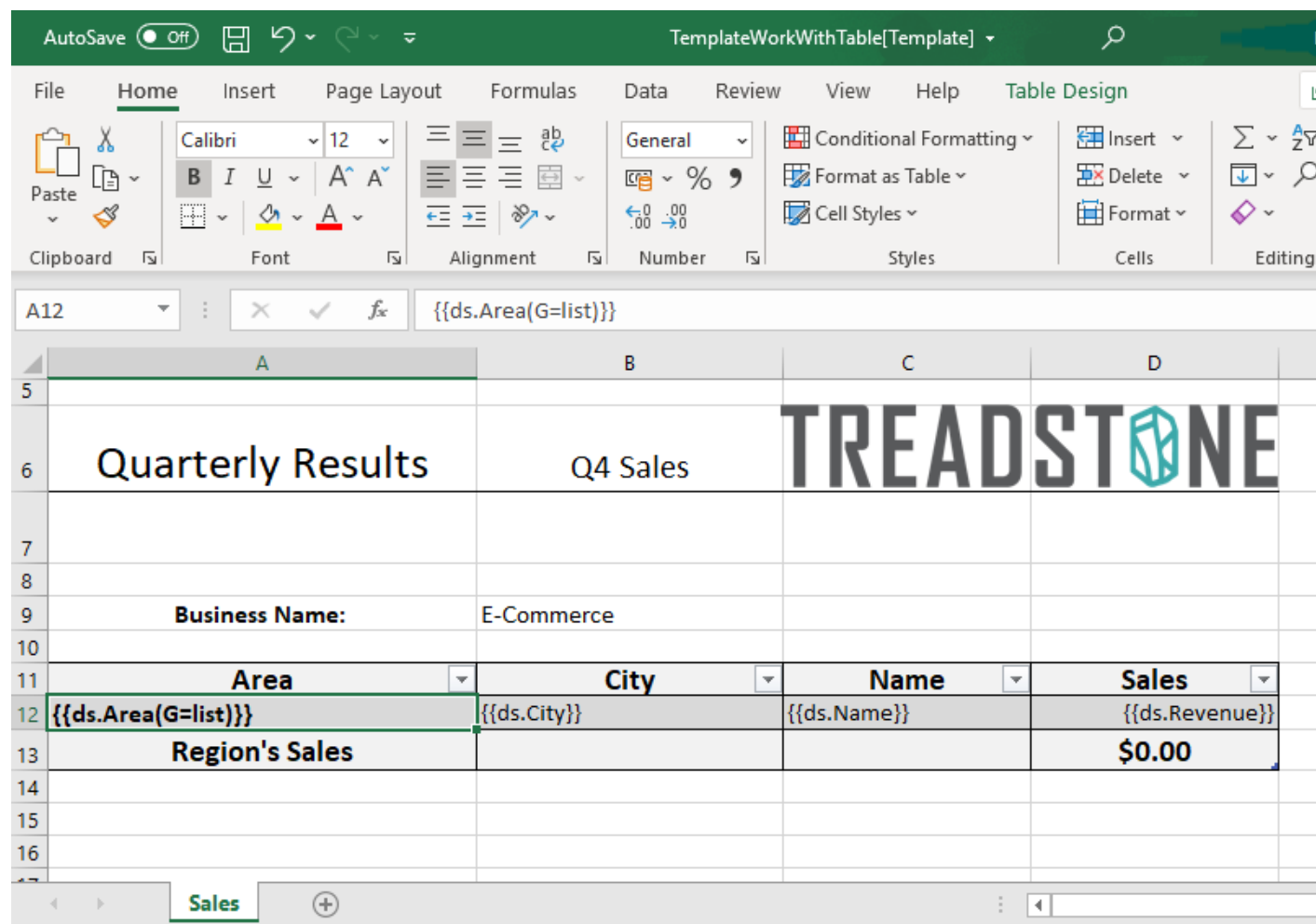


## Tables


Tables are essential to depict large amounts of data in an organized way. GcExcel supports using Excel tables in template layouts where various operations can also be performed on it like filtering, sorting etc.

This template example lists sales information for different areas grouped as a list. The template cells are defined within the table layout.

The below image displays a template layout where a table is used and template cells are defined inside the table. You can also download the **Excel template layout** from [here](#).




After GcExcel processes the template layout, the Excel report will look like below:

A	B	C	D
Quarterly Results	Q4 Sales	TREADSTONE 	
Business Name:	E-Commerce		
Area	City	Name	Sales
North America	Chicago	Bose 785593-0050	\$92,800.00
North America	New York	Bose 785593-0050	\$92,800.00
South America	Santiago	Bose 785593-0050	\$19,550.00
North America	Chicago	Canon EOS 1500D	\$98,650.00
North America	Minnesota	Canon EOS 1500D	\$89,110.00
South America	Santiago	Canon EOS 1500D	\$4,59,000.00
North America	Chicago	Haier 394L 4Star	\$3,67,050.00
South America	Quito	Haier 394L 4Star	\$7,29,100.00
South America	Santiago	Haier 394L 4Star	\$5,78,900.00
North America	Fremont	IFB 6.5 Kg FullyAuto	\$9,04,930.00
South America	Buenos Aires	IFB 6.5 Kg FullyAuto	\$6,73,800.00
Region's Sales			\$41,05,690.00

An Excel table can be incorporated in a template layout in two ways:

1. **Template cells inside an Excel table:** You can insert a table in Excel's template layout and define template cells inside it, as shown in above screenshot. The table is resized according to the expanded data after processing the template in GcExcel.
2. **Excel table inside a template cell's range:** You can define a template cell with [Range property](#) and insert a table anywhere within that range. The table is copied according to the expansion data after processing the template in GcExcel.

 **Note:** Table formulas are also supported in template cells.

## Limitations

- In GcExcel Templates, the default group type is "Merge", which is not supported in case of tables. Hence, you should explicitly set the group type to any other value except "Merge".
- **Excel table inside a template cell's range:** The complete range of table should be included in the template cell's Range property. For example, if a table occurs in the range C5:D8, the template cell should have the "Range(R)" property, for example: `{{ds.Area(R=C5:D8)}}`, to include table inside cell range C5:D8. However, for [sheet name template](#), any table in the current sheet is included by default. So, it doesn't need to set "Range" property.
- **Template cells inside an Excel table:** If [sheet name template](#) is also used along with table, there might be layout issues while expanding the template and the table might be moved to an incorrect location. Hence, you should convert table to cell range before processing.

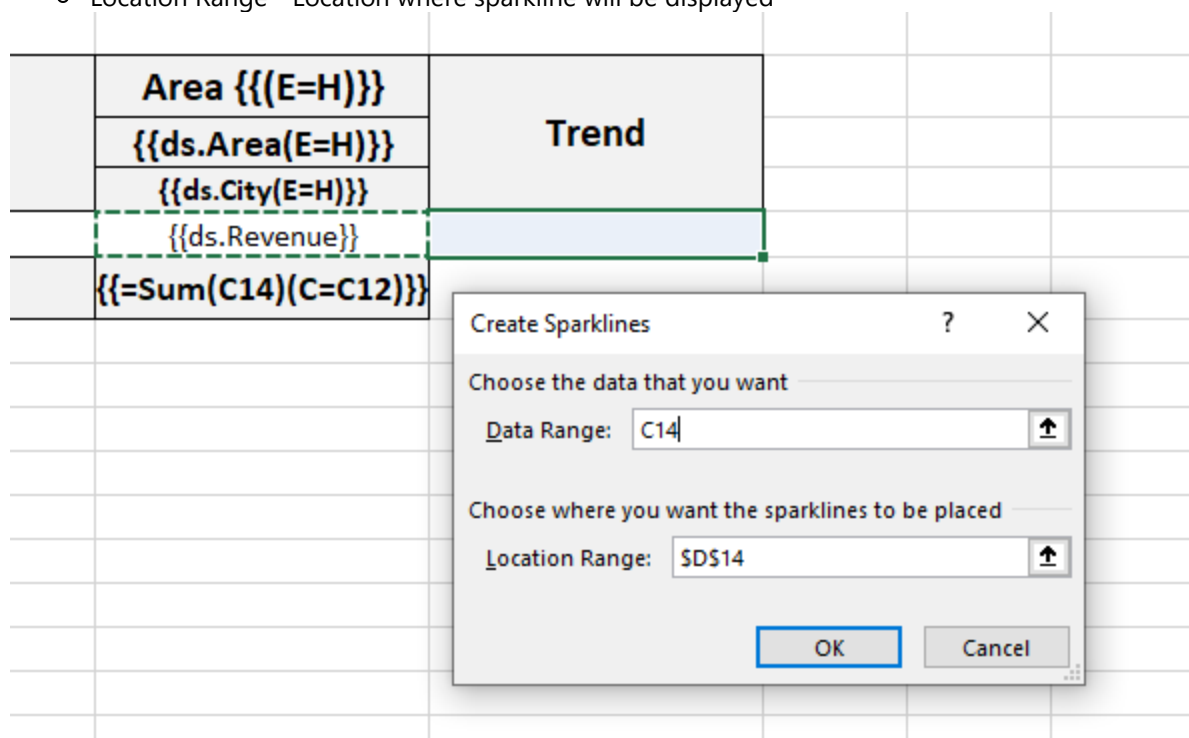
## Sparklines

GcExcel supports adding sparklines in template layout, which are visible in the Excel report generated after processing the template.

Follow the steps mentioned below to add a sparkline in template layout and configure its data to template cells:

You can also download the **Excel template layout** used in below example.

1. Insert a sparkline in Excel's template layout by choosing **Menu | Insert | Sparklines**.
2. In the "Create Sparklines" dialog box, choose a template cell as:
  - Data Range - Data to be displayed by sparkline
  - Location Range - Location where sparkline will be displayed



After GcExcel processes the template layout, the Excel report will look like below:

Business Name:		E-Commerce									
Sales		Area								Trend	
		North America				South America					
		Chicago	Fremont	Minnesota	New York	Buenos Aires	Medillin	Quito	Santiago		
Consumer Electronics	Bose 785593-0050	\$92,800.00			\$92,800.00				\$19,550.00	<div><div></div><div></div><div></div></div>	
	Canon EOS 1500D	\$98,650.00		\$89,110.00					\$4,59,000.00	<div><div></div><div></div><div></div></div>	
	Haier 394L 4Star	\$3,67,050.00						\$7,29,100.00	\$5,78,900.00	<div><div></div><div></div><div></div></div>	
	IFB 6.5 Kg FullyAuto		\$9,04,930.00			\$6,73,800.00	\$82,910.00			<div><div></div><div></div><div></div></div>	
	Mi LED 40inch	\$5,50,010.00		\$17,84,702.00					\$1,02,905.00	<div><div></div><div></div><div></div></div>	
Mobile	Sennheiser HD 4.40	\$1,78,100.00						\$2,34,459.00		<div><div></div><div></div><div></div></div>	
	Iphone XR			\$17,34,621.00					\$1,09,300.00	<div><div></div><div></div><div></div></div>	
	OnePlus 7Pro	\$4,99,100.00						\$2,15,000.00		<div><div></div><div></div><div></div></div>	
	Redmi 7			\$81,650.00				\$2,76,390.00		<div><div></div><div></div><div></div></div>	
	Samsung S9			\$8,96,250.00		\$8,96,250.00		\$7,16,520.00		<div><div></div><div></div><div></div></div>	
Region's Sales		\$73,69,773.00				\$50,94,084.00					

**Note:** In Excel report, the sparkline whose data range and location range are in the same column is displayed as a 'vertical' sparkline, otherwise, as 'horizontal' sparkline.



## Data Source Binding

Once the template layout is prepared in Excel including bound fields, expressions, formula and sheet name fields, these fields need to be bound to a data source. You can add a data source using the **AddDataSource** method and bind the data with template using the **ProcessTemplate** method. This will populate the data from datasource in the template fields to generate the Excel report.

Also, you can use multiple data sources or multiple data tables within a data source and populate data through them. The syntax requires you to define the object of the data source followed by the data field. For example, the below template layout merges data from two data sources, the employee information from one data table and Department information from another table.

Employee Name	Employee ID	Department	Department HOD Name
{{emp.name(G = list, S = None)}}	{{emp.id}}	{{emp.department.department}}	{{emp.department.hodname}}

GcExcel supports the below data sources while using templates:

### DataTable

A single table which has collection of rows and columns from any type of database

### Template syntax

[Alias of data source].[Column name]

For example:

{{ds.ID}}

{{ds.Name}}

### Bind DataTable datasource

C#

```
var datasource = new System.Data.DataTable();
datasource.Columns.Add(new DataColumn("ID", typeof(Int32)));
datasource.Columns.Add(new DataColumn("Name", typeof(string)));
datasource.Columns.Add(new DataColumn("Score", typeof(Int32)));
datasource.Columns.Add(new DataColumn("Team", typeof(string)));

...//Init data
```

```
//Add data source  
workbook.AddDataSource("ds", datasource);
```

## DataSet

A collection of one or more DataTables

### Template syntax

[Alias of data source].[Table name].[Column name]

For example:

{{ds.Table1.ID}}

{{ds.Table2.Team}}

### Bind DataSet datasource

```
C#  
  
var dTable1 = new System.Data.DataTable();  
var dTable2 = new System.Data.DataTable();  
  
...//Init data  
  
var datasource = new System.Data.DataSet();  
datasource.Tables.Add(team1);  
datasource.Tables.Add(team2);  
  
//Add data source  
workbook.AddDataSource("ds", datasource);
```

## Custom Object

A user-defined object from user code or serialized object of JSON String/File/XML, etc. GcExcel Template supports any data source that can be serialized as a custom object.

### Template syntax

[Alias of data source].[Field name]

or

[Alias of data source].[Property name]

For example:

```
{{ds.Records.Area}}
```

```
{{{ds.Records.Product}}}
```

### Bind Custom Object datasource

C#

```
var datasource = new SalesData
{
    Records = new List<SalesRecord>()
};

var record1 = new SalesRecord
{
    Area = "NorthChina",
    Salesman = "Hellen",
    Product = "Apple",
    ProductType = "Fruit",
    Sales = 120
};
datasource.Records.Add(record1);

var record2 = new SalesRecord
{
    Area = "NorthChina",
    Salesman = "Hellen",
    Product = "Banana",
    ProductType = "Fruit",
    Sales = 143
};
datasource.Records.Add(record2);

...//Init data

//Add data source
workbook.AddDataSource("ds", datasource);
```

### Variable

A user-defined variable in code

### Template Syntax

[Alias of data source]

For example:

{{cName}}

{{count}}

{{owner}}

## Bind Variable datasource

C#

```
var className = "Class 3";  
var count = 500;  
  
//Add data source  
workbook.AddDataSource("cName", datasource);  
workbook.AddDataSource("count", count);  
workbook.AddDataSource("owner", "Hunter Liu");
```

## Array or List

A user-defined array or list in code

### Template syntax

1. Array or List of base type variable(string, int , double, etc.)

[Alias of data source]

2. Array or List of custom object

[Alias of data source].[Field name]

or

[Alias of data source].[Property name]

For example:

{{p.Name}}

{{p.Age}}

{{countries}}

{{numbers}}

## Bind Array or List datasource

C#

```
int[] numbers = new int[] { 10, 12, 8, 15};
List<string> countries = new List<string>() { "USA", "Japan", "UK", "China" };

List<Person> peoples = new List<Person>();

Person p1 = new Person();
p1.Name = "Helen";
p1.Age = 12;
peoples.Add(p1);

Person p2 = new Person();
p2.Name = "Jack";
p2.Age = 23;
peoples.Add(p2);

Person p3 = new Person();
p3.Name = "Fancy";
p3.Age = 25;
peoples.Add(p3);

workbook.AddDataSource("p", peoples);
workbook.AddDataSource("countries", countries);
workbook.AddDataSource("numbers", numbers);
```

## Create Excel Report using Template

This walkthrough considers the use case to create a Marketing Report of a company which is launching a new series of smartphones. Hence, an Excel report for the planned marketing activities needs to be created. The report details out the planned events for the launch, its budget and expenses. The datasource used for binding the data, in this case, is DataTable. The template layout is created in different Excel tabs to generate multiple reports.

The below steps describe how to create an Excel report using template. You can also download the **Excel template layout** [here](#).

1. Create template layouts in different Excel worksheets of a workbook. Define the template layout of Marketing report using different types of fields:
  - o **Static Fields:** Define the static fields in template layout, that is, the fields whose values will remain constant in the final report. For example, the header fields or template header like Marketing Report, SmartPhone, Event etc.
  - o **Bound Fields:** Specify the datasource bound fields in mustache braces {{ }}. For DataTable datasource, define the bound fields as {{ds.FieldName}} where ds is the alias of the datasource, specified in code using AddDataSource method.
  - o **Expression Fields:** Specify the functions in fields whose value will be calculated using formulas.
  - o **Sheet Name:** Create multiple template layouts in different Excel tabs, namely, Marketing Report, Smartphone expenses and Launch events. Specify a data bound FieldName for last sheet, {{ds.Country}}, which will generate multiple reports based on the values of 'Country' field in the data source.

### Template Layout: Marketing Report

The below layout uses the Group property (G=Merge), which will group the smartphones against the corresponding records by displaying it once per group. The merge value merges the cells of each group.

