Table of Contents

Table of Contents 1-10
Document Solutions for Excel, .NET Edition Overview 11
Key Features 12-13
Getting Started 14-17
Quick Start 17-20
License Information 20-23
Upgrade to Latest Version 23-24
Technical Support 24
Contacting Sales 24
Redistribution 24-25
End User License Agreement 25
Features 26-27
Worksheet 27
Work with Worksheets 27-34
Range Operations 35
Access a Range 35-36
Get Intersection, Union and Offset Range 36-38
Access Areas in a Range 38
Get Special Cell Ranges 38-43
Access Cells, Rows and Columns in a Range 43-44
Get Address of Cell Range 44-45
Cut or Copy Cell Ranges 45-46
Cut or Copy Shape, Slicer, Chart and Picture 46-48
Paste or Ignore Data in Hidden Range 48-49
Find and Replace Data 49-51
Get Row and Column Count 51-52
Hide Rows and Columns 52
Insert And Delete Cell Ranges 52-54
Insert and Delete Rows and Columns 54-55
Merge Cells 55
Set Values to a Range 55-56
Set Custom Objects to a Range 56-58

- Set Row Height and Column Width 58-59
- Auto Fit Row Height and Column Width 59-60
- Work with Used Range 60-62
- Measure Digital Width 62-63
- Set Default Values for Cell Range 63-64
- Set Cell Background Image for Cell Range 64-66
- Ignore Errors in Cell Range 66-67
- Freeze Panes in a Worksheet 67-69
- Freeze Trailing Panes in a Worksheet 69-70
- Customize Worksheets 70-71
- Worksheet Views 71-74
- Cell Types 74-76
  - Range Template Cell 76-81
- Quote Prefix 81
- Tags 81-84
- Rich Text 84-88

Workbook 88-89

- Create Workbook 89
- Open and Save Workbook 89-92
- Protect Workbook 92-95
- Cut or Copy Across Sheets 95
- Enable or Disable Calculation Engine 95-96
- Workbook Views 96-98

Comments 98-101

- Threaded Comments 101-104

Hyperlinks 104-106

Sort 106-109

Filter 109-112

Group 112-113

- Create Row or Column Group 113-114
- Remove a Group 114-115
- Summary Row 115
- Outline Subtotals 115-117

Outline Column 117-122
Row or Column Group Information 122-125
Conditional Formatting 125
Cell Value Rule 125-126
Date Occurring Rule 126
Average Rule 126-127
Color Scale Rule 127
Data Bar Rule 127-128
Top Bottom Rule 128-129
Unique Rule 129
Icon Sets Rule 129-130
Expression Rule 130
Data Validations 130-131
Add Validations 131-134
Delete Validation 134
Modify Validation 134-135
Data Binding 135-139
Digital Signatures 139-150
Formulas 150-151
Formula Parser 151-158
Formula Functions 158-178
Set Formula to Range 178-179
Set Table Formula 179-182
Set Array Formula 182
Dynamic Array Formulas 182-185
Precedents and Dependents 185-189
Iterative Calculation 189-190
Calculation Mode 190-191
Cross Workbook Formula 191-193
Localized Formulas 193-195
Custom Functions 195-207
Shapes and Pictures 207-211
Customize Shape Format and Shape Text 211-222

- Hyperlink on Shape 222-224
- Group or Ungroup Shapes 224-226
- Shape Adjustment 226-227
- Background Image 227-228
- Size and Position of Image 228
- Image Transparency 228-229
- Control Position of Overlapping Shapes 229-230
- Linked Picture 230-231

Document Properties 231-232

Styles 232-233

- Set Sheet Styling 233-237
- Create and Set Custom Named Style 237-239

Form Controls 239-247

Barcodes 247-249

- QRCode 249-252
- EAN-13 252-254
- EAN-8 254-256
- Codabar 256-259
- Code39 259-261
- Code93 261-263
- Code128 264-266
- GS1-128 266-268
- Code49 268-270
- PDF417 270-272
- Data Matrix 272-275