	A	B	C	D	E	F	G
1	<b>Marketing Report</b>						
2							
3	SmartPhone	Event	Budget	Expense	City	Country	Saving
4							
5	{{ds.SmartPhone(G=Merge)}}	{{ds.Event}}	{{ds.Budget}}	{{ds.Expense}}	{{ds.City}}	{{ds.Country}}	{{=ds.Budget-ds.Expense}}
6							
7			Total Expenses:	{{=SUM(ds.Expense)}}			
8							
9							
10							
11							

Marketing Report | Smartphone Expenses | Launch Events | {{ds.Country}}

### Template Layout: SmartPhone Expenses

The below layout uses two template properties, Cell expansion (E=H) and Cell context (C=A3)

- The cell expansion property will expand the smartphone field horizontally.
- The cell context property will make sure that the expense field expands horizontally depending upon the smartphone field.

	A	B	C	D	E	F	G	H
1	<b>SmartPhone Expenses</b>							
2								
3	{{ds.SmartPhone (E=H)}}							
4								
5	Expense							
6	\$ {{ds.Expense(C=A3)}}							
7								
8								
9								

Marketing Report | Smartphone Expenses | Launch Events | {{ds.Country}}

### Template Layout: Launch Events

The below layout uses four template properties, Range (R=A3:B5), Sort (S=None), Cell expansion (E=H) and Page break (PageBreak=True)

- The Range property acts as the fallback context for the fields in specified range, which means, that the fields which have no default or explicit context will use this current field as their context.
- The Sort property will not sort the events based on its 'none' value
- The event field will expand horizontally based on the cell expansion property
- The Page Break property will add a vertical and a horizontal page break

	A	B	C	D	E	F	G	H	I
1	Launch Events								
2									
3	SmartPhone	{{ds.SmartPhone (R=A3:B5)}}							
4	Event	{{ds.Event (S=None,E=H,PageBreak=True)}}							
5									
6									
7									
8									
9									

Marketing Report   Smartphone Expenses   **Launch Events**   {{ds.Country}}

#### Template Layout: {{ds.Country}}

	A	B	C	D	E	F	G	H	I
1	Marketing Report								
2									
3	SmartPhone	Event	Budget	Expense	City				
4									
5	{{ds.SmartPhone}}	{{ds.Event}}	{{ds.Budget}}	{{ds.Expense}}	{{ds.City}}				
6									
7			Total Expenses:	{{=SUM(ds.Expense)}}					
8									
9									

Marketing Report   Smartphone Expenses   Launch Events   **{{ds.Country}}**

- Load the template in GcExcel.

C#

```
Console.WriteLine("Generating Marketing Report using GcExcel Templates");
// Initialize Workbook
var workbook = new Workbook();
// Load BudgetPlan_DataTable.xlsx Template in workbook from Resource
var templateFile = "../../../Resources/BudgetPlan_DataTable.xlsx";
workbook.Open(templateFile);
```

- Configure DataSource and add DataColumnns and data to the DataTable.

C#

```
// We can have mutiple types of DataSources like Custom Object/ DataSet/ DataTable/
// Json/ Variable.
// Here dataSource is a DataTable
var dataSource = new DataTable();

// Adding DataColumnns in DataTable according to the Template fields
dataSource.Columns.Add(new DataColumn("SmartPhone", typeof(string)));
dataSource.Columns.Add(new DataColumn("Event", typeof(string)));
dataSource.Columns.Add(new DataColumn("Budget", typeof(Int32)));
dataSource.Columns.Add(new DataColumn("Expense", typeof(Int32)));
dataSource.Columns.Add(new DataColumn("City", typeof(string)));
dataSource.Columns.Add(new DataColumn("Country", typeof(string)));
```

```
// Adding Data in DataTable
dataSource.Rows.Add("Apple iPhone 11", "Phone Launch", 1000, 950, "Seattle", "USA");
dataSource.Rows.Add("Apple iPhone 11", "CEO Meet", 2000, 1800, "New York", "USA");
dataSource.Rows.Add("Samsung Galaxy S10", "CEO Meet", 1600, 1550, "Paris", "France");
dataSource.Rows.Add("Apple iPhone XR", "Phone Launch", 1800, 1650, "Cape Town", "South Africa");
dataSource.Rows.Add("Samsung Galaxy S9", "Phone Launch", 1500, 1300, "Paris", "France");
dataSource.Rows.Add("Apple iPhone XR", "CEO Meet", 1600, 1500, "New Jersey", "USA");
dataSource.Rows.Add("Samsung Galaxy S9", "CEO Meet", 1200, 1150, "Seattle", "USA");
dataSource.Rows.Add("Samsung Galaxy S10", "Phone Launch", 1100, 1070, "Durban", "South Africa");
```

4. Add DataSource in GcExcel, using the AddDataSource method.

```
C#

// Add DataSource
// Here "ds" is the name of dataSource which is used in templates to define fields like
//{{ds.SmartPhone}}
workbook.AddDataSource("ds", dataSource);
```

5. Execute the template using ProcessTemplate method.

```
C#

// Invoke to process the template
workbook.ProcessTemplate();
```

6. Save the final report.

```
C#

// Save to an excel file
Console.WriteLine("BudgetPlan_DataTable.xlsx Template is now bound to DataTable and
generated MarketingReport_DataTable.xlsx file");
workbook.Save("MarketingReport_DataTable.xlsx");
```

The output of the Marketing Report is shown as below.

**Excel Report: Marketing Report**



	A	B	C	D	E	F	G	H
1	<b>Marketing Report</b>							
2								
3	SmartPhone	Event	Budget	Expense	City	Country	Saving	
4								
5	Apple iPhone 11	CEO Meet	2000	1800	New York	USA	200	
6		Phone Launch	1000	950	Seattle	USA	50	
7	Apple iPhone XR	CEO Meet	1600	1500	New Jersey	USA	100	
8		Phone Launch	1800	1650	Cape Town	South Africa	150	
9	Samsung Galaxy S10	CEO Meet	1600	1550	Paris	France	50	
10		Phone Launch	1100	1070	Durban	South Africa	30	
11	Samsung Galaxy S9	CEO Meet	1200	1150	Seattle	USA	50	
12		Phone Launch	1500	1300	Paris	France	200	
13								
14			Total Expenses:	10970				
15								
16								

Marketing Report | Smartphone Expenses | Launch Events | France | South Africa | USA | ... (+) |

Excel Report: Smartphone Expenses

	A	B	C	D	E	F	G
2							
3	Apple iPhone 11	Apple iPhone XR	Samsung Galaxy S10	Samsung Galaxy S9			
4							
5	Expense						
6	\$ 950	\$ 1500	\$ 1070	\$ 1150			
7	\$ 1800	\$ 1650	\$ 1550	\$ 1300			
8							
9							

Marketing Report | Smartphone Expenses | Launch Events | France | South Africa | USA | ... (+) |

Excel Report: Launch Events

	A	B	C	D	E	F	G	H
1	<b>Launch Events</b>							
2								
3	SmartPhone	Apple iPhone 11						
4	Event	Phone Launch	CEO Meet					
5								
6	SmartPhone	Apple iPhone XR						
7	Event	Phone Launch	CEO Meet					
8								
9	SmartPhone	Samsung Galaxy S10						
10	Event	CEO Meet	Phone Launch					
11								
12	SmartPhone	Samsung Galaxy S9						
13	Event	Phone Launch	CEO Meet					
14								
15								

Marketing Report | Smartphone Expenses | Launch Events | France | South Africa | USA | ... (+) |

Excel Report: Countries (Multiple reports are created)


	A	B	C	D	E	F	G
1	Marketing Report						
2							
3	SmartPhone	Event	Budget	Expense	City		
4							
5		CEO Meet	2000	1800	New York		
6	Apple iPhone 11	Phone Launch	1000	950	Seattle		
7	Apple iPhone XR	CEO Meet	1600	1500	New Jersey		
8	Samsung Galaxy S9	CEO Meet	1200	1150	Seattle		
9							
10			Total Expenses:	5400			
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							
36							
37							
38							
39							
40							
41							
42							
43							
44							
45							
46							
47							
48							
49							
50							
51							
52							
53							
54							
55							
56							
57							
58							
59							
60							
61							
62							
63							
64							
65							
66							
67							
68							
69							
70							
71							
72							
73							
74							
75							
76							
77							
78							
79							
80							
81							
82							
83							
84							
85							
86							
87							
88							
89							
90							
91							
92							
93							
94							
95							
96							
97							
98							
99							
100							
101							
102							
103							
104							
105							
106							
107							
108							
109							
110							
111							
112							
113							
114							
115							
116							
117							
118							
119							
120							
121							
122							
123							
124							
125							
126							
127							
128							
129							
130							
131							
132							
133							
134							
135							
136							
137							
138							
139							
140							
141							
142							
143							
144							
145							
146							
147							
148							
149							
150							
151							
152							
153							
154							
155							
156							
157							
158							
159							
160							
161							
162							
163							
164							
165							
166							
167							
168							
169							
170							
171							
172							
173							
174							
175							
176							
177							
178							
179							
180							
181							
182							
183							
184							
185							
186							
187							
188							
189							
190							
191							
192							
193							
194							
195							
196							
197							
198							
199							
200							
201							
202							
203							
204							
205							
206							
207							
208							
209							
210							
211							
212							
213							
214							
215							
216							
217							
218							
219							
220							
221							
222							
223							
224							
225							
226							
227							
228							
229							
230							
231							
232							
233							
234							
235							
236							
237							
238							
239							
240							
241							
242							
243							
244							
245							
246							
247							
248							
249							
250							
251							
252							
253							
254							
255							
256							
257							
258							
259							
260							
261							
262							
263							
264							
265							
266							
267							
268							
269							
270							
271							
272							
273							
274							
275							
276							
277							
278							
279							
280							
281							
282							
283							
284							
285							
286							
287							
288							
289							
290							
291							
292							

## 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 PDF](#)
- [Export to a HTML 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](#)
- [Convert to Image](#)

 **Note:** While exporting a worksheet to PDF, HTML or image file formats, you can set the **HorizontalAlignment** enumeration to **CenterContinuous** to center align the text across multiple cells. However, the horizontal alignment should be set before merging the cells. Otherwise, the cells may be drawn incorrectly in exported file formats.

## 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** 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	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	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	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	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	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	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 you want to import only the conditional formatting rule applied to the worksheet while opening the Excel file.
DataValidation	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	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	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** property of the **XlsxOpenOptions** class allows you to set a boolean value (True or False) which specifies whether or not the formulas will be recalculated when the file is being imported.

The **IgnoreFormulas** property of **XlsxSaveOptions** class, when set to true, allows you to export formula cells in GcExcel as value cells in Excel. When *workbook.Save* is called, the resultant values for formula cells are calculated and are saved in Excel.

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);

// Exporting .xlsx document by ignoring cell formulas
workbook.ActiveSheet.Range["A4"].Formula = "=Sum(A1+A2+A3)";
XlsxSaveOptions options2 = new XlsxSaveOptions();
options2.IgnoreFormulas = true;
workbook.Save(@"ignoreformulas.xlsx", options2);

// 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 PDF

GcExcel .NET allows you to export workbook to a PDF file. You can also apply styles, customize fonts, add security options, configure document properties and adjust row height or column width while performing the export operation.

To save all visible spreadsheets in a workbook to a Portable Document File (PDF), use the **Save()** method of the **IWorkbook** interface. Each worksheet in a workbook is saved to a new page in the PDF file. You can also export only the current sheet (active sheet) to PDF format using the **Save()** method of the **IWorksheet** interface.

The handling of images in the case of PDF export is also very efficient. If a picture is used multiple times in a spreadsheet, GcExcel maintains a single copy of the picture which reduces the size of exported PDF file.

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);
```

GcExcel also supports setting JavaScript in PDF documents by using **OpenActionScript** property of **PdfSaveOptions** class. The JavaScript is executed when the saved PDF document is opened.


Refer to the following example code to set JavaScript in an Excel template which is processed to create a PDF form.

```
C#  
  
Workbook workbook = new Workbook();  
workbook.Open("SampleTemplate.xlsx");  
  
workbook.ProcessTemplate();  
PdfSaveOptions options = new PdfSaveOptions();  
  
//Set JavaScript  
options.OpenActionScript = "var fld1 = this.getField(\"num\");" +  
"fld1.value = fld1.value;" +  
"this.dirty = false;";  
  
workbook.Save("SampleTemplate.pdf", options);
```

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](#)
- [Support Security Options](#)
- [Support Document Properties](#)
- [Adjust Column Width and Row Height](#)
- [Export Charts](#)
- [Export Slicers](#)
- [Export Barcodes](#)
- [Export Signature Lines](#)
- [Support Sheet Background Image](#)
- [Support Background Color Transparency](#)
- [Control Image Quality](#)
- [Track Export Progress](#)

While printing the PDF document, you can also configure to choose the paper source automatically based on PDF page size. For more information, please refer to [Configure Paper Source](#).

 **Note:** The Export to PDF feature doesn't support exporting picture settings (such as LineFormat, FillFormat, Brightness, Contrast, Watermark Color Type and black and white pictures in emf format) to PDF files.

## 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** property of the **Workbook** 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 necessary that the user sets the font folder path and copies the used font files to it from the folder "C:\Windows\Fonts".

You can use the **GetUsedFonts()** method of the **IWorkbook** 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.

## 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** property of the **IPivotTable** 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	Description
<b>IPivotTable.ShowTableStyleColumnHeaders</b>	This property can be used to get or set whether the column headers should be displayed in the Pivot table.
<b>IPivotTable.ShowTableStyleRowHeaders</b>	This property can be used to get or set whether the row headers should be displayed in the Pivot table.
<b>IPivotTable.ShowTableStyleColumnStripes</b>	This property can be used to get or set whether the banded columns in which even columns are formatted differently from odd columns.
<b>IPivotTable.ShowTableStyleRowStripes</b>	This property can be used to get or set whether the banded rows in which even row are formatted differently from odd rows.
<b>IPivotTable.ShowTableStyleLastColumn</b>	This property can be used to get or set whether to display the grand total columns style.
<b>ITableStyle.ShowAsAvailablePivotStyle</b>	This property can be used to get or set whether the specified style is shown as available in the pivot styles gallery.
<b>IPivotField.NumberFormat</b>	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** property of the **IShape** 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** property of the **IRange** interface sets the orientation of the text.
- `IRange.Font.Name` - Sets the specific font name using the **Font** property of the **IRange** 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** 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** property of the **PdfSaveOptions** class.
- The **CanShrinkToFitWrappedText** property of the **IShrinkToFitSettings** 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** 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** 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** 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** class
- Step 2 - Get the default pagination information of the workbook using the **Paginate()** method.
- Step 3 - Adjust pagination using different properties and methods of the **PrintManager** class.
- Step 4 - Save the PDF file by using the **SavePDF()** 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](#)
- [Export Worksheet to 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** class and then use the **Draw()** 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()** method of the PrintManager class after configuring all the pagination settings. Finally, call the **UpdatePageNumberAndPageSettings()** 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()** 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** class, get the default pagination settings of the workbook using the **Paginate()** method, use the **Draw()** method of the PrintManager class and finally save the PDF file using the **SavePDF()** 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** class and use the **Paginate()** method to ensure the desired rows are displayed together. When you are done, simply save your PDF file using the **SavePDF()** 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** class and use the **Paginate** method to get the default pagination of the workbook. Now, you can use the **HasPrintContent** method to check whether the pages have content or not. Finally, call the **UpdatePageNumberAndPageSettings** 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()** 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** property, the **TitleRowEnd** property, and other properties of the **RepeatSetting** class. When you are done, simply create an instance of the **PrintManager** class, get the default pagination information using the **Paginate()** method and finally save your PDF file using the **SavePDF()** 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];

IEnumerable<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
IEnumerable<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()** method of the **PrintManager** class. Then, you can use the **PageContent** property of the **PageInfo** class and the **TitleRowStart** property of the **PageContentInfo** class in order to modify the header index of the last page. When you are done, simply save your file using the **SavePDF()** 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** 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** 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** 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** class and get the default pagination using the **Paginate()** method. Next, you need to specify the pages that you want to export or print. Finally, call the **UpdatePageNumberAndPageSettings()** 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()** 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()**



method of the **PrintManager** 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);
```

## Export Worksheet to PDF

GcExcel .NET provides the option to paginate a worksheet automatically, according to page boundaries, while exporting to PDF file.

The **GetPaginationInfo** method of **PrintManager** class gets an array of the page boundaries for horizontal and vertical paging. The method needs to be called separately for horizontal and vertical pagination. The retrieved pagination information is based on the page setup settings. The print area of the worksheet can also be defined. In case it is not defined, the default area is considered from cell A1 till the last cell where any cell data is present.

In addition to the page setup settings, you can also define ranges which need to be kept together and repeat settings of a range by using the overload of **GetPaginationInfo** method.

### Using Code

Refer to the following example code to paginate a worksheet while exporting to PDF file based on the page setup settings.

C#

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

// The row and column headings are printed
worksheet.PageSetup.PrintHeadings = true;
// The range "B6:N80" will be printed
worksheet.PageSetup.PrintArea = "B6:N80";

// Set data
worksheet.Range["B6:S8"].Value = "1";
// Add a table
worksheet.Tables.Add(worksheet.Range["B6:N20"], true);

PrintManager printManager = new PrintManager();

// The columnIndex is [9, 13], this means that the horizontal direction is split after
the column 10th and 14th
IList<int> columnIndex = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Horizontal);
Console.WriteLine("In horizontal direction, page is split after column : " +
columnIndex[0].ToString() + " & " + columnIndex[1].ToString());

// The rowIndex is [50, 79], this means that the vertical direction is split after the
```

```
row 51th and 80th
IList<int> rowIndexes = printManager.GetPaginationInfo(worksheet,
    PaginationOrientation.Vertical);
Console.WriteLine("In vertical direction, page is split after row : " +
    rowIndexes[0].ToString() + " & " + rowIndexes[1].ToString());

worksheet.Save(@"GetPagination.pdf", SaveFileFormat.Pdf);
```

Refer to the following example code to paginate a worksheet while exporting to PDF file based on the page setup settings, range to be kept together and repeat settings.

C#

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

// The row and column headings are printed
worksheet.PageSetup.PrintHeadings = true;
// The range "B6:N80" will be printed
worksheet.PageSetup.PrintArea = "B6:N80"; ;

// Set data
worksheet.Range["B60:N80"].Value = 1;
// Add a table
worksheet.Tables.Add(worksheet.Range["B6:N20"], true);

// The row 6th will be printed at the top of each page
IList<RepeatSetting> repeatSettings = new List<RepeatSetting>();
RepeatSetting repeatSetting = new RepeatSetting();
repeatSetting.TitleRowStart = 5;
repeatSetting.TitleRowEnd = 5;
repeatSetting.Range = worksheet.Range["B6:N80"];
repeatSettings.Add(repeatSetting);

// The rows from 25th to 60th should be paged to one page
IList<IRange> keepTogetherRanges = new List<IRange>();
keepTogetherRanges.Add(worksheet.Range["$25:$60"]);

PrintManager printManager = new PrintManager();

// The columnIndex is [9, 13], this means that the horizontal direction is split after
the column 10th and 14th.
IList<int> columnIndex = printManager.GetPaginationInfo(worksheet,
    PaginationOrientation.Horizontal, keepTogetherRanges, repeatSettings);
// The rowIndex is [23, 66, 79], this means that the vertical direction is split after
the row 24th, 67th and 80th.
IList<int> rowIndex = printManager.GetPaginationInfo(worksheet,
    PaginationOrientation.Vertical, keepTogetherRanges, repeatSettings);
```

```
ICollection<PageInfo> pages = printManager.Paginate(worksheet, keepTogetherRanges, repeatSettings);  
printManager.SavePDF(@"GetPaginationRangesRepeatSettings.pdf", pages);
```

## Working With Page Setup

GcExcel .NET allows users to paginate each worksheet using the properties of the **IPageSetup** 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 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** interface and the **IWorksheet** interface:

Settings	Description
<b>IPageSetup.PaperSize</b>	This property can be used to determine the size of each page. For more information on implementation of this property, refer to <a href="#">Configure Paper Settings</a> .
<b>IPageSetup.Orientation</b>	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 <a href="#">Configure Page Settings</a> .
<b>IPageSetup.PrintTitleRows</b>	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 <a href="#">Configure Rows to Repeat at Top and Bottom</a> .
<b>IPageSetup.PrintTitleColumns</b>	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 <a href="#">Configure Columns to Repeat at Left and Right</a> .
<b>IPageSetup.PrintArea</b>	This property can be used to specify the print area in a worksheet. If the 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 <a href="#">Configure Print Area</a> .
<b>IPageSetup.Zoom</b>	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 <a href="#">Configure Paper Settings</a> .
<b>IPageSetup.FitToPagesWide</b>	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 <a href="#">Configure Paper Settings</a> .
<b>IPageSetup.FitToPagesTall</b>	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 <a href="#">Configure Paper Settings</a> .
<b>IPageSetup.IsPercentScale</b>	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 <a href="#">Configure Paper Settings</a> .
<b>IWorksheet.HPageBreaks</b>	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 <a href="#">Configure Page Breaks</a> .
<b>IWorksheet.VPageBreaks</b>	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 <a href="#">Configure Page Breaks</a> .

For more information on setting pagination, refer to [Print Settings](#).

 **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.

## Support Security Options

Sometimes, users need to secure their digital documents with user/owner passwords, print permission, content permission, annotation permission etc. PDF documents have always been the preferred format for sharing digital files among professionals. The GcExcel library supports Security Options while saving Excel spreadsheets to PDF files. It helps in securing a PDF Document by restricting the PDF's access to unauthorized users as per the options specified.

With GcExcel's **PdfSecurityOptions** class, you can restrict access to your PDF document, while converting Excel spreadsheet to PDF document. You can choose through the following security properties in the **PdfSecurityOptions** class:

Properties	Description
<b>UserPassword</b>	Gets or sets the user password of the PDF document.
<b>OwnerPassword</b>	Gets or sets the owner password of the PDF document. This password is required to change the permissions for the PDF document.
<b>PrintPermission</b>	Gets or sets the permission to print the PDF document. The default value is true for this property.
<b>FullQualityPrintPermission</b>	Gets or sets the permission to print in high quality. The default value is true for this property, and it only works when <b>PrintPermission</b> property is set to true.
<b>ExtractContentPermission</b>	Gets or sets the permission to copy or extract content. The default value is true for this property.
<b>ModifyDocumentPermission</b>	Gets or sets the permission to modify the PDF

	document. The default value is true for this property.
<b>AssembleDocumentPermission</b>	Gets or sets the permission to insert, rotate or delete pages, and create bookmarks/thumbnail images. The default value is true for this property. If you want to prevent a user from inserting, rotating or deleting pages, you need to set <b>ModifyDocumentPermission</b> property to false as well.
<b>ModifyAnnotationsPermission</b>	Gets or sets the permission to modify text annotations and fill the form fields. The default value is true for this property.
<b>FillFormsPermission</b>	Gets or sets the permission to fill the form fields even if the <b>ModifyAnnotationsPermission</b> property returns false. The default value for this property is true. Note that if you want to prevent a user from filling interactive form fields, you need to set the <b>ModifyAnnotationsPermission</b> property to false.

## Using Code

Refer to the following example to add security options while exporting Excel spreadsheets to PDF documents.

C#
<pre> public void SavePDFPdfSecurityOptions() {     // Initialize workbook     Workbook workbook = new Workbook();     // Fetch default worksheet     IWorksheet worksheet = workbook.Worksheets[0];     // Data     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; </pre>

```
//The security settings of pdf when converting excel to pdf
PdfSecurityOptions securityOptions = new PdfSecurityOptions();
//Sets the user password
securityOptions.UserPassword = "user";
//Sets the owner password
securityOptions.OwnerPassword = "owner";
//Printing the pdf document is not allowed
securityOptions.PrintPermission = false;
//Filling the form fields of the pdf document is not allowed
securityOptions.FillFormsPermission = false;

PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
//Sets the security settings of the pdf
pdfSaveOptions.SecurityOptions = securityOptions;

// Saving workbook to PDF
workbook.Save(@"4-SavePDFPdfSecurityOptions.pdf", pdfSaveOptions);
}
```



**Note:** GcExcel uses RC4 encryption with key from 40 to 128 bit length and allows to define additional permission flags.

## Support Document Properties

GcExcel provides support for document properties while saving Excel spreadsheets to PDF documents. The document properties contain the basic information about a document, such as title, author, creation date, subject, creator, version etc. You can store such useful information in the exported PDF document.

The **DocumentProperties** class contains the properties such as **PdfVersion**, **EmbedStandardWindowsFonts**, **Title**, **Author**, **Subject**, **Keywords**, **Creator**, **Producer**, **CreationDate** and **ModifyDate**.

### Using Code

Refer to the following example code to add document properties in the exported PDF document.

C#


```
DocumentProperties documentProperties = new DocumentProperties();
//Sets the name of the person that created the document
documentProperties.Author = "Will Smith";
//Sets the title of the document
documentProperties.Title = "Document properties Info Sample";
//Sets the subject of the document
documentProperties.Subject = "PDF created from GcExcel";
//Do not embed a font
documentProperties.EmbedStandardWindowsFonts = false;

PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
```

```
//Sets the DocumentProperties of the pdf
pdfSaveOptions.DocumentProperties = documentProperties;
```

## Adjust Column Width and Row Height

GcExcel provides **BestFitColumns** and **BestFitRows** properties in the **IPageSetup** interface to properly display long or large-size font texts in cells while printing PDF documents. These properties support SSJSON I/O and are consistent with SpreadJS. The **BestFitColumns** property when set to True, resizes the column width to fit the text with the longest width for printing. Similarly, the **BestFitRows** property when set to True, resizes the row height to fit the text with the tallest height for printing.

 **Note:** GcExcel preserves useMax property during JSON I/O. In case a file contains large amount of data, these properties may not work as expected.

### Using Code

Refer to the following example code to adjust column width or row height.

```
C#
// Set bestFitColumns/bestFitRows as true
worksheet.PageSetup.BestFitColumns = true;
worksheet.PageSetup.BestFitRows = true;
```

### Export Charts

In GcExcel, you can export charts to PDF documents. This helps in the easy generation of PDF reports from spreadsheets, like finance, sales, marketing, health care etc. Following chart types can be exported to PDF documents:

- Column Chart
- Line Chart
- Pie Chart
- Bar Chart
- Area Chart
- XY (Scatter) Chart
- Stock Chart
- Radar Chart
- Combo Chart

#### Using Code

Refer to the following example code to export charts in Excel files to PDF document.

```
C#
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnStacked, 50, 120, 350, 250);
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);

// Saving workbook to pdf
workbook.Save(@"ExportChartsToPDF.pdf", SaveFileFormat.Pdf);
```

#### Limitations

The following chart types are not supported and the corresponding area would appear empty when exported to PDF:

- Column Chart
  - 3D Clustered
  - 3D Stacked



- 3D 100% Stacked
  - 3D Column
- Line Chart
  - 3D Line
- Pie Chart
  - 3D Pie
  - Pie of Pie
  - Bar of Pie
- Bar Chart
  - 3D Clustered
  - 3D Stacked
  - 3D 100% Stacked
- Area Chart
  - 3D Area
  - 3D Stacked Area
  - 3D 100% Stacked Area
- Scatter Chart
  - 3D Bubble
- Map
- Surface
- Histogram
- Pareto
- Box and Whisker
- Waterfall
- Funnel

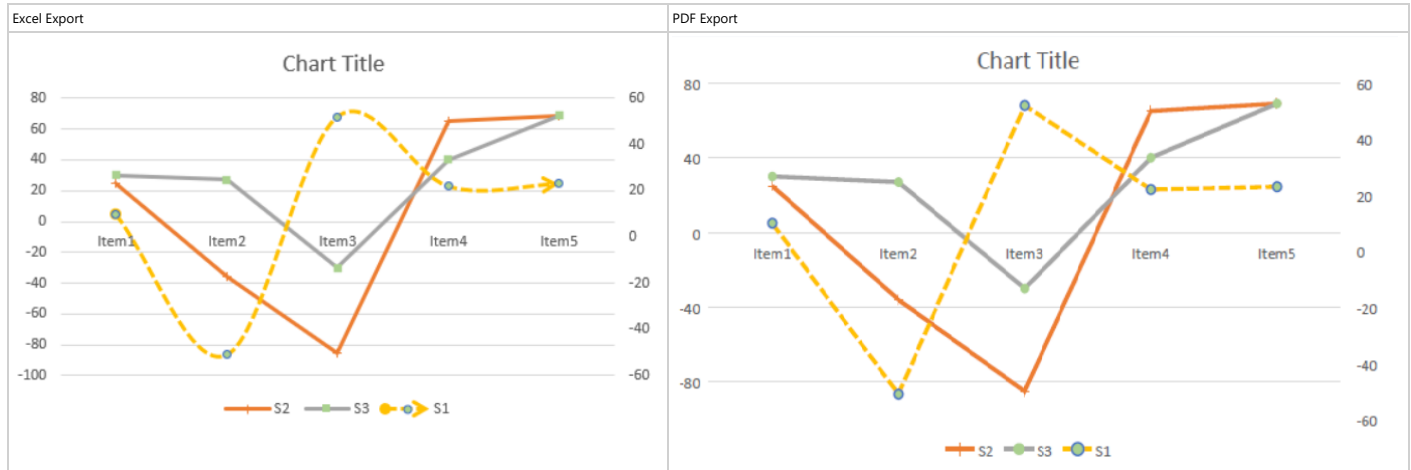
#### Supported features

The below table shows the supported features in different chart types when exported to PDF.

**Common Features** (Supported by all Chart Types)

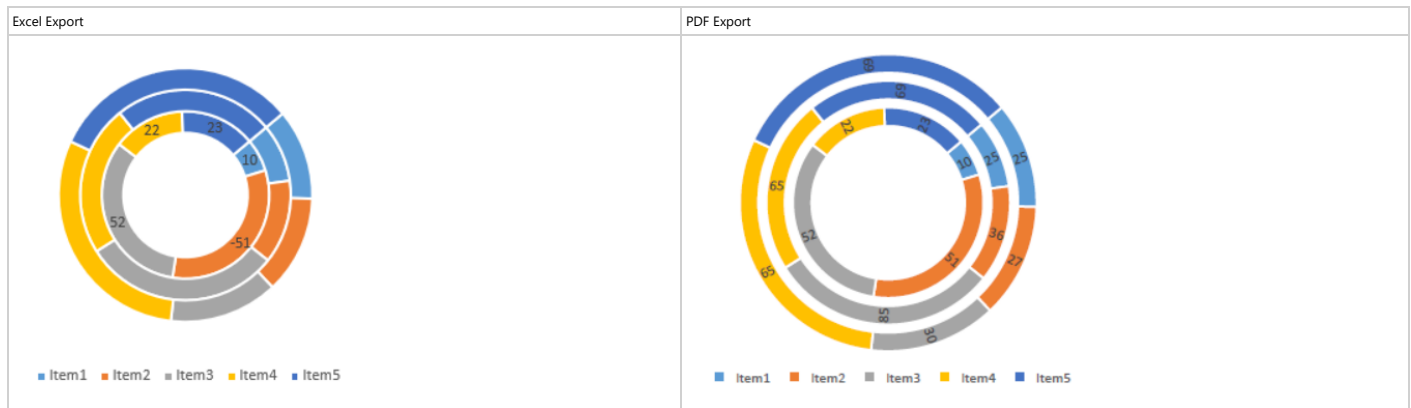
Features	Settings	Supported
Chart Title	font size	Yes
	font color	Yes
	border	Yes
	fill	Yes
	overlap with plot area	No
	custom angle	No
	text direction	No
Plot Area	border	Yes
	fill	Yes
	free layout(resize)	Yes
Axes	show/hide	Yes
	fill	Yes
	title	Yes
	title free layout	No
	border	Yes
	angle(text rotate)	No
	max/min bounds	Yes (when min/max is auto, the rendered result might be different between MSeExcel and GcExcel)
	major/minor unit	
	horizontal axis cross position	No
	display units	Yes
	logarithmic scale	Yes
	values in reverse order	Yes
	tick marks	Yes
	label position	Yes
	number format	Yes
Data Label	fill	Yes
	border	Yes
	font	Yes
	position	Yes
	number format	Yes
	contains(series name/category name/values)	Yes
Data Table		No
Error Bars	direction	Yes
	end style	Yes
	error amount	Yes
Gridlines	major/minor	Yes
	value axis	Yes
	category axis	Yes
	color	Yes
Legend	fill	Yes
	border	Yes

	location(top/bottom/left/right)	Yes
	free layout(resize)	Yes
Trend Line		Yes
Series Option	primary axis	Yes
	secondary axis	Yes
	series overlap	Yes
	gap width	Yes

**Line Chart**

The below table shows the supported features in line chart type when exported to PDF.

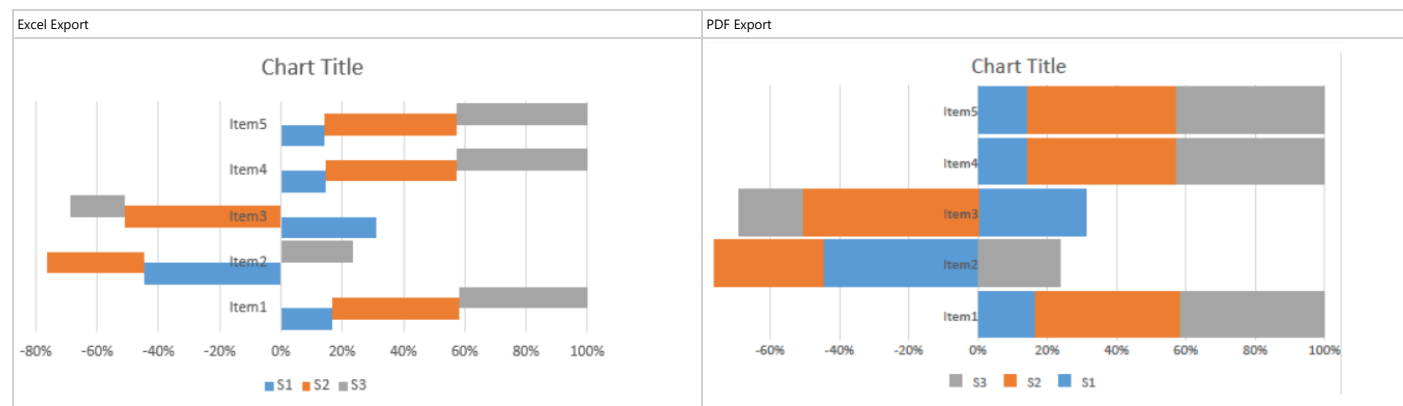
Features	Settings	Supported
Line	solid color	Yes
	gradient color	No
	weight	Yes
	cap type	No
	join type	No
	dash type	Yes
	begin arrow	No
	end arrow	No
	smooth line	No
Marker	size	Yes
	fill	Yes
	border	Yes
Drop Lines	-	No
High-Low Lines	-	No
Up-Down Bars	-	No

**Pie Chart**

The below table shows the supported features in pie chart type when exported to PDF.