Theme 275-276

Chart 276-277

- Create and Delete Chart 277-278
- Configure Chart 278
  - Chart Title 278-280
  - Chart Area 280-281
  - Plot Area 281-282
- Customize Chart Objects 282

Series 282-286
Configure Chart Series 286-299
Error Bars 299-305
Display Empty Cells 305
Display #N/A Values 305-306
Walls 306-307
Axis and Other Lines 307-310
Configure Chart Axis 310-313
Floor 313-314
Data Label 314-316
Legends 316-317
Data Table 318-319
Chart Types 319-321
Area Chart 321-323
Bar Chart 323-325
Column Chart 325-327
Combo Chart 327-329
Line Chart 329-331
Pie Chart 331-333
Stock Chart 333-335
Surface Chart 335-337
XY (Scatter) Chart 337-339
Radar Chart 339-341
Statistical Chart 341-342
Box Whisker 342-343
Histogram 343-344
Waterfall Chart 344-345
Pareto Chart 345-346
Specialized Chart 346-347
Sunburst 347-348
TreeMap 348-349
Funnel 349-351
Chart Sheet 351-353

Table 353
Create and Delete Tables 353-354
Convert Table to Range 354
Modify Tables 354-356
Table Sort 356-357
Table Filters 357
Add and Delete Table Columns and Rows 357-359
Table Style 359
Modify Table with Custom Style 359-361
Modify Table Layout 361-362
Pivot Table 362
Create Pivot Table 362-363
Pivot Table Settings 363-372
Pivot Table Style 372-377
Pivot Chart 377-380
Sparkline 380-385
Slicer 385
Add Slicer in Table 385-386
Add Slicer in Pivot Table 386-388
Slicer Style 388-389
Modify Slicer with Custom Style 389
Modify Table Layout for Slicer Style 389-390
Auto-Filter Table with Slicer 390-391
Configure Slicer Layout 391-392
Cut or Copy Slicer 392-394
Duplicate Slicer 394-395
Use Do Filter Operation 395-396
Print 396-397
Page Setup 397
Configure Page Header and Footer 397-399
Configure Page Settings 399-400
Configure Page Breaks 400-402
Configure Paper Settings 402-403

Configure Print Area 403
Configure Columns to Repeat at Left and Right 403-404
Configure Rows to Repeat at Top and Bottom 404-405
Configure Sheet Print Settings 405
Configure Paper Source 405-406
Logging 406-412
Defined Names 412-415
Templates 416-419
Template Configuration 419-421
Template Fields 421-424
Template Properties 424-437
Cell Expansion 437-438
Cell Context 438-441
Conditional Formatting 441-443
Global Settings 443-449
Fixed Layout 449-452
Default Values in Template Cells 452-453
PDF Form Builder 453-466
Custom Form Input Types 466-471
Charts 471-476
Tables 476-478
Sparklines 478-480
Paginated Templates 480-481
Pagination Properties and Functions 481-494
Data Source Binding 494-503
Create Excel Report using Template 503-508
File Operations 509
Import and Export .xlsx Document 509-511
Export to PDF 511-512
Configure Fonts and Set Style 512-514
Export Pivot Table Styles And Format 514-516
Export Shapes 516-517
Export Vertical Text 517-518

Shrink To Fit With Text Wrap 518-519
Control Pagination 519-520
Render Excel Range Inside PDF 520-523
Export Multiple Sheets To One Page 523-524
Keep Rows Together Over Page Breaks 524-525
Delete Blank Pages From Middle 525-526
Export Different Headers On Different Pages 526-527
Export Last Page Without Headers 527-528
Export Custom Page Information 528-529
Export Specific Pages To PDF 529-530
Save Multiple Workbooks to Single PDF 530-531
Export Worksheet to PDF 532-533
Working With Page Setup 533-535
Support Security Options 535-537
Support Document Properties 537-538
Adjust Column Width and Row Height 538-539
Export Charts 539-544
Export Slicers 544-545
Export Barcodes 545-546
Export Signature Lines 546-547
Export Form Controls to Form Fields 547-549
Support Sheet Background Image 549-551
Support Background Color Transparency 551-552
Control Image Quality 552
Track Export Progress 552-554
Export to HTML 554-559
Import and Export CSV File 559-561
Import CSV File with Custom Parser 561-562
Import and Export CSV File with Delimiters 562-564
Import and Export JSON Stream 564-567
Import and Export from JSON string 567-580
Import and Export from JSON without Worksheets 580-581
Import and Export SpreadJS Files 581

Import and Export JSON Files 581-595
SpreadJS Sparklines 595-597
Import and Export .sjs Files 597-599
Support for SpreadJS Features 599-603
Import and Export Macros 603-604
Import and Export Excel Templates 604-605
Import and Export OLE Objects 605-606
Convert to Image 606-613
Import and Export Excel Options 613-615
Document Solutions Data Viewer 616
API Reference 617
Release Notes 618
Breaking Changes 618
Release Notes for Version 7.1.0 618-619
Release Notes for Version 7.0.0 619-620
Release Notes for Version 6.2.0 620
Release Notes for Version 6.1.0 621
Release Notes for Version 6.0.0 621-622
Release Notes for Version 5.2.0 622
Release Notes for Version 5.1.0 622-623
Release Notes for Version 5.0.3 623-624
Release Notes for Version 5.0.0 624
Release Notes for Version 4.2.0 624-625
Release Notes for Version 4.1.0 625-626
Release Notes for Version 4.0.0 626-627
Release Notes for Version 3.2.0 627
Release Notes for Version 3.1.0 627-628
Release Notes for Version 3.0.0 628-629
Release Notes for Version 2.2.0 629-630
Release Notes for Version 2.1.0 630-631
Release Notes for Version 2.0.0 631
Release Notes for Version 1.5.0.4 631
Release Notes for Version 1.5.0.3 631-632

Release Notes for Version 1.5.0.1 632
Release Notes for Version 1.4.0 632-633
Index 634-642

Document Solutions for Excel, .NET Edition Overview

Document Solutions for Excel, .NET Edition (DsExcel .NET, previously GcExcel .NET) 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.

DsExcel .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 DsExcel .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 DsExcel .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 DsExcel .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 DsExcel .NET; developers can quickly build everything right from the scratch to organize and structure business-critical data for maximum productivity and enhanced analysis.

DsExcel.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**
  DsExcel .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, DsExcel .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**
  DsExcel .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 DsExcel .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**
  DsExcel .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 DsExcel .NET, you can apply cloud based deployments and deploy your applications on Azure and AWS Lambda.

- **Shapes and Pictures**
  With DsExcel 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**
  DsExcel provides templates with comprehensive API to create custom Excel reports with advanced layouts. You can use multiple data soures 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 DsExcel .NET, refer to the Features topic in the documentation.

Getting Started

System Requirements

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

- .NET 5, .NET 6, and .NET 7
- .NET Core 2.x and 3.x
- .NET Standard 2.x
- .NET Framework 4.6.1 or higher

Dependencies

The following table lists the open-source library that is used in DsExcel .NET:

Name	Version	Description and Usage
Newtonsoft.Json	13.0.1	Reads and writes JSON. It also Implements JSON DOM data binding.

Setting up an Application

DsExcel .NET reference is available through NuGet, a Visual Studio extension that automatically adds libraries and references to your project. To work with DsExcel .NET, you need to have following references in your application:

Reference	Purpose
DS.Documents.Excel	To use DsExcel in an application, you need to reference (install) just the DS.Documents.Excel package. It pulls in the required infrastructure packages.