Features	Settings	Supported
Pie Settings	angle of first slice	Yes
	explosion	No
Dough Chart	doughnut hole size	Yes

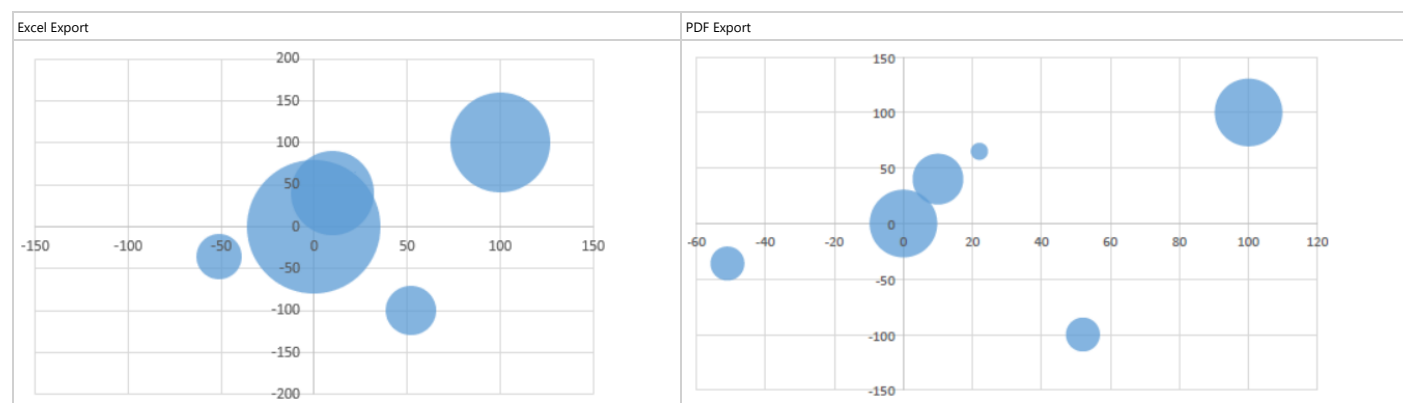
#### Bar Chart



The below table shows the supported features in bar chart type when exported to PDF.

Features	Settings	Supported
Series Option	overlap	No
	gap width	Yes

#### Scatter Chart

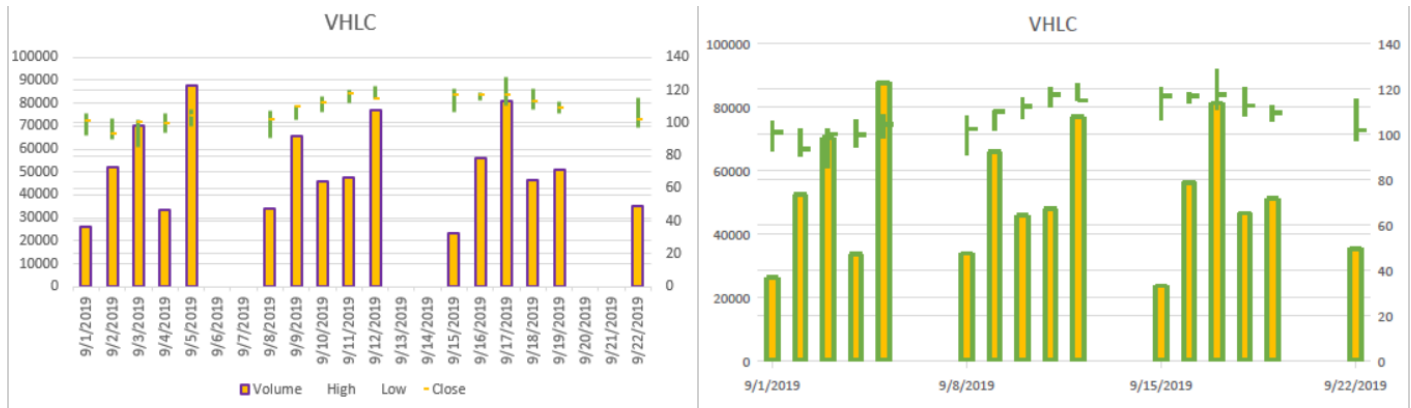


The below table shows the supported features in scatter chart type when exported to PDF.

Features	Settings	Supported
Chart Type	scatter	Yes
	scatter with smooth lines and markers	Yes
	scatter with smooth lines	Yes
	scatter with straight lines and markers	Yes
	scatter with straight lines	Yes
	bubble	Yes
	3D-bubble	No
Bubble Settings	size represents	No
	scale bubble size	No

#### Stock Chart

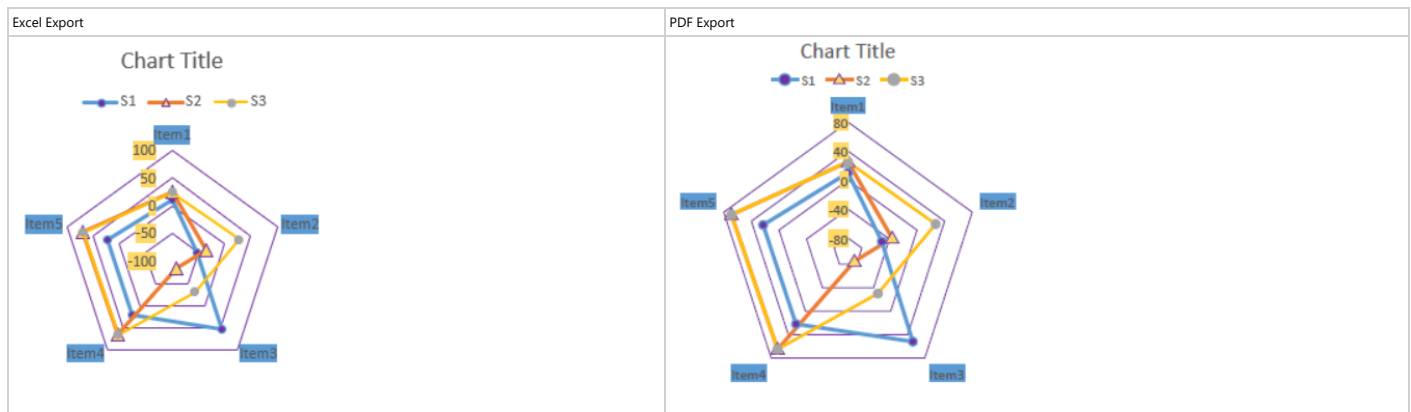
Excel Export	PDF Export
--------------	------------



The below table shows the supported features in stock chart type when exported to PDF.

Features	Settings	Supported
Common Features	line color	Yes
	stock	Yes
	line dash type	No
	line cap	No
	line join	No
	series marker	No
	series line	No
Open-High-Low-Close	gap width	No
	down-bar fill	Yes
	down-bar border	No
	up-bar	No
Volume-High-Low-Close / Volume-Open-High-Low-Close	volumn fill	Yes
	volume border	No

#### Radar Chart

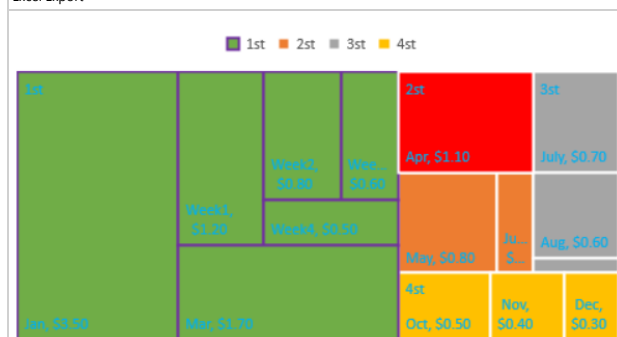


The below table shows the supported features in radar chart type when exported to PDF.

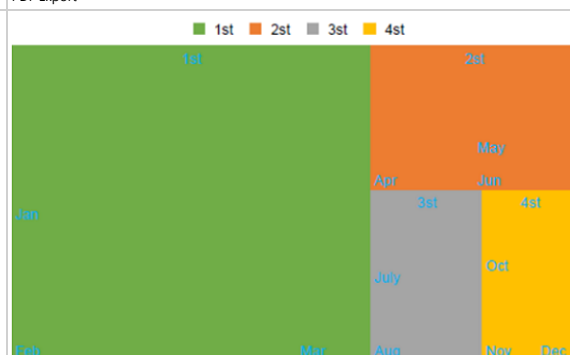
Features	Settings	Supported
Chart Type	radar	Yes
	radar with markers	Yes
	filled radar	Yes
Series Line	width	Yes
	color	Yes
Marker	type	Yes
	size	Yes
	fill	Yes
	border	Yes

#### TreeMap Chart

Excel Export



PDF Export

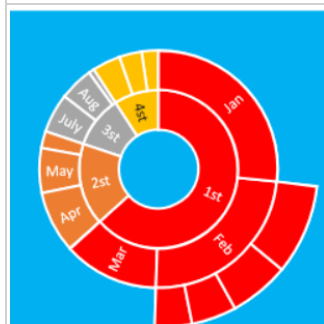


The below table shows the supported features in Treemap chart type when exported to PDF.

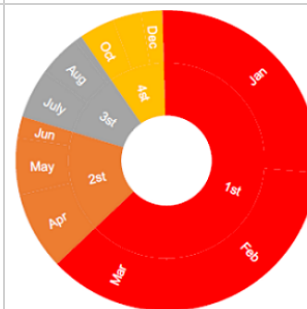
Features	Settings	Supported
Series Option	banner	No
	overlapping	No
Label Option	contains(series name/category name/value)	No
	number format	No
	text font	Yes
	text color	Yes
Point Formatting	fill	Yes
	line	Yes

Sunburst Chart

Excel Export



PDF Export



The below table shows the supported features in Sunburst chart type when exported to PDF.

Features	Settings	Supported
Plot Area	fill	No
Label Option	contains(series name/category name/value)	No
	number format	No
	text font	Yes
	text color	Yes
Point Formatting	fill	Yes
	line	Yes

## Export Slicers

Slicers are visual filters that are used to filter data in Excel spreadsheets. You can filter the data by clicking on desired type of data in slicer.

GcExcel supports the export of Excel spreadsheet containing a slicer to PDF document. So, if an Excel spreadsheet containing a slicer is exported to PDF, the resulting PDF will contain the applied slicer.

## Using Code

Refer to the following example code to export slicers to PDF document.

C#

```
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A:F"].ColumnWidth = 13;
// Set Data
worksheet.Range["A1:F16"].Value = sourceData;
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F16"], true);
table.Columns[3].DataBodyRange.NumberFormat = "$#,##0.00";
// Create slicer cache for table
ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");

// Add two slicers for Category column
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "catel", "Category",
300, 50, 100, 200);

// Saving workbook to pdf
workbook.Save(@"ConvertExcelSlicersToPDFExport.pdf");
```

## Limitations

The following is not supported while exporting slicers to PDF documents:

- Pivot table slicers or report connections
- Custom height of slicer items
- Slicer settings
- Slicer styles (except the color property)
- Slicer header styles
- Scroll viewer which surrounds the items panel
- Slicer item styles for the "No data" visual state group

## Export Barcodes

GcExcel provides support to export barcodes to PDF documents. For more information about supported barcodes and the sample code implementation to export them to PDF documents, refer:

- [QRCode](#)
- [EAN-13](#)
- [EAN-8](#)
- [Codabar](#)
- [Code39](#)
- [Code93](#)
- [Code128](#)
- [GS1-128](#)
- [Code49](#)

- [PDF417](#)
- [Data Matrix](#)

### Limitations

The following parameters in different barcode types have limited or no support while exporting to PDF documents, as is elaborated in the below table:

Barcode Type	Parameter	PDF Export
QRcode	charSet	default "UTF8"
	charCode	Not Supported
PDF417	compact	Not Supported
EAN-13	addOn addOnLabelPosition	Not Supported
codabar	checkDigit	Not Supported
code39	labelWithStartAndStopCharacter checkDigit fullASCII	Not Supported
code93	checkDigit fullASCII	Not Supported

Some barcode types support various font options like fontFamily, fontWeight, fontStyle etc. The following font options have limited support as mentioned:

Font Options	PDF Export
fontStyle	'normal' and 'italic'
fontTextDecoration	'normal' and 'underline'

## Export Signature Lines

GcExcel supports exporting signature lines to PDF documents. The signature lines are exported as images and the exported signature line is different depending upon the validity of certificate. This validity or invalidity is decided by **SkipCertificateValidationOnExporting** property of **ISignatureSet** interface. Its default value is true, meaning that the certificate will be treated as valid. However, you can set it to false to validate the certificate which requires an internet connection.

### Using Code

Refer to the following example code to export signature lines to a PDF document.

```
C#
```

```
//create a new workbook
var workbook = new Workbook();

workbook.Open("Signature.xlsx");
workbook.Signatures.SkipCertificateValidationOnExporting = false;

//save to a pdf file
workbook.Save("exportsignaturelinetopdf.pdf");
```

## Support Sheet Background Image

GcExcel supports sheet background image which can be included while exporting the worksheet to a PDF file. This is very useful for displaying company logos and watermarks in PDF documents.

### Render Background Image

In a worksheet, you can set a background image using the **BackgroundPicture** property of the **IWorksheet** interface. The **PrintBackgroundPicture** property in **PdfSaveOptions** class renders the background image in the center of the page while exporting worksheet to PDF document.

Refer to the following example code to include sheet background image while exporting to PDF document.

```
C#

// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A1"].Value = "GrapeCity Documents for Excel";
worksheet.Range["A1"].Font.Size = 25;

using (FileStream pictureStream = File.Open(@"grapecity.png", FileMode.Open,
FileAccess.Read))
{
    MemoryStream pictureMemoryStream = new MemoryStream();
    pictureStream.CopyTo(pictureMemoryStream);
    byte[] picturebytes = pictureMemoryStream.ToArray();

    //Add background image of the worksheet
    worksheet.BackgroundPicture = picturebytes;
}
PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
//Print the background picture in the centre of exported pdf file
pdfSaveOptions.PrintBackgroundPicture = true;

// Saving workbook to pdf
workbook.Save(@"PrintBackgroundPicture.pdf", pdfSaveOptions);
```



## Render Multiple Background Images

Multiple background images can be rendered in GcExcel using the **BackgroundPictures** property of the **IWorksheet** interface. These images can be included while exporting the worksheet to PDF documents. The background images in PDF are drawn based on the gridlines and can be positioned anywhere in the document by specifying the coordinates of the destination rectangle.

Further, the image transparency, border, corner radius and other formatting options can also be applied. For setting the corner radius, the minimum value is 0 and the maximum value is the height or width (whichever is smaller) of the destination rectangle divided by two. The **ImageLayout** enum can be used to specify the way the image should be placed to fill the destination rectangle in PDF.

GcExcel also supports JSON export of background mages by using **ToJSON** method. However, the image is discarded when exported to Excel.

Refer to the following example code to include multiple background images while exporting to PDF document.

C#

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

//Add two background pictures in the worksheet
IBackgroundPicture picture1 = worksheet.BackgroundPictures.AddPictureInPixel("logo.png",
100, 100, 350, 250);
IBackgroundPicture picture2 =
worksheet.BackgroundPictures.AddPictureInPixel("watermark.png", 180, 10, 150, 100);

//Set the border style of the destination rectangle
picture1.Line.Color.RGB = Color.Red;
picture1.Line.Weight = 1;

//The background picture will be resized to fill the destination dimensions.The aspect
ratio is not preserved.
picture1.BackgroundImageLayout = ImageLayout.Tile;
//Sets the rounded corner of the destination rectangle
picture1.CornerRadius = 50;
//Sets the transparency of the background pictures
picture1.Transparency = 0.5;
picture2.Transparency = 0.5;

//Save to PDF file
workbook.Save("ExportBackgroundImageToPDF.pdf");
```

## Limitation

GcExcel uses the first background image found from the first worksheet to last worksheet while exporting to JSON.

For more information about adding a background image to a worksheet, refer the [Customize Worksheets](#) topic.

## Support Background Color Transparency

When backcolor is applied on a cell or range, any background image or data gets hidden behind it while exporting to PDF.

GcExcel allows you to make the cell's backcolor transparent when exported to PDF by using the **PrintTransparentCell** property of the **PdfSaveOptions** class. The default value of this property is false. When set to true, it prints the transparency of the cell's background color which makes any background image or data visible.

Refer to the following example code to make cell's backcolor transparent to view the background image in PDF document.