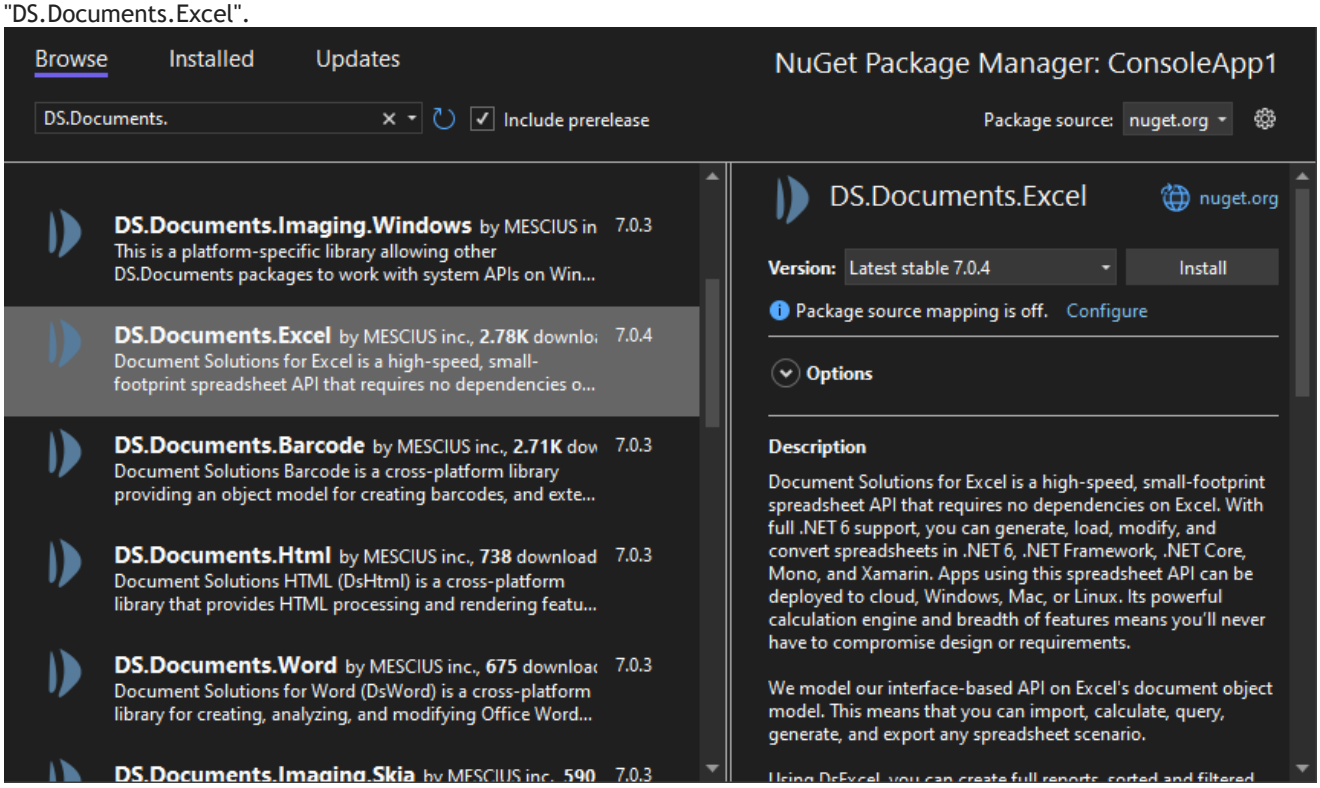
Add reference to DsExcel .NET in your application

In order to use DsExcel .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:

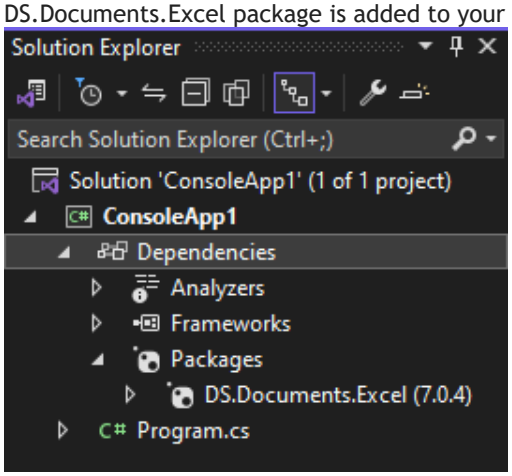
VISUAL STUDIO FOR WINDOWS

To find and install the DS.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 "ds.documents" or "DS.Documents" in the search text box at the top and find the package "DS.Documents.Excel".



4. Click **Install** to install the **DS.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 DS.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 DsExcel from the

website and put the DS.Documents.Excel nuget package 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 DS.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 DsExcel .NET NuGet package using the following command:
   dotnet add package DS.Documents.Excel

To add DS.Documents.Excel package reference

DsExcel .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>net6.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="DS.Documents.Excel" Version="7.0.0" />
  </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 "DS.Documents".
5. From the list of packages displayed in the left panel, select **DS.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 "**Ds**" and press Enter.
   Observe: Document Solutions packages get displayed in the dropdown.
10. Choose **DS.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 DsExcel 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 DsExcel .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 ds.documents', select **DS.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 DsExcel .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 ds.documents', select **DS.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: DsExcelController) and click **Add.**
4. Add the DsExcel reference in Controller file:
using GrapeCity.Documents.Excel;
5. In the Index() method of the Controller, add the following code:

DsExcelController.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: DsExcel).
3. Right click the folder DsExcel, 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"] = "Document Solutions for Excel, .NET Edition";
```

```
}
<script>
    onload = function () {
       alert("File Saved: HelloWord.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/DsExcel/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

This topic includes:

Types of Licenses
Apply License

DsExcel .NET supports the following types of license:

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

Unlicensed

When you download DsExcel 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 DsExcel .NET.

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

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

- **Maximum Operating Time**

  While executing an application program, the duration of operating DsExcel .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 Document Solutions for Excel, .NET Edition. Contact us.sales@mescius.com to get your 30-day evaluation key to remove this text and other limitations."

Evaluation License

DsExcel .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@mescius.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 expired evaluation license, the following watermark will appear:

"Expired Evaluation copy of Document Solutions for Excel, .NET Edition. Contact us.sales@mescius.com to purchase license."

Licensed

DsExcel .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 DsExcel .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 DsExcel .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:

DsExcelController.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 DsExcel.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@mescius.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 DsExcel.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 Workbook("new key");
- To upgrade the license of all the instances:
Workbook.SetLicenseKey("new key");

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

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

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

Technical Support

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

- Product Forum: https://developer.mescius.com/forums/
- Email: us.sales@mescius.com

Contacting Sales

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

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

Redistribution

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

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

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

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

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

End User License Agreement

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

Features

This section comprises the features available in DsExcel.

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.

Dynamic Array Formulas
: Work with Dynamic Array Formulas.

Styles
: Set styles to format cell appearance.

Document Properties
: Learn how to set built-in or custom document properties.

Form Controls
: Use form controls to make your worksheet interactive and to get inputs from the users.

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
: Configure print and page settings to manage printing options.

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

Defined Names
: Name the tables or ranges within workbook or worksheet scope.

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 DsExcel .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