```
C#  
  
// Initialize workbook  
Workbook workbook = new Workbook();  
  
// Fetch default worksheet  
IWorksheet worksheet = workbook.Worksheets[0];  
  
// Set the background color of range ["A1:K20"]  
worksheet.Range["A1:K20"].Interior.Color = System.Drawing.Color.FromArgb(50, 255, 0, 0);  
  
// Add a background picture  
IBackgroundPicture picture = worksheet.BackgroundPictures.AddPictureInPixel("image.png",  
0, 0, 300, 200);  
  
// Set the transparency of cell's background color, so the background picture will come  
// out to the front  
PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();  
pdfSaveOptions.PrintTransparentCell = true;  
  
// Save to pdf file  
workbook.Save("PrintTransparentCell.pdf", pdfSaveOptions);
```

## Control Image Quality

GcExcel enables users to control the quality of images while exporting them to PDF documents. The **ImageQuality** property of **PdfSaveOptions** class can be used for the same. The property takes percentage values and its default value is 75. However, it can vary between 0 to 100, depicting the below behavior:

Value	Image Quality	Image Compression
0	Lowest	Maximum
100	Highest	Nil

## Using Code

Refer to the following example code to export an image to PDF with highest image quality.

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.Worksheets[0];

//Add a picture
worksheet.Shapes.AddPictureInPixel("Logo.png", 0, 0, 639, 578);

//Create PdfSaveOptions
PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();

//Set image quality as 100 % (highest quality)
pdfSaveOptions.ImageQuality = 100;

//Save to pdf with PdfSaveOptions
workbook.Save("LogoInPDF.pdf", pdfSaveOptions);
```

## Track Export Progress

GcExcel provides **PagePrinting** and **PagePrinted** events in **PdfSaveOptions** class to track the export progress of a workbook to PDF. The **PagePrinting** event occurs before printing a page and provides **SkipThisPage** property to skip pages while exporting. Similarly, the **PagePrinted** event occurs after printing a page and provides **HasMorePages** property to exit PDF exporting.

## Display Export Progress

Refer to the following example code to display the export progress of a workbook to PDF.

C#

```
//create a pdf file stream
FileStream outputStream = new FileStream("pageprinteventstrackprogress.pdf",
    FileMode.Create);

//create a new workbook
var workbook = new Workbook();

var activeSheet = workbook.ActiveSheet;
activeSheet.Range["A1"].Value = 1;
activeSheet.Range["A2:A100"].FormulaR1C1 = "=R[-1]C+1";
var options = new PdfSaveOptions();
options.PagePrinting += (sender, e) =>
```

```
Console.WriteLine($"Printing page {e.PageNumber} of {e.PageCount}");
activeSheet.PageSetup.CenterHeader = "Page &P of &N";
workbook.Save(outputStream, options);

//close the pdf stream
outputStream.Close();
```

### Skip a Page while Exporting

Refer to the following example code to skip second page while exporting a workbook to PDF.

```
C#

//create a pdf file stream
FileStream outputStream = new FileStream("pageprinteventsskippage.pdf",
    FileMode.Create);

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

var activeSheet = workbook.ActiveSheet;
activeSheet.Range["A1"].Value = 1;
activeSheet.Range["A2:A100"].FormulaR1C1 = "=R[-1]C+1";
var options = new PdfSaveOptions();

//skip second page
options.PagePrinting += (sender, e) =>
{
    if (e.PageNumber == 2)
    {
        e.SkipThisPage = true;
    }
};
activeSheet.PageSetup.CenterHeader = "Page &P of &N";
workbook.Save(outputStream, options);

//close the pdf stream
outputStream.Close();
```

### Exit Exporting

Refer to the following example code to exit PDF exporting after second page.

```
C#

//create a pdf file stream
FileStream outputStream = new FileStream("pageprinteventsexitprinting.pdf",
    FileMode.Create);
```

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

var activeSheet = workbook.ActiveSheet;
activeSheet.Range["A1"].Value = 1;
activeSheet.Range["A2:A100"].FormulaR1C1 = "=R[-1]C+1";
var options = new PdfSaveOptions();

//exit printing after second page
options.PagePrinted += (sender, e) =>
{
    if (e.PageNumber == 2)
    {
        e.HasMorePages = false;
    }
};
activeSheet.PageSetup.CenterHeader = "Page &P of &N";
workbook.Save(outputStream, options);

//close the pdf stream
outputStream.Close();
```

## Export to HTML

Many organizations maintain their product inventories, hiring positions, price lists etc. in Excel files. However, it is very convenient to publish such data on websites to share it with relevant customers. Hence, exporting to HTML files becomes an important feature in such cases.

GcExcel allows users to export a workbook, worksheet or any specific range to an HTML file. By default, it exports an HTML file and a folder containing additional files. These additional files can be images in a worksheet, htm files of other worksheets in a workbook or css file used for styling the html files. However, a single HTML file can also be exported while exporting a worksheet or any range of a worksheet.

With various properties in **HtmlSaveOptions** class, the exported content can be controlled in various ways like exporting headings, gridlines, document properties or apply other settings like scalable width, page title, displaying tooltip text etc.

The **ExportCssSeparately** property in **HtmlSaveOptions** class exports the css file separately (in the additional folder), as its default value is true. However, it can be set to false so that the css style data is exported directly to each worksheet and separate css file is not created.



**Note:** If unlicensed version of GcExcel is used:

- While exporting a workbook to HTML: An "Evaluation warning" sheet is appended to the workbook along with an "Evaluation warning" message at the head of each worksheet.
- While exporting a worksheet or range to HTML: An "Evaluation warning" message is added at the head of worksheet or range file.

### Export workbook to HTML

The **Save** method of **IWorkbook** interface can be used to export a workbook to HTML file.

Refer to the following example code to export workbook to a zip folder containing workbook's HTML file and folder carrying additional files.

C#

```
//create a zip file stream
FileStream outputStream = new FileStream("saveworkbooktohtml.zip", FileMode.Create);

//create a new workbook
var workbook = new Workbook();

workbook.Open("pricelist.xlsx");

//save workbook to html format
workbook.Save(outputStream, SaveFileFormat.Html);

//close the zip stream
outputStream.Close();
```

### Export worksheet to HTML

The **Save** method of **IWorkbook** interface can be used to export a worksheet to HTML file. The headings and gridlines of the worksheet can also be exported by using **ExportHeadings** and **ExportGridlines** properties of **HtmlSaveOptions** class. The **ExportSheetName** property can be used to define which worksheet needs to be exported.

Refer to the following example code to export worksheet to a zip folder containing worksheet's HTML file and a folder carrying additional files.

C#

```
//create a zip file stream
FileStream outputStream = new FileStream("saveworksheettohtml.zip", FileMode.Create);

//create a new workbook
var workbook = new Workbook();

workbook.Open("hiringpositions.xlsx");

HtmlSaveOptions options = new HtmlSaveOptions();

//set exporting row/column headings
options.ExportHeadings = true;

//set exporting gridlines
options.ExportGridlines = true;

//export first sheet
options.ExportSheetName = workbook.Worksheets[0].Name;
```

```
//set exported html file name
options.ExportFileName = "hiringdetails";

workbook.Save(outputStream, options);

//close the zip stream
outputStream.Close();
```

A worksheet can also be exported to a single HTML file when the specific properties of **HtmlSaveOptions** class are set, as in the code below.

```
C#

Workbook workbook = new Workbook();

//open an excel file
workbook.Open("productlist.xlsx");

IWorksheet worksheet = workbook.Worksheets[0];

//create HtmlSaveOptions
HtmlSaveOptions options = new HtmlSaveOptions();

//export first sheet
options.ExportSheetName = workbook.Worksheets[0].Name;

//set exported image as base64
options.ExportImageAsBase64 = true;

//set exported css style in html file
options.ExportCssSeparately = false;

//set not to export single tab in html
options.ExportSingleTab = false;

//save first worksheet to html
workbook.Save("saveworksheettosinglehtml.html", options);
```

### Export worksheet range to HTML

The **Save** method of **IWorkbook** interface can be used to export any range of a worksheet to HTML file. The **ExportArea** property of **HtmlSaveOptions** class can be used to define the range which needs to be exported.

Refer to the following example code to export range in a worksheet to a zip folder containing range's HTML file and a folder carrying additional files.

```
C#
```

```
//create a zip file stream
FileStream outputStream = new FileStream("saverangetohtml.zip", FileMode.Create);

//create a new workbook
var workbook = new Workbook();

workbook.Open("projecttracker.xlsx");

HtmlSaveOptions options = new HtmlSaveOptions();

//export first sheet
options.ExportSheetName = workbook.Worksheets[0].Name;

//set export area
options.ExportArea = "D2:G23";

//set exported html file name
options.ExportFileName = "range";

//set html with scalable width
options.IsWidthScalable = true;

workbook.Save(outputStream, options);

//close the zip stream
outputStream.Close();
```

A range in a worksheet can also be exported to a single HTML file when the specific properties of **HtmlSaveOptions** class are set, as in the code below.

#### C#

```
Workbook workbook = new Workbook();

//open an xlsx file
workbook.Open("netprofit.xlsx");
IWorksheet worksheet = workbook.Worksheets[0];

//create HtmlSaveOptions
HtmlSaveOptions options = new HtmlSaveOptions();

//specify exported sheet name
options.ExportSheetName = workbook.Worksheets[0].Name;

//set export area
options.ExportArea = "D2:G23";
```



```
//set exported image as base64
options.ExportImageAsBase64 = true;

//set exported css style in html file
options.ExportCssSeparately = false;

//set not to export single tab in html
options.ExportSingleTab = false;

//save the specified range of first worksheet to html
workbook.Save("range.html", options);
```

### Limitations

The following features are not supported while exporting to html file:

- Vertical text
- Picture's texture or picture fill
- Pictures's brightness or contrast
- Shape with gradient line
- Shape with rectangular gradient fill and path gradient fill
- Text alignment in shape-like distribution
- Font: Alignment preferences like Center across selection, Fill alignment, Orientation, Text reading order etc.
- Shrink to fit
- Chart
  - The following chart types are not supported: 3D, Pie of Pie, Bar of Pie, Combo chart with Pie or Doughnut or Radar, Map, Treemap, Sunburst, Histogram, Box & Whisker, Waterfall and Funnel.
  - The chart layout, major unit and min or max value of the chart axis may not be exactly same as Excel after exporting to HTML file.
  - The following chart features are not supported:
    - Line marker and round corners
    - Chart elements: Stock's upBar, data table and label callout
    - Fill settings: Gradient, Picture or Texture and Pattern
- Slicer
  - Pivot table slicers or report connections
  - Custom height of slicer items
  - Slicer settings
  - Slicer styles (except the color property)
  - Slicer header styles
  - Scroll viewer which surrounds the items panel
  - Slicer item styles for "No data" visual state groups

## 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** class and the **CsvSaveOptions** class in order to configure several open and save options in a workbook.

Settings	Description
<b>CsvOpenOptions.ConvertNumericData</b>	This property can be used to get or set a value that indicates whether the string in text file is converted to numeric data.
<b>CsvOpenOptions.ConvertDateTimeData</b>	This property can be used to get or set a value that indicates whether the string in text file is converted to date data.
<b>CsvOpenOptions.SeparatorString</b>	This property can be used to get or set the string value as a separator.
<b>CsvOpenOptions.Encoding</b>	This property can be used to get or set the default encoding which is UTF-8.
<b>CsvOpenOptions.ParseStyle</b>	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.
<b>CsvOpenOptions.HasFormula</b>	This property can be used to specify whether the text is formula if it starts with "=".
<b>CsvSaveOptions.SeparatorString</b>	This property can be used to get or set the string value as the separator. By default, this value is a comma separator.
<b>CsvSaveOptions.Encoding</b>	This property can be used to specify the default encoding which is UTF-8.
<b>CsvSaveOptions.ValueQuoteType</b>	This property can be used to get or set how to quote values in the exported text file.
<b>CsvSaveOptions.TrimLeadingBlankRowAndColumn</b>	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
<b>CsvOpenOptions.ColumnSeparator</b>	This property can be used to get or set the column delimiter while opening CSV files.
<b>CsvSaveOptions.ColumnSeparator</b>	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
<b>CsvOpenOptions.RowSeparator</b>	This property can be used to get or set the row delimiter while opening CSV files.
<b>CsvSaveOptions.RowSeparator</b>	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
<b>CsvOpenOptions.CellSeparator</b>	This property can be used to get or set the cell delimiter while opening CSV files.
<b>CsvSaveOptions.CellSeparator</b>	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** 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-OpenCSVDelimeterRowColumnCell.csv");
```

Refer to the following example code in order to import and export CSV files with delimiters using **CsvSaveOptions** 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

GcExcel .NET supports the import and export of a json stream using .NET core.

### Import and Export JSON Stream for Workbook

You can export a workbook to a json string/stream using the **ToJson method** of the **IWorkbook interface**. You can also import a json string or stream to your workbook using the **FromJson method** 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 spreadjs 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 spreadjs product.
//spreadjs will show the excel file contents.

//spreadjs 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);
```

### Import and Export JSON String for Worksheet

You can export the information in a worksheet to a json string using the **ToJson** method of the **IWorksheet** interface. Similarly, you can also import a json string to your worksheet using the **FromJson** of the **IWorksheet** interface. The worksheet can also be exported or imported to the same or another workbook.

It also enables you to view a large Excel file in SpreadJS. The Excel file can be opened in GcExcel and the json string of a worksheet can be exported using the **ToJson** method. Further, the json string of the worksheet can be transferred to client to be loaded in SpreadJS.

#### Limitations

- Importing worksheet json to another workbook on server might cause data loss or conflict
- Cell styles used in SpreadJS ssjson are lost in Excel after using Worksheet.toJSON()
- SpreadJS doesn't support all the page settings of Excel. Hence, GcExcel does not get all the settings when imported from ssjson.

Refer to the following example code to export and import json string of a worksheet.

```
C#
var workbook = new GrapeCity.Documents.Excel.Workbook();

//ToJson&FromJson can be used in combination with spreadjs
product:http://spread.grapecity.com/spreadjs/sheets/

//GrapeCity Documents for Excel import an excel file
string source = "ExcelJsonInput.xlsx";
workbook.Open(source);

//Open the file
GrapeCity.Documents.Excel.Workbook new_workbook = new
GrapeCity.Documents.Excel.Workbook();
new_workbook.Open(source);

foreach (IWorksheet worksheet in workbook.Worksheets)
```

```

{
    worksheet.Range["D40:F40"].Value = new string[] { "Device", "Quantity", "Unit Price"
};
    worksheet.Range["D41:F44"].Value = new object[, ]
{ { "T540p", 12, 9850 },
  { "T570", 5, 7460 },
  { "Y460", 6, 5400 },
  { "Y460F", 8, 6240 } };

    //GrapeCity Documents for Excel export a worksheet to json string
    string json = worksheet.ToJson();

    //You can use the json string to initialize spreadjs product
    //Product spreadjs will show the excel file contents
    //You can use spreadjs product export a json string of worksheet

    //GrapeCity Documents for Excel use the json string to update content of the
    corresponding worksheet
    new_workbook.Worksheets[worksheet.Name].FromJson(json);
}

//GrapeCity Documents for Excel export workbook to an excel file
string export = "ExcelJsonOutput.xlsx";
new_workbook.Save(export);

```

### Retrieve Errors while Importing JSON Files

GcExcel provides the option to get JSON errors, if any, while importing the JSON file using **FromJson** method of **IWorkbook** interface. The error message is displayed by the **ErrorMessage** property of **JsonError** class. Two types of error messages are supported:

- Formula JSON Error - Implemented using the **FormulaJsonError** class and can be raised in case of a formula error in JSON file
- Data Validation JSON Error - Implemented using the **DataValidationJsonError** class and can be raised in case of a data validation error in JSON file

Refer to the below example code which will display a formula JSON error as the JSON file containing formula error is imported in GcExcel.

```

C#

Workbook workbook = new Workbook();
IList<JsonError> errors = workbook.FromJson(File.OpenRead("ErrorJson.json"));
foreach (JsonError item in errors)
{
    if (item is FormulaJsonError)
    {
        FormulaJsonError fError = item as FormulaJsonError;
        Console.WriteLine(fError.ErrorMessage + " " +

```

```
workbook.Worksheets[fError.WorksheetName].Range[fError.Row, fError.Column].ToString() +  
" " + fError.Formula);  
}  
if (item is DataValidationJsonError)  
{  
    DataValidationJsonError dError = item as DataValidationJsonError;  
    Console.WriteLine(dError.ErrorMessage + " " +  
workbook.Worksheets[dError.WorksheetName].Range[dError.Range.ToString()] + " " +  
dError.ErrorContent);  
}  
}
```

### Limitation

If the data validation in JSON file has error in its formula, Data Validation JSON error will be generated.



## Import and Export SpreadJS Files

GcExcel .NET support the JSON I/O of [SpreadJS](#) files. You can also import an ssjson file created with SpreadJS Designer and save it back after modifying it as per your preferences.

The below example code loads an ssjson file and then saves it to xlsx format.

C#

```
//Create a new workbook  
Workbook workbook = new Workbook();  
  
//Load SSJSON file  
var stream = new System.IO.FileStream("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** property of the **IBorder** interface in order to set an index color, it will return a valid value only if the **Color** 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 a versa is not possible.

The below mentioned features are supported for JSON I/O by GcExcel. You can use **FromJson** and **ToJson** methods for the same, as is also demonstrated in the sample code above.

### Shapes

GcExcel .NET allows you to perform JSON I/O of SpreadJS files containing shapes. You can also download the JSON file containing shape from [here](#).