DsExcel 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. DsExcel 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");
```

Copy and Move Multiple Worksheets

DsExcel provides **Copy**, **CopyBefore**, **CopyAfter**, **Move**, **MoveBefore**, and **MoveAfter** methods with the **IWorksheets** interface that allow you to copy or move multiple worksheets at once. You can copy or move worksheets to the end or a specific location within the same workbook or a different workbook, as described below:

Method	Description
Copy	This method copies the sheet collection to the end of the target workbook. If the target workbook is null, the sheet collection will be copied to the current workbook.
CopyBefore	This method copies the sheet collection before the specified sheet. The target worksheet can belong to any workbook.
CopyAfter	This method copies the sheet collection after the specified sheet. The target worksheet can belong to any workbook.
Move	This method moves the sheet collection to the end of the target workbook. If the target workbook is null, the sheet collection will be moved to the current workbook.
MoveBefore	This method moves the sheet collection before the specified sheet. The target worksheet can belong to any workbook.
MoveAfter	This method moves the sheet collection to the specified location after the specified sheet. The target worksheet can belong to any workbook.

Refer to the following example code to copy multiple sheets in the same workbook:

C#

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

// Open the Excel file.
workbook.Open("FlowChartsFile.xlsx");

// Copy the selected sheets to the end of the current workbook.
workbook.Worksheets[new string[] { "FlowChart1", "FlowChart2" }].Copy();

// Save the Excel file.
workbook.Save("CopyMultipleWorksheets.xlsx");
```

Refer to the following example code to copy multiple sheets to another workbook:

C#

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

// Open the Excel file.
workbook.Open("FlowChartsFile.xlsx");

// Create another Excel file.
Workbook copyWorkbook = new Workbook();

// Copy the selected sheets to the end of the target workbook.
workbook.Worksheets[new string[] { "FlowChart1", "FlowChart2" }].Copy(copyWorkbook);
```

```
// Save the Excel file.
copyWorkbook.Save("CopyMultipleWorksheets.xlsx");
```

Refer to the following example code to move multiple sheets in the same workbook:

C#

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

// Open the Excel file.
workbook.Open("FlowChartsFile.xlsx");

// Copy the selected sheets to the end of the current workbook.
workbook.Worksheets[new string[] { "FlowChart1", "FlowChart2" }].Move();

// Save the Excel file.
workbook.Save("MoveMultipleWorksheets.xlsx");
```

Refer to the following example code to move multiple sheets to another workbook:

C#

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

// Open the Excel file.
workbook.Open("FlowChartsFile.xlsx");

// Create another Excel file.
Workbook moveWorkbook = new Workbook();

// Copy the selected sheets to the end of the target workbook.
workbook.Worksheets[new string[] { "FlowChart1", "FlowChart2" }].Move(moveWorkbook);

// Save the Excel file.
moveWorkbook.Save("MoveMultipleWorksheets.xlsx");
```

Note:

- All the worksheets in the current workbook cannot be moved to another workbook; this will raise an exception because the workbook must have at least one sheet.
- When all the worksheets are moved in the current workbook, nothing will happen as the sheets are added in the same manner.
- When copying a worksheet with the same name that exists in the target workbook, the worksheet will be renamed by adding a suffix (x). x represents the index with the same name.

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
- Paste or Skip Data in Hidden Range
- 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
- Set Default Values for Cell Range
- Set Cell Background Image for Cell Range
- Ignore Errors in Cell Range

Access a Range

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

DsExcel 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();
```

Note: DsExcel also supports defined names, which can be used to name the range. For more information, see Defined Names.

Get Intersection, Union and Offset Range

DsExcel allows you to get the intersection, union and offset of specified ranges using the **IRange** interface.

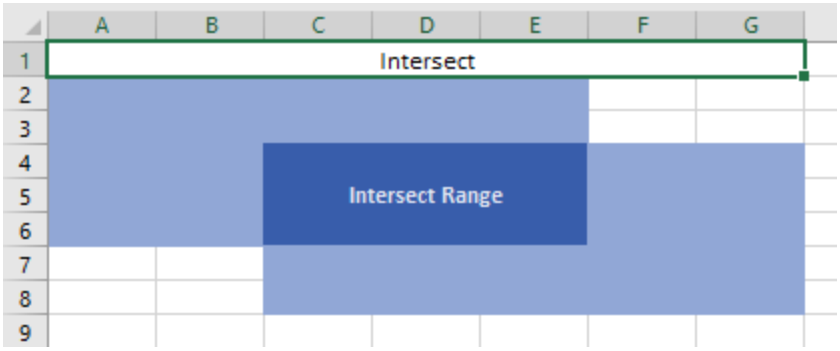
To get the intersection and union of two or more ranges, you can use **Intersect** method and **Union** method of the IRange interface respectively. Similarly, the interface also provides **Offset** method to get the offset of a specified range.