## Barcodes

GcExcel supports JSON I/O and PDF export of SpreadJS files containing barcodes. However while exporting to PDF, partial SpreadJS barcode properties are supported. To know more about unsupported properties, refer [Export Barcodes](#).

You can also download the JSON file containing barcodes from [here](#).

## Cell Buttons

SpreadJS files containing cell buttons are supported by GcExcel for JSON I/O, HTML, image and PDF exporting. You can also download the JSON file containing cell buttons from [here](#).

## Cell Dropdowns

GcExcel supports JSON I/O of SpreadJS files containing cell dropdowns like calculator, color picker, time picker etc. You can also download the JSON file containing cell dropdowns from [here](#).

## Validation Styles

Validation styles can be used to highlight invalid data in a worksheet. GcExcel supports JSON I/O, image and PDF exporting of SpreadJS files containing validation styles. You can also download the JSON file containing validation style from [here](#).

## Text Ellipsis

When text in a cell is longer than the column width, SpreadJS allows you to show ellipsis instead of overflowing text in the other cell. The SpreadJS files containing text ellipsis are supported for JSON I/O and PDF exporting in GcExcel. You can also download the JSON file containing text ellipsis from [here](#).

## Limitation

SpreadJS allows different types of text alignment composed with text ellipsis but GcExcel does not. Hence, text ellipsis is only shown at the end of text in exported PDF.

## Range Template

In SpreadJS, you can create a range cell type which can be used to specify a cell range in the worksheet as a template. You can modify the display mode and appearance of the resultant data just by changing the template. GcExcel supports JSON I/O and PDF exporting of SpreadJS files containing [Range templates](#).

You can also download the JSON file containing range template from [here](#).

## Format String

SpreadJS supports [Format string](#) feature which allows cells to have both formulas and text as a part of text value templates. GcExcel supports JSON I/O of SpreadJS files containing format strings.

You can also download the JSON file containing format string from [here](#).

## JSON Options

In SpreadJS, while importing or exporting custom data from/to a JSON object, you can set several serialization or

deserialization options. GcExcel also supports some of these options for workbook and worksheet JSON I/O and in GcExcel API. The below table explains the supported options in SpreadJS and GcExcel.

	SpreadJS (toJSON and fromJSON)	GcExcel (ToJSON and FromJSON)
Serialization	ignoreStyle ignoreFormula rowHeadersAsFrozenColumns columnHeadersAsFrozenRows	ignoreStyle ignoreFormula
Deserialization	ignoreStyle ignoreFormula frozenColumnsAsRowHeaders frozenRowsAsColumnHeaders doNotRecalculateAfterLoad	ignoreStyle ignoreFormula doNotRecalculateAfterLoad

GcExcel provides **SerializationOptions** and **DeserializationOptions** classes in API with above-mentioned supported properties.

The following example code serializes a workbook to JSON with options in GcExcel.

C#

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

var worksheet = workbook.Worksheets[0];

worksheet.Range["B3:C16"].Value = new object[,]
{
    { "ITEM", "AMOUNT" },
    { "Rent/mortgage", 800 },
    { "Electric", 120 },
    { "Gas", 50 },
    { "Cell phone", 45 },
    { "Groceries", 500 },
    { "Car payment", 273 },
    { "Auto expenses", 120 },
    { "Student loans", 50 },
    { "Credit cards", 100 },
    { "Auto Insurance", 78 },
    { "Personal care", 50 },
    { "Entertainment", 100 },
    { "Miscellaneous", 50 },
};
```

```
// Create a table
ITable expensesTable = worksheet.Tables.Add(worksheet.Range["B3:C16"], true);
expensesTable.Name = "tblExpenses";
worksheet.Range["C3:C16"].NumberFormat = "$#,##0_); ($#,##0)";

worksheet.Range["B2:C2"].Merge();
worksheet.Range["B2"].Value = "MONTHLY EXPENSES";
worksheet.Range["B2"].Interior.Color = System.Drawing.Color.FromArgb(219, 219, 219);
worksheet.Range["E2"].Value = "Total Monthly Expenses";
worksheet.Range["E3"].Formula = "SUM(tblExpenses[AMOUNT])";
worksheet.Range["E3"].NumberFormat = "$#,##0_); ($#,##0)";

worksheet.Range["B:B"].ColumnWidth = 15;
worksheet.Range["C:C"].ColumnWidth = 15;
worksheet.Range["E:F"].ColumnWidth = 15;

string json = workbook.ToJson();

// ignore style and formula when deserialize workbook from json.
workbook.FromJson(json, new DeserializationOptions() { IgnoreFormula = true,
IgnoreStyle = true });

//save to an excel file
workbook.Save("fromjsonwithoptions.xlsx");
```

The following example code deserializes a workbook from JSON with options in GcExcel.

C#

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

var worksheet = workbook.Worksheets[0];

worksheet.Range["B3:C16"].Value = new object[,]
{
    { "ITEM", "AMOUNT" },
    { "Rent/mortgage", 800 },
    { "Electric", 120 },
    { "Gas", 50 },
    { "Cell phone", 45 },
    { "Groceries", 500 },
    { "Car payment", 273 },
    { "Auto expenses", 120 },
    { "Student loans", 50 },
    { "Credit cards", 100 },
    { "Auto Insurance", 78 },
    { "Personal care", 50 },
}
```

```

{ "Entertainment", 100 },
{ "Miscellaneous", 50 },
};

// Create a table
ITable expensesTable = worksheet.Tables.Add(worksheet.Range["B3:C16"], true);
expensesTable.Name = "tblExpenses";
worksheet.Range["C3:C16"].NumberFormat = "$#,##0_); ($#,##0)";

worksheet.Range["B2:C2"].Merge();
worksheet.Range["B2"].Value = "MONTHLY EXPENSES";
worksheet.Range["B2"].Interior.Color = System.Drawing.Color.FromArgb(219, 219, 219);
worksheet.Range["E2"].Value = "Total Monthly Expenses";
worksheet.Range["E3"].Formula = "SUM(tblExpenses[AMOUNT])";
worksheet.Range["E3"].NumberFormat = "$#,##0_); ($#,##0)";

worksheet.Range["B:B"].ColumnWidth = 15;
worksheet.Range["C:C"].ColumnWidth = 15;
worksheet.Range["E:F"].ColumnWidth = 15;

// ignore style and formula when serialize workbook to json
string jsonWithOptions = workbook.ToJson(new SerializationOptions() { IgnoreFormula =
true, IgnoreStyle = true });

workbook.FromJson(jsonWithOptions);

//save to an excel file
workbook.Save("tojsonwithoptions.xlsx");

```



**Note:** SpreadJS supports multi-level row or column headers but GcExcel does not. However, you can still retain the header information in GcExcel by following the below steps:

1. Use SpreadJS to export JSON with 'rowHeadersAsFrozenColumns or columnHeadersAsFrozenRows' option as true to convert multi-header to frozen area, and use GcExcel to load the JSON file.
2. Manipulate the frozen area in GcExcel.
3. Use GcExcel to export JSON file, and use SpreadJS to load JSON file with 'frozenColumnsAsRowHeaders or frozenRowsAsColumnHeaders' option as true to convert frozen area to header.

## Checkbox or Radiobutton List Cell Type

GcExcel supports JSON I/O and PDF exporting of SpreadJS files containing checkbox list and radiobutton list cell types. You can also download the JSON file containing radiobutton list and checkbox list cell type from [here](#).

GcExcel also provides **RadioButtonListCellType** and **CheckBoxListCellType** classes in its API to add these cell types.

The following example code creates a checkbox list cell type for a cell in GcExcel.

C#

```
//create a new workbook
```

```
var workbook = new GrapeCity.Documents.Excel.Workbook();
IWorksheet worksheet = workbook.Worksheets[0];

CheckBoxListCellType cellType = new CheckBoxListCellType
{
    Direction = CellTypeDirection.Horizontal,
    TextAlign = CellTypeTextAlign.Right,
    IsFlowLayout = false,
    MaxColumnCount = 2,
    MaxRowCount = 1,
    HorizontalSpacing = 20,
    VerticalSpacing = 5,
};
cellType.Items.Add(new SelectFieldItem("sample1", "1"));
cellType.Items.Add(new SelectFieldItem("sample2", "2"));
cellType.Items.Add(new SelectFieldItem("sample3", "3"));
cellType.Items.Add(new SelectFieldItem("sample4", "4"));
cellType.Items.Add(new SelectFieldItem("sample5", "5"));
worksheet.Range["A1"].RowHeight = 60;
worksheet.Range["A1"].ColumnWidth = 25;
worksheet.Range["A1"].CellType = cellType;

//check multiple options in the check box list
worksheet.Range["A1"].Value = new object[,]
{
    {new object[] {"1", "3", "5" } }
};

//save to a pdf file
workbook.Save("addcheckboxlistcelltype.pdf");
```

The following example code creates checkbox list cell type and sets the value of the option as a custom object.

C#

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

GrapeCity.Documents.Excel.Workbook.ValueJsonSerializer = new
CustomObjectJsonSerializer();
IWorksheet worksheet = workbook.Worksheets[0];

CheckBoxListCellType cellType = new CheckBoxListCellType
{
    Direction = CellTypeDirection.Horizontal,
    TextAlign = CellTypeTextAlign.Right,
    IsFlowLayout = false,
    MaxColumnCount = 2,
```

```
        MaxRowCount = 1,
        HorizontalSpacing = 20,
        VerticalSpacing = 5,
    };
    cellType.Items.Add(new SelectFieldItem("player1", new People { Name = "Tom", Age = 5
    }));
    cellType.Items.Add(new SelectFieldItem("player2", new People { Name = "Jerry", Age = 3
    }));
    cellType.Items.Add(new SelectFieldItem("player3", new People { Name = "Mario", Age = 6
    }));
    cellType.Items.Add(new SelectFieldItem("player4", new People { Name = "Luigi", Age = 4
    }));
    worksheet.Range["A1"].RowHeight = 60;
    worksheet.Range["A1"].ColumnWidth = 25;
    worksheet.Range["A1"].CellType = cellType;

    worksheet.Range["A1"].Value = new object[,]
    {
        {new object[] { new People { Name = "Tom", Age = 5 }, new People { Name = "Mario", Age =
        6 }} }
    };

    //save to a pdf file
    workbook.Save("addcheckboxlistcelltypecustomobject.pdf");
}

class CustomObjectJsonSerializer : IJsonSerializer
{
    public object Deserialize(string json)
    {
        return Newtonsoft.Json.JsonConvert.DeserializeObject<People>(json);
    }

    public string Serialize(object value)
    {
        if (value is People)
        {
            return Newtonsoft.Json.JsonConvert.SerializeObject(value);
        }
        return null;
    }
}

class People
{
    private int age;
    private string name;

    public int Age { get => age; set => age = value; }
    public string Name { get => name; set => name = value; }
```

```
public override bool Equals(object obj)
{
    return obj is People people &&
        age == people.age &&
        name == people.name;
}

public override int GetHashCode()
{
    return age.GetHashCode() ^ name.GetHashCode();
}
}
```

The following example code creates a radio list cell type for a cell in GcExcel.

C#

```
//create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();
IWorksheet worksheet = workbook.Worksheets[0];

RadioButtonListCellType cellType = new RadioButtonListCellType
{
    Direction = CellTypeDirection.Horizontal,
    TextAlign = CellTypeTextAlign.Right,
    IsFlowLayout = false,
    MaxColumnCount = 2,
    MaxRowCount = 1,
    HorizontalSpacing = 20,
    VerticalSpacing = 5,
};
cellType.Items.Add(new SelectFieldItem("sample1", "1"));
cellType.Items.Add(new SelectFieldItem("sample2", "2"));
cellType.Items.Add(new SelectFieldItem("sample3", "3"));
cellType.Items.Add(new SelectFieldItem("sample4", "4"));
cellType.Items.Add(new SelectFieldItem("sample5", "5"));
worksheet.Range["A1"].RowHeight = 60;
worksheet.Range["A1"].ColumnWidth = 25;
worksheet.Range["A1"].CellType = cellType;
worksheet.Range["A1"].Value = 1;

//check multiple options in the radio button list
worksheet.Range["A1"].Value = new object[,]
{
    { new object[] { "1", "3", "5" } }
};
```

```
//save to a pdf file
workbook.Save("addradiobuttonlistcelltype.pdf");
```

The following example code creates radiobutton cell type and sets the value of the option as a custom object.

C#

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

GrapeCity.Documents.Excel.Workbook.ValueJsonSerializer = new
CustomObjectJsonSerializer();
IWorksheet worksheet = workbook.Worksheets[0];

RadioButtonListCellType cellType = new RadioButtonListCellType
{
    Direction = CellTypeDirection.Horizontal,
    TextAlign = CellTypeTextAlign.Right,
    IsFlowLayout = false,
    MaxColumnCount = 2,
    MaxRowCount = 1,
    HorizontalSpacing = 20,
    VerticalSpacing = 5,
};
cellType.Items.Add(new SelectFieldItem("player1", new People { Name = "Tom", Age = 5
}));
cellType.Items.Add(new SelectFieldItem("player2", new People { Name = "Jerry", Age = 3
}));
cellType.Items.Add(new SelectFieldItem("player3", new People { Name = "Mario", Age = 6
}));
cellType.Items.Add(new SelectFieldItem("player4", new People { Name = "Luigi", Age = 4
}));
worksheet.Range["A1"].RowHeight = 60;
worksheet.Range["A1"].ColumnWidth = 25;
worksheet.Range["A1"].CellType = cellType;

worksheet.Range["A1"].Value = new object[,]
{
    {
        new People { Name = "Tom", Age = 5 } }
    };
};

//save to a pdf file
workbook.Save("addradiobuttoncelltypecustomobject.pdf");
}

class CustomObjectJsonSerializer : IJsonSerializer
{
```



```
public object Deserialize(string json)
{
    return Newtonsoft.Json.JsonConvert.DeserializeObject<People>(json);
}

public string Serialize(object value)
{
    if (value is People)
    {
        return Newtonsoft.Json.JsonConvert.SerializeObject(value);
    }
    return null;
}
}

class People
{
    private int age;
    private string name;

    public int Age { get => age; set => age = value; }
    public string Name { get => name; set => name = value; }

    public override bool Equals(object obj)
    {
        return obj is People people &&
            age == people.age &&
            name == people.name;
    }

    public override int GetHashCode()
    {
        return age.GetHashCode() ^ name.GetHashCode();
    }
}
```

### Cell Padding and Labels

GcExcel allows you to perform JSON I/O and PDF exporting for SpreadJS files containing cell padding and labels. You can also download the JSON file containing cell padding and labels from [here](#).

In addition to this, GcExcel also provides **CellPadding** and **Margin** class, **ILabelOptions** interface, **LabelAlignment** and **LabelVisibility** enumerations to support cell padding and labels in GcExcel.

The following example code adds cell padding and labels in a GcExcel worksheet.

```
C#

// create a new workbook
Workbook workbook = new Workbook();
// get the sheet
```

```
IWorksheet worksheet = workbook.Worksheets[0];
// set row height
worksheet.Range["A:A"].RowHeight=40;
// set column width
worksheet.Range["A:A"].ColumnWidth=25;
// set watermark
worksheet.Range["A1"].Watermark="GcExcel JAVA";
// set cell padding
worksheet.Range["A1"].CellPadding=new CellPadding(50, 0, 0, 0);
// set label options
worksheet.Range["A1"].LabelOptions.Visibility = LabelVisibility.visible;
worksheet.Range["A1"].LabelOptions.ForeColor = Color.Green;
worksheet.Range["A1"].LabelOptions.Margin=new Margin(15, 0, 0, 0);
worksheet.Range["A1"].LabelOptions.Font.Size=14;
worksheet.Range["A1"].LabelOptions.Font.Name="Calibri";
worksheet.Range["A1"].Borders.LineStyle=BorderLineStyle.Thin;

// save to a pdf file
workbook.Save("CellPaddingAndLabels.pdf");
```

## Background Image

GcExcel supports JSON I/O and PDF exporting of SpreadJS files containing background images. You can also download the JSON file containing background image from [here](#).

GcExcel also provides **BackgroundPictures** property in IWorksheet interface to add background pictures in GcExcel. For more information, refer [Support Sheet Background Image](#).

The following example code sets background image in GcExcel worksheet.

```
C#

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

IWorksheet worksheet = workbook.Worksheets[0];

FileStream stream = File.Open(@"Logo.png", FileMode.Open, FileAccess.Read);

//Add background picture
IBackgroundPicture picture = worksheet.BackgroundPictures.AddPictureInPixel(stream,
ImageType.PNG, 10, 10, 500, 370);
//Set image layout
picture.BackgroundImageLayout = ImageLayout.Zoom;

//Set options
workbook.ActiveSheet.PageSetup.PrintGridlines = true;

//save to a pdf file
workbook.Save("backgroundimage.pdf");
```

The following example code imports background image from JSON and exports to PDF document.

```
C#  
  
Workbook workbook = new Workbook();  
  
string ssjson = string.Empty;  
try  
{  
    var jsonFile = @"D:\bgimage.ssjson";  
  
    using (StreamReader sr = System.IO.File.OpenText(jsonFile))  
    {  
        ssjson = sr.ReadToEnd();  
    }  
}  
catch (Exception e)  
{  
    Console.WriteLine(e);  
}  
  
//Importing from ssjson  
workbook.FromJson(ssjson);  
  
//Set options  
workbook.ActiveSheet.PageSetup.PrintGridlines = true;  
  
//Exporting to PDF  
workbook.Save("bgimage.pdf");
```

### Limitations

- While importing from JSON, the background image is placed at the (left : 0, top: 0) location of each worksheet.
- After exporting to PDF, all pages of PDF document will have the same background image as was imported from ssjson

### Background Color

GcExcel supports JSON I/O and PDF exporting of SpreadJS files containing background color. You can also download the JSON file containing background color from [here](#).