The sample code below shows how to get the intersection, union and offset of different ranges:

C#

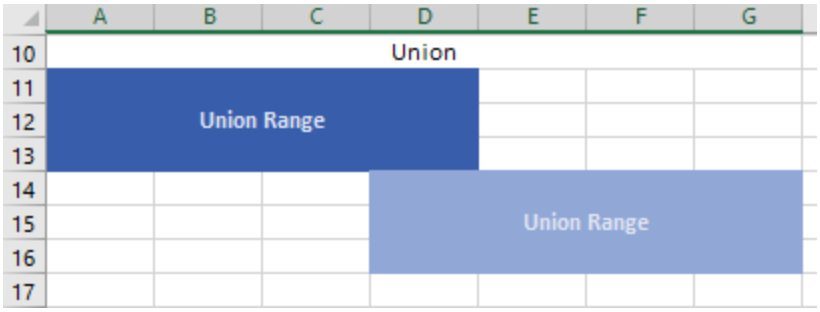
```
// Set the intersection of two range value and style.
var intersectRange = worksheet.Range["A2:E6"].Intersect(worksheet.Range["C4:G8"]);
intersectRange.Interior.Color = Color.FromArgb(56, 93, 171);
intersectRange.Merge();
intersectRange.Value = "Intersect Range";
intersectRange.Font.Bold = true;
intersectRange.Font.Color = Color.FromArgb(226, 231, 243);
```

```
intersectRange.HorizontalAlignment = HorizontalAlignment.Center;
intersectRange.VerticalAlignment = VerticalAlignment.Center;
```



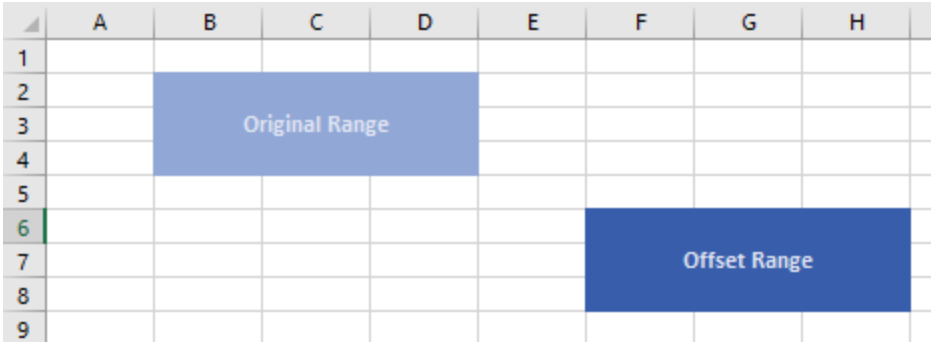
C#

```
// Set the union of two range value and font style.
var unionRange = worksheet.Range["A11:D13"].Union(worksheet.Range["D14:G16"]);
unionRange.Value = "Union Range";
unionRange.Font.Bold = true;
unionRange.Font.Color = Color.FromArgb(226, 231, 243);
```



C#

```
// Set the offset of the range value and style.
var offsetRange = worksheet.Range["B2:D4"].Offset(4, 4);
offsetRange.Merge();
offsetRange.Value = "Offset Range";
```



> **Note:** An exception is thrown, if one or more ranges from a different worksheet are specified or the offset set in Offset method is out of bounds.

To view the code in action, see Intersection and Union, and Offset Demo sample.

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.

DsExcel 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.