GcExcel also provides **BackColor** and **GrayAreaBackColor** properties in IWorkbookView interface to set background color in GcExcel.

The following code example sets background color for all the worksheets in GcExcel.

```
C#  
  
//create a new workbook  
var workbook = new GrapeCity.Documents.Excel.Workbook();
```

```
//Set background color
workbook.BookView.BackColor = Color.LightSkyBlue;
workbook.BookView.GrayAreaBackColor = Color.Gray;

//Set value to a cell
IWorksheet worksheet = workbook.ActiveSheet;
worksheet.Range["H20"].Value = "The text";

//Set page options
worksheet.PageSetup.PrintGridlines = true;
worksheet.PageSetup.PrintHeadings = true;

//save to a pdf file
workbook.Save("backgroundcolor.pdf");
```

### Limitation

In SpreadJS, background image always overrides the background color. Thus, the background image needs to be removed for the background color to take effect while exporting to PDF documents.

## Support for SpreadJS Features

The following table describes the SpreadJS features supported by GcExcel either in its API or for JSON I/O or PDF export.

Scope	SpreadJS Features	JSON I/O	GcExcel API	PDF Export
Workbook	numbersFitMode	Yes	No	No
	tabNavigationVisible	Yes	No	No
	backColor	Yes	Yes	Yes
	backgroundImage	Yes	Yes	Yes
	backgroundImageLayout	Yes	Yes	Yes
	highlightInvalidData	Yes	No	Yes
	grayAreaBackColor	Yes	Yes	Yes
	scrollbarAppearance	Yes	No	No

Scope	SpreadJS Features	JSON I/O	GcExcel API	PDF Export
Worksheet	frozenTrailingRowCount	Yes	Yes	No
	frozenTrailingColCount	Yes	Yes	No
	rowCount	Yes	No	No
	columnCount	Yes	No	No
	rowHeaderColCount	Yes	No	No

Scope	SpreadJS Features	JSON I/O	GcExcel API	PDF Export
	colHeaderRowCount	Yes	No	No
	tag	Yes	Yes	No
	rowHeaderData	Yes	No	No
	colHeaderData	Yes	No	No
	autoGenerateColumns	Yes	Yes	No
	showRowOutline	Yes	Yes	Yes
	showColumnOutline	Yes	Yes	Yes
	frozenlineColor	Yes	Yes	No
	rowHeaderAutoText	Yes	No	No
	colHeaderAutoText	Yes	No	No
	outlineColumnOptions	Yes	Yes	Yes
	autoMergeRangeInfos	Yes	No	No

Scope	SpreadJS Features	JSON I/O	GcExcel API	PDF Export
Cell	tag	Yes	Yes	No
	bindingPath	Yes	Yes	No
Style	ShowEllipsis	Yes	No	Yes
	cellTypes: <ul style="list-style-type: none"> <li>RangeTemplateCellType</li> <li>CheckBoxListCellType</li> <li>RadioButtonListCellType</li> <li>ButtonCellType</li> <li>CheckBoxCellType</li> <li>ComboBoxCellType</li> <li>HyperLinkCellType</li> <li>BaseCellType</li> </ul>	Yes	Yes	Yes
	watermark	Yes	Yes	Yes
	cellPadding	Yes	Yes	Yes
	labelOptions	Yes	Yes	Yes
	cellButton	Yes	No	Yes
	dropdownlist	Yes	No	Yes


## 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** is Xlsm, macros will be imported. When the **SaveFileFormat** 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.xlsm");

// 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");
```

## Convert to Image

GcExcel allows you to convert a worksheet, any specified range and various shape types to images. Hence, making it convenient to use the converted images directly in other documents, like Word, PDF or a PPT document. The supported image formats for conversion are PNG, JPG/JPEG and GIF.



**Note:** To convert images with transparency, PNG like image format should be used as GIF format doesn't support image transparency.

## Convert Worksheet to Image

A worksheet can be converted to image using the **ToImage** method of **IWorksheet** interface. The converted image displays the rectangular area of the worksheet enclosed under cell A1 and the last cell where any data or shape is present. For eg, if a worksheet contains a shape or data in the range D5:F9, the converted image will display the area under the range A1:F9.

A blank worksheet cannot be converted to image.

Refer to the following example code to convert a worksheet to image.

```
C#
```

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

//Add data
worksheet.Range["S50"].Value = 10;

//Save worksheet to image
worksheet.ToImage("ConvertWorksheetToImage.png");
```

Refer to the following example code to convert a worksheet to image from existing file.

```
C#

//Create a png file stream
FileStream outputStream = new FileStream("ConvertWorksheetToImage.png",
    FileMode.Create);

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

FileStream fileStream = new FileStream("Workbook.xlsx", FileMode.Open);

//Open a xlsx file
workbook.Open(fileStream);
IWorksheet worksheet = workbook.Worksheets[0];

//Export the worksheet to image
worksheet.ToImage(outputStream, ImageType.PNG);

//Close the image stream
outputStream.Close();
```

### Convert Range to Image

A specific range in a worksheet can be converted to image using the **ToImage** method of the **IRange** interface. The resulting image displays the rectangular area of the worksheet enclosed under the specified range.

Refer to the following example code to convert a specified range to image.

```
C#

//create a new workbook
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];

//Add data
worksheet.Range["D10:F10"].Value = new string[] { "Device", "Quantity", "Unit Price" };
worksheet.Range["D11:F14"].Value = new object[,]
{ { "T540p", 12, 9850 },
```



```
{ "T570", 5, 7460 },
{ "Y460", 6, 5400 },
{ "Y460F", 8, 6240 } };

IRange range = worksheet.Range["D10:F14"];

//Save range to image
range.ToImage("ConvertRangeToImage.png");
```

Refer to the following example code to convert a specified range to image from an existing file.

```
C#

//Create a png file stream
FileStream outputStream = new FileStream("ConvertRangeToImage.png", FileMode.Create);

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

FileStream fileStream = new FileStream("RangeWorkbook.xlsx", FileMode.Open);

//Open a xlsx file contains data in a range
workbook.Open(fileStream);
IWorksheet worksheet = workbook.Worksheets[0];

//Export the range to image
worksheet.Range["A1:C5"].ToImage(outputStream, ImageType.PNG);

//Close the image stream
outputStream.Close();
```

## Convert Shape to Image

GcExcel allows you to convert various shape types to image using the **ToImage** method of the **IShape** interface. The shape types include shapes like chart, picture, slicer and autoshape. The resulting image displays the rectangular area of the worksheet enclosed under the shape.

Refer to the following example code to convert an autoshape to image.

```
C#

//create a new workbook
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];

//Add an oval
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Oval, 20, 20, 200, 100);

//Save oval to image
shape.ToImage("ConvertShapeToImage.png");
```

Refer to the following example code to convert an autoshape to image from existing file.

C#

```
//create a png file stream
FileStream outputStream = new FileStream("ConvertShapeToImage.png", FileMode.Create);

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

FileStream fileStream = new FileStream("ShapeWorkbook.xlsx", FileMode.Open);

//Open a xlsx file contains a group shape
workbook.Open(fileStream);
IWorksheet worksheet = workbook.Worksheets[0];

//Export the shape to image
worksheet.Shapes[0].ToImage(outputStream, ImageType.PNG);

//close the image stream
outputStream.Close();
```

Refer to the following example code to convert a chart to image.

C#

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

// Prepare data for chart
worksheet.Range["A1:D4"].Value = new object[,]
{
    {null, "Q1", "Q2", "Q3"},
    {"Mobile Phones", 1330, 2345, 3493},
    {"Laptops", 2032, 3632, 2197},
    {"Tablets", 6233, 3270, 2030}
};

worksheet.Range["A:D"].Columns.AutoFit();

// Add Area Chart
IShape shape = worksheet.Shapes.AddChart(ChartType.Area, 250, 20, 360, 230);

// Add series to SeriesCollection
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"], RowCol.Columns, true, true);

// Configure Chart Title
```

```
shape.Chart.ChartTitle.TextFrame.TextRange.Paragraphs.Add("Annual Sales Record");

//Save chart to image
shape.ToImage("ConvertChartToImage.png");
```

Refer to the following example code to convert a chart to image from existing file.

C#

```
//create a png file stream
FileStream outputStream = new FileStream("ConvertChartToImage.png", FileMode.Create);

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

FileStream fileStream = new FileStream("ScatterChart.xlsx", FileMode.Open);

//Open a xlsx file contains a chart
workbook.Open(fileStream);
IWorksheet worksheet = workbook.Worksheets[0];

//Export the chart to image
worksheet.Shapes[0].ToImage(outputStream, ImageType.PNG);

//close the image stream
outputStream.Close();
```

## 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	Description
<a href="#">Grapacity.Documents.Excel</a>	Provides the namespaces for the GcExcel .NET component functionality for .Net Core.

For help with using the product, refer to the [Key Features](#).

## Release Notes

### Current Release Notes

Refer to the release notes for the major releases of the product.

- [Release Notes for Version 4.1.0](#)
- [Release Notes for Version 4.0.0](#)
- [Release Notes for Version 3.2.0](#)
- [Release Notes for Version 3.1.0](#)
- [Release Notes for Version 3.0.0](#)
- [Release Notes for Version 2.2.0](#)
- [Release Notes for Version 2.1.0](#)
- [Release Notes for Version 2.0.0](#)
- [Release Notes for Version 1.5.0.4](#)
- [Release Notes for Version 1.5.0.3](#)
- [Release Notes for Version 1.5.0.1](#)
- [Release Notes for Version 1.4.0](#)

For details about latest hotfixes, see the [nuget page](#).



**Note :** GcExcel .NET currently does not provide support for Smart Art Graphics, exporting Excel files to XPS and SVG files, importing XLS files and the Form Controls.

## Release Notes for Version 4.1.0

### Enhancements from the Previous Release

The following features have been added with this version of the product:

- Parse formula string into a syntax tree.
- Ignore Formulas when saving Excel files.
- Support open action script on PdfSaveOptions.
- New overload method to load JSON.
- More Features for SpreadJS Integration: RangeTemplate cell type, get/set custom object as cell value.
- New ToJson and FromJSON methods to Workbook elements.

### Performance Enhancements

The following performance enhancements have been done in this version of the product:

- Excel Template processing performance has been improved.
- Calculation Engine's performance has been improved while setting values.

### Resolved Issues

The following issues have been resolved since the last release.

- Performance issue when updating data in Excel using GcExcel API.
- When the worksheet contains a shape, the row height cannot be changed.
- The formula =TEXT("12345","[dbnum2]") does not work.
- The font is bold after exporting to PDF.
- ROUNDDUP result is different with Excel.
- GcExcel formula result is different with Excel.
- COUNTIF result is different with Excel.
- After saving to Excel, the hidden rows are displayed.
- Conditional Format is not exported in PDF/HTML.
- The position of radio button list is wrong in the exported PDF.
- After setting the cell style in two ways, the results are different.
- Using IRange.HasFormula and IRange.Value together, degrades performance.
- Mixed order of Set and Get cell values degrades performance.

## Release Notes for Version 4.0.0

### Enhancements from the Previous Release

The following features have been added with this version of the product:

- Support for new PDF Form custom input types in Excel Templates with advanced input and validation settings.
- Support for adding, modifying and deleting Pivot Charts in Excel documents.
- Support for iterative calculations in Excel documents.
- Support for adding Barcodes while exporting to PDF, HTML or Image file formats.
- Support for cross-workbook formulas.
- Support setting default value for template cell.
- Support for getting range address to get cell's address.
- Add page printing events to track progress of Excel to PDF conversion.
- Support for selecting multiple worksheets.
- Support for getting special cells in a range.
- Disable auto grouping for date/times in PivotTable.
- Add more features for GrapeCity SpreadJS integration: cell buttons, radio and checkbox list cell type, etc.

### Resolved Issues

The following issues have been resolved since the last release.

- PivotTable MergeLabel's merged area is incorrect.
- PivotTable.DataBodyRange throws exception.
- Cannot open xlsx when the pivot source contains null values.
- Failed to export an Excel with pivot table.
- Program does not end and CPU utilisation is 100% while exporting to PDF.
- Broken Excel file is generated when copying a sheet.
- ArrayIndexOutOfBoundsException when generating JSON.
- When margins are set to the same value, the rendering position is not the same in PDF.
- Labels does not merge in exported xlsx file.
- SUM is calculated incorrectly when using GcExcel template functionality.

- PDF form is not displayed when 'printed' and 'hidden' settings are false in form field.
- Rows do not repeat while using GcExcel Template.
- Null exception is thrown during loading ssjson.
- Null exception is thrown when calling Workbook.Calculate.
- The text in exported image is wrong.
- The pivot table label does not merge in exported xlsx file.

## Release Notes for Version 3.2.0

### Enhancements from the Previous Release

The following features has been added with this version of the product:

- Support for generating PDF Form from Excel Templates.
- Support using sparklines and tables in Excel Templates.
- Support defining Fixed layout for Excel report and fill data in specific range.
- Support exporting workbook/worksheet/range to HTML.
- Support Digital Signatures API: add and sign signature lines, add and sign non-visible signatures, verify signatures, etc.
- Pivot Table enhancements: create multiple files from one Pivot Field, Defer updating, Sorting, Field layout settings, etc.
- Support adjustment of shape z-order.
- Support image quality when exporting to PDF.
- Support Picture Transparency when adding Images to Excel.
- Support more SpreadJS features: show or hide horizontal and vertical grid lines, freeze trailing rows/columns, etc.

### Resolved Issues

The following issues have been resolved since the last release.

- Object reference error on converting the Workbook to JSON with DataValidation definitions.
- ToJSON method throws error when cell has formatter on Linux.
- FromJSON method takes long time to import ssjson file using GcExcel.
- Some content displays incompletely in exported PDF.

## Release Notes for Version 3.1.0

### Enhancements from the Previous Release

The following features has been added with this version of the product:

- Support for charts, images and conditional formatting in Templates.
- Support exporting formulas in Templates
- Support global settings in Templates.
- Support converting Excel objects(chart or shape) to image formats.
- Support password protected workbook and worksheet.
- Support adding Error Bars in Chart.
- Support text angle of chart title, axis tick label and data label.

- Support alignment of Shape's TextFrame.
- Add image to specific range.
- Support Gradient Fill Type enum in Shapes.
- Support creating chart/shape/pictures with a custom name.
- Enhanced Background image support for printing to PDF.
- Get pagination info for printing to PDF.
- Support Transparent Cell Background color in PDF.
- Support worksheet JSON I/O.
- Support Outline column to display hierarchical data in saved PDF.
- Support data binding of Range, Table and Worksheet.
- Return errors from JSON Import in GcExcel.

## Resolved Issues

The following issues have been resolved since the last release.

- Filtered data cannot be re-displayed after JSON(made by SJS) I/O in GcExcel.
- Exception occurs on loading specific ssjson.
- Exception may occur on exporting certain Excel sheets with charts to PDF
- Hiding fixed columns and rows causes incorrect display in Excel file.
- Pagination is inconsistent with SpreadJS when the form is exported to PDF.
- JSON file size is bigger when converted using GcExcel vs Online designer tool.
- The Value property value of the ComboBox cell is lost after JSON I/O.
- NullPointerException may occur on loading certain Excel file and saving it.
- Conditional formatting is lost if the rule references another sheet.

## Release Notes for Version 3.0.0

### Enhancements from the Previous Release

The following features has been added with this version of the product:

- The support for templates have been added to generate Excel reports.
- The support for converting Excel spreadsheets having Charts to PDF documents have been added to the package.
- The support for converting Excel spreadsheets having Slicers to PDF documents have been added to the package.
- The support for New Excel 2016 Chart types have been added to the package.
- The support for Security options while saving to PDF have been added to the package.
- The support for document properties while saving to PDF have been added to the package.
- The API has been enhanced to support Protect Workbook features.
- The support for Chart Sheet option has been added.
- The Support for shape with hyperlink has been added.
- The Support for Group/Ungroup shapes have been added.
- Now, users can calculate Outline Subtotal.
- Now, users can get the Precedents and Dependents of formula cell.
- Now, the Pivot Table's Grand Totals and Report Layout options are similar to MExcel.
- The support for Shape Adjustment has been provided.
- The support for sheet background image to PDF has been provided.
- Now, user can export Excel files with multiple images to PDF with reduced file size.
- The support for License Workbook instance has been added.



- Now, user can rename Pivot fields and Data Fields.
- The support for Cell tags of GrapeCity SpreadJS has been added.
- The support for Cell types of GrapeCity SpreadJS has been added.
- The support for Best fit rows/columns feature of GrapeCity SpreadJS has been added.

## Resolved Issues

The following issues have been resolved since the last release.

- The `NullReferenceException` no more occurs on using `Workbook.ToJson()` and `Workbook.Save()` methods.
- Now, user can set Icon for `IconCriteria`.
- Now, the `_xlfn` prefix is not added before IFNA formula while converting to JSON.
- User can export to PDF with a specified culture
- Fixed the issue where the precision of calculated result was incorrect.
- Row/Col Header and every cells are retained after JSON(made by SJS) I/O in GcExcel
- Row/Cols with empty date are retained after JSON(made by SJS) I/O in GcExcel
- Cell types are retained after JSON(made by SJS) I/O in GcExcel
- No longer messy code while debugging GcExcel code

## Release Notes for Version 2.2.0

### Enhancements from the Previous Release

The following features have been added with this version of the product.

- Excel files with [shapes can be exported to PDF](#).
- [Ranges between different workbooks](#) can be copied.
- Worksheet between different workbooks [can be copied or moved](#).
- [Adjusting page breaks](#) after inserting or deleting rows or columns can be controlled.
- [The row, column or cell delimiter](#) can be customized while loading or saving a CSV file.
- [The tail repeated rows](#) and [right repeated columns](#) can be set when saving to PDF.
- [Paste options are supported](#) during copying and pasting ranges.
- [IRange.Find\(\)](#) and [IRange.Replace\(\) methods](#) are supported.
- [Different kinds of pivot table styles](#) can be shown or hidden.
- [Pivot table styles](#) can be exported to PDF.
- [The number format](#) setting for each pivot field is supported.
- [Japanese ruby](#) characters can be preserved after executing the Excel I/O.
- [Users can get and customize each page setting](#) before saving to a PDF file.
- Any sheet range can be [rendered inside a PDF file](#).
- Rows or columns can be [kept together](#) when saving to PDF.
- Multiple workbooks can be [saved to one PDF file](#).
- [Specific pages](#) from spreadsheet can be exported to PDF.
- [Multiple spreadsheet pages](#) can be saved into one PDF page.
- [IRange.AutoFit method](#) to fit rows or columns is supported.
- [IRange.FormulaArrayR1C1](#) property to get or set array formula in R1C1 format is supported.
- [More import flags are supported](#) while opening an Excel file.
- [OLE Objects](#) will be preserved after Excel I/O.
- [Shrink to fit feature for wrapped text](#) is supported while saving to a PDF file.

## Resolved Issues

The following issues have been resolved since the last release.

- GcExcel .NET no longer ignores the 'ignore\_empty' parameter in TEXTJOIN formula.
- Fixed the issue of large JSON file generation when using ToJson() method on a particular Workbook.
- UsedRange.Value now sets proper values to the range when the Formula is set to Empty.

## Release Notes for Version 2.1.0

### Enhancements from the Previous Release

The following features has been added with this version of the product.

- The support for .NET Framework 4.6.1 Target Framework has now been added to the package.
- Users can now [import and export spreadsheets that contain macros](#). While these will not be executed, the macros will now be preserved when saving.
- The support for [loading and saving GrapeCity SpreadJS JSON files with shapes](#) have been added.
- Users can now [set rich text format in the cells](#) by applying different styles to the textual information entered in the cell.
- While working with custom named styles, users can now [modify an existing style and add it to the Styles collection](#).
- Users can now [export Excel files with vertical text to PDF](#).
- Now, users can [insert any background image to the worksheet](#) including their organization logo, custom watermark or a wallpaper of their choice without any issues.
- The pivot table has been enhanced in order to support the date field group in Excel 2016.
- Some overloads have been added for Open and Save methods to avoid passing file format.

## Resolved Issues

The following issues have been resolved since the last release.

- The **Workbook.Calculate()** method now evaluates the cell values correctly.
- While saving an Excel file to open XML format, the logical value of the cell is now calculated without any errors.
- After configuring the **Workbook.FontsFolderPath** property, the text in the file completely renders to PDF without any issues.
- Loading SSJSON file with null values no longer throws an exception.
- While saving an Excel file to PDF, the merged range in a table now renders appropriately without any issues.
- Loading the SSJSON file now renders hidden rows correctly while saving an Excel file to PDF.

## Release Notes for Version 2.0.0

### Changes from the Previous Release

This version of the product has the following change:

- The default value of **AutoParse** has been changed to boolean false in order to enhance the performance while setting values to a range.

## Breaking Change

This version of the product has the following breaking change:

- GcExcel .NET 2.0.0 version comes with an evaluation license key that allows users to use the product without any limitations for a time period of 30 days. However, this version will not work with the license keys of older versions. This is a FREE upgrade for existing customers who already have licensed version of GcExcel. For more information on upgrading to the new version, refer [Upgrade to Latest Version](#).

## Release Notes for Version 1.5.0.4

### Enhancements from the Previous Release

The following features have been added with this version of the product.

- Custom functions are now supported. The **CustomFunction** class has been introduced in order to allow users to perform custom arithmetic logic.
- GcExcel .NET now uses **System.Drawing.Color** instead of GrapeCity.Documents.Excel.Color in order to allow users to set standard colors.
- The Calculation engine now works on **iOS**.

### Resolved Issues

The following issues have been resolved since the last release.

- Calculation results are now displayed correctly when a formula applied on a cell references an external workbook that was deleted.
- Data validation rule is now retained when saving an Excel file with a source from a different sheet's range.

## Release Notes for Version 1.5.0.3

### Enhancements from the Previous Release

The following features have been added with this version of the product.

- The **StandardWidthInPixel** property has been introduced in order to allow users to get or set the standard width(in pixel) of all the columns in the worksheet.
- The **StandardHeightInPixel** property has been introduced in order to allow users to get or set the standard height(in pixel) of all the rows in the worksheet.
- While setting borders for a range, users also have the option to reset the range of the adjacent border using the **ResetAdjacentRangeBorder** property.
- Now, you can use the **CellInfo** class with some helper functions in order to convert the row/column/cell index to expression and vice a versa.

### Changes from the Previous Release

This version of the product has the following changes:

- Now, you can get or set the single cell values in a spreadsheet considerably faster than before.
- The performance of setting an array of double/int/float values to a range has been significantly improved. For instance, `Range.Value = new double[,]{ {1d, 2d}, {3d, 4d}}`.
- You can get or set the style for a single cell in a worksheet quickly and efficiently.

## Resolved Issues

The following issues have been resolved since the last release.

- PDF is now saved correctly when the width of the column exceeds the width of the paper.

## Release Notes for Version 1.5.0.1

### Enhancements from the Previous Release

The following features have been added with this version of the product.

- Export PDF operations are extensively supported in this version. You can use the `Workbook.Save` method with `SaveFileFormat.pdf` parameter in order to save spreadsheets to pdf files and then you can view the pdf in any pdf viewer, such as browser.
- Cut, Copy and Paste operations are now supported in Shape, Chart, Slicer and Picture.
- Enhanced workbook and worksheet views in terms of display (zoom, horizontal and vertical scrollbar, tabs, gridline color, outlines, whitespace, zeros and a lot more) . Also, you can split a worksheet into panes.
- Support for more built-in themes.
- Some open and save enhancements have been introduced. Now, you can configure open and save settings while opening a csv file or stream and saving a csv file or stream respectively.

## Release Notes for Version 1.4.0

### Enhancements from the Previous Release

The following features have been added with this version of the product.

- Excel PageSetup options are now supported to manage printing.
- SpreadJS v11 SSJSON (chart ssjson and data validation ssjson) is now supported.
- Open options are now supported while opening Excel.
- The `RefersToR1C1` property in the `IName` interface is now supported.
- All the GcExcel .NET features except calc engine are supported on Xamarin.iOS. Users simply need to disable calc engine as shown below:  
`Workbook.EnableCalculation = false;`

## Resolved Issues

The following issues have been resolved since the last release.

- SSJSON now loads without losing custom named styles.
- RGBA field can be read in SSJSON without throwing exceptions.
- The Value of `Range.Rows[i].Hidden` is now accurate.

- Cell value is now changed after exporting an Excel file when the value contains "\r".

## Index

**Access a Range, 30-31**  
**Access Areas in a Range, 31**  
**Access Cells, Rows and Columns in a Range, 36-37**  
**Add and Delete Table Columns and Rows, 290-292**  
**Add Slicer in Pivot Table, 316-318**  
**Add Slicer in Table, 315-316**  
**Add Validations, 106-109**  
**Adjust Column Width and Row Height, 432**  
**API Reference, 476**  
**Area Chart, 254-256**  
**Auto Fit Row Height and Column Width, 49-51**  
**Average Rule, 102**  
**Axis and Other Lines, 243-245**  
**Background Image, 181-182**  
**Bar Chart, 256-258**  
**Barcodes, 191**  
**Box Whisker, 275-276**  
**Cell Context, 352-356**  
**Cell Expansion, 352**  
**Cell Types, 59-62**  
**Cell Value Rule, 101**  
**Chart, 218-219**  
**Chart Area, 221-222**  
**Chart Sheet, 284-286**  
**Chart Title, 220-221**  
**Chart Types, 252-254**  
**Charts, 384-389**  
**Codabar, 198-201**  
**Code128, 206-208**  
**Code39, 201-203**  
**Code49, 210-212**  
**Code93, 203-205**  
**Color Scale Rule, 102-103**  
**Column Chart, 258-260**

- Combo Chart, 260-262**
- Comments, 80-83**
- Conditional Formatting, 100-101 , 356-358**
- Configure Chart, 220**
- Configure Chart Axis, 245-248**
- Configure Chart Series, 228-238**
- Configure Columns to Repeat at Left and Right, 327-328**
- Configure Fonts and Set Style, 407-408**
- Configure Page Breaks, 324-325**
- Configure Page Header and Footer, 321-322**
- Configure Page Settings, 322-324**
- Configure Paper Settings, 325-326**
- Configure Paper Source, 329-330**
- Configure Print Area, 326-327**
- Configure Rows to Repeat at Top and Bottom, 328**
- Configure Sheet Print Settings, 328-329**
- Control Image Quality, 442-443**
- Control Pagination, 413-414**
- Control Position of Overlapping Shapes, 183-184**
- Convert to Image, 471-475**
- Create and Delete Chart, 219-220**
- Create and Delete Tables, 287**
- Create and Set Custom Named Style, 189-191**
- Create Excel Report using Template, 397-402**
- Create Pivot Table, 295-296**
- Create Row or Column Group, 91-92**
- Create Workbook, 74-75**
- Cross Workbook Formula, 158-159**
- Custom Form Input Types, 379-384**
- Custom Functions, 159-168**
- Customize Chart Objects, 223-224**
- Customize Shape Format and Shape Text, 171-176**
- Customize Worksheets, 55-57**
- Cut or Copy Across Sheets, 78-79**
- Cut or Copy Cell Ranges, 38-41**
- Cut or Copy Shape, Slicer, Chart and Picture, 41-42**

- Data Bar Rule, 103**
- Data Binding, 110-114**
- Data Label, 249-251**
- Data Matrix, 214-217**
- Data Source Binding, 393-397**
- Data Validations, 106**
- Date Occurring Rule, 101-102**
- Default Values in Template Cells, 365-366**
- Delete Blank Pages From Middle, 419-420**
- Delete Validation, 109**
- Digital Signatures, 114-124**
- EAN-13, 194-196**
- EAN-8, 196-198**
- Enable or Disable Calculation Engine, 79-80**
- End User License Agreement, 22**
- Error Bars, 238-243**
- Export Barcodes, 438-439**
- Export Charts, 432-437**
- Export Custom Page Information, 422-423**
- Export Different Headers On Different Pages, 420-421**
- Export Last Page Without Headers, 421-422**
- Export Multiple Sheets To One Page, 417-418**
- Export Pivot Table Styles And Format, 408-410**
- Export Shapes, 410-411**
- Export Signature Lines, 439-440**
- Export Slicers, 437-438**
- Export Specific Pages To PDF, 423-424**
- Export to HTML, 445-449**
- Export to PDF, 405-407**
- Export Vertical Text, 411-412**
- Export Worksheet to PDF, 426-428**
- Expression Rule, 105-106**
- Features, 23-24**
- File Operations, 403**
- Filter, 89-91**
- Find and Replace Data, 42-44**



- Fixed Layout, 362-365**
- Floor, 248-249**
- Formula Functions, 131-149**
- Formula Parser, 125-131**
- Formulas, 124-125**
- Freeze Panes in a Worksheet, 53-54**
- Freeze Trailing Panes in a Worksheet, 54-55**
- Funnel, 283-284**
- GcExcel .NET Overview, 9**
- Get Address of Cell Range, 37-38**
- Get Row and Column Count, 44-45**
- Get Special Cell Ranges, 31-36**
- Getting Started, 12-15**
- Global Settings, 358-361**
- Group, 91**
- Group or Ungroup Shapes, 178-180**
- GS1-128, 208-210**
- Hide Rows and Columns, 45**
- Histogram, 276-277**
- Hyperlink on Shape, 176-178**
- Hyperlinks, 83-85**
- Icon Sets Rule, 105**
- Image Transparency, 183**
- Import and Export .xlsx Document, 403-405**
- Import and Export CSV File, 449-451**
- Import and Export CSV File with Delimiters, 451-453**
- Import and Export from JSON string, 456**
- Import and Export JSON Stream, 453-456**
- Import and Export Macros, 469-470**
- Import and Export OLE Objects, 470-471**
- Import and Export SpreadJS Files, 456-468**
- Insert And Delete Cell Ranges, 45-47**
- Insert and Delete Rows and Columns, 47-48**
- Iterative Calculation, 157-158**
- Keep Rows Together Over Page Breaks, 418-419**
- Key Features, 10-11**

- Legends, 251-252**
- License Information, 18-21**
- Line Chart, 262-264**
- Logging, 330-336**
- Measure Digital Width, 52-53**
- Merge Cells, 48**
- Modify Slicer with Custom Style, 320**
- Modify Table Layout, 293-294**
- Modify Table Layout for Slicer Style, 320-321**
- Modify Table with Custom Style, 292-293**
- Modify Tables, 287-289**
- Modify Validation, 109-110**
- Open and Save Workbook, 75-76**
- Outline Column, 96-100**
- Outline Subtotals, 94-96**
- Pareto Chart, 279-280**
- PDF Form Builder, 366-379**
- PDF417, 212-214**
- Pie Chart, 264-266**
- Pivot Chart, 307-310**
- Pivot Table, 294-295**
- Pivot Table Settings, 296-302**
- Pivot Table Style, 302-307**
- Plot Area, 222-223**
- Precedents and Dependents, 154-157**
- Print Settings, 321**
- Protect Workbook, 76-78**
- QRCode, 191-194**
- Quick Start, 15-18**
- Quote Prefix, 66-67**
- Radar Chart, 273-274**
- Range Operations, 29-30**
- Range Template Cell, 62-66**
- Redistribution, 22**
- Release Notes, 477**
- Release Notes for Version 1.4.0, 484-485**

**Release Notes for Version 1.5.0.1, 484**  
**Release Notes for Version 1.5.0.3, 483-484**  
**Release Notes for Version 1.5.0.4, 483**  
**Release Notes for Version 2.0.0, 482-483**  
**Release Notes for Version 2.1.0, 482**  
**Release Notes for Version 2.2.0, 481-482**  
**Release Notes for Version 3.0.0, 480-481**  
**Release Notes for Version 3.1.0, 479-480**  
**Release Notes for Version 3.2.0, 479**  
**Release Notes for Version 4.0.0, 478-479**  
**Release Notes for Version 4.1.0, 477-478**  
**Remove a Group, 92-93**  
**Render Excel Range Inside PDF, 414-417**  
**Rich Text, 70-74**  
**Save Multiple Workbooks to Single PDF, 424-426**  
**Series, 224-228**  
**Set Array Formula, 153-154**  
**Set Formula to Range, 149-151**  
**Set Row Height and Column Width, 49**  
**Set Sheet Styling, 185-189**  
**Set Table Formula, 151-153**  
**Set Values to a Range, 48-49**  
**Shape Adjustment, 180-181**  
**Shapes and Pictures, 168-171**  
**Shrink To Fit With Text Wrap, 412-413**  
**Size and Position of Image, 182-183**  
**Slicer, 315**  
**Slicer Style, 319-320**  
**Sort, 85-88**  
**Sparkline, 310-315**  
**Sparklines, 391-393**  
**Specialized Chart, 280**  
**Statistical Chart, 274-275**  
**Stock Chart, 266-269**  
**Styles, 184-185**  
**Summary Row, 93-94**

- Sunburst, 280-281
- Support Background Color Transparency, 442
- Support Document Properties, 431-432
- Support for SpreadJS Features, 468-469
- Support Security Options, 429-431
- Support Sheet Background Image, 440-441
- Surface Chart, 269-270
- Table, 286-287
- Table Filters, 289-290
- Table Sort, 289
- Table Style, 292
- Tables, 389-391
- Tags, 67-70
- Technical Support, 22
- Template Configuration, 340-342
- Template Fields, 342-345
- Template Properties, 345-352
- Templates, 337-340
- Theme, 217-218
- Top Bottom Rule, 103-104
- Track Export Progress, 443-445
- TreeMap, 281-283
- Unique Rule, 104-105
- Upgrade to Latest Version, 21-22
- Use Do Filter Operation, 318-319
- Walls, 243
- Waterfall Chart, 277-279
- Work with Used Range, 51-52
- Work with Worksheets, 24-29
- Workbook, 74
- Workbook Views, 80
- Working With Page Setup, 428-429
- Worksheet, 24
- Worksheet Views, 57-59
- XY (Scatter) Chart, 270-273