The following table lists the types of cells or values that SpecialCellType and SpecialCellsValue enumerations allow you to find:

Enumeration	Type	Description
SpecialCellType	AllFormatConditions	Cells of any format condition in the specified range.
	AllValidation	Cells having validation criteria in the specified range.
	Blanks	Empty cells in the specified range.
	Comments	Cells containing notes in the specified range.
	Constants	Cells containing constants. Use the **SpecialCellsValue** to filter values by data types.
	Formulas	Cells containing formulas. Use the **SpecialCellsValue** to filter formulas by return types.

	LastCell	The last visible cell in the used range of the worksheet in the specified range.
	MergedCells	Merged cells that intersect with the specified range.
	SameFormatConditions	Cells having the same format as the top-left cell of the specified range.
	SameValidation	Cells having the same validation criteria as the top-left cell of the specified range.
	Visible	All visible cells in the specified range.
	Tags	Cells containing tags in the specified range.
SpecialCellsValue	Errors	Cells with errors.
	Logical	Cells with logical values.
	Numbers	Cells with numeric values.
	TextValues	Cells with text.

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();

// Get active sheet.
IWorksheet ws = workbook.ActiveSheet;

// Add data to the range.
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", "8zgnvlkp2@163.com", true},
{ 2d, "User 2", "b9fvaswb@163.com", false},
{ 3d, "User", "md78b", false},
{ 4d, "User 4", "1qasghjfg@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["A9"].Tag = "Number type";
ws.Range["B9"].Tag = "Text type";
ws.Range["C9"].Tag = "Text type";
ws.Range["D9"].Tag = "Bool type";

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);

// Find cells containing tags.
var tagCells = searchScope.SpecialCells(SpecialCellType.Tags);

// Add output of above search to the range.
ws.Range["A16"].Value = "Find result";
ws.Range["A16:C16"].Merge();
ws.Range["A16:C16"].HorizontalAlignment = HorizontalAlignment.Center;
ws.Range["$A$17:$A$26"].Value = new object[,] {
{"Comments"},
{"LastCell"},
{"Visible"},
{"Blanks"},
{"AllFormatConditions"},
{"AllValidation"},
{"SameFormatConditions B10"},
{"SameValidation D10"},
{"MergedCells"},
{"TagCells"}
};
ws.Range["$C$17:$C$26"].Value = new object[,] {
{comments.Address},
{lastCell.Address},
{visible.Address},
{blanks.Address},
{allFormatConditions.Address},
{allValidation.Address},
{sameFormatConditions.Address},
{sameValidation.Address},
{merged.Address},
```

```
{tagCells.Address},
};

ws.UsedRange.EntireColumn.AutoFit();

// Save the 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#

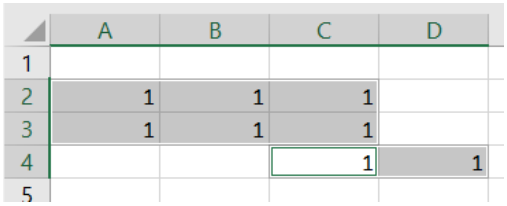
```
//create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();

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)0xFFF2F2F2);
    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");
```

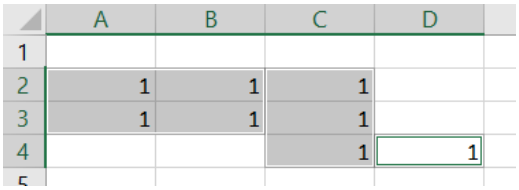
Limitations

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

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



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



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 DsExcel, 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 DsExcel API members to retrieve the address of cell[0,0] in different notations and references.

Cell Reference Notation	Absolute Reference	Relative Reference
A1	Address property *Output: $A$1*	GetAddress method (set rowAbsolute and columnAbsolute parameters to False) *Output: A1*

R1C1	GetAddress method (set referenceStyle parameter to R1C1) *Output: R1C1*	GetAddress method (set referenceStyle parameter to R1C1, rowAbsolute and columnAbsolute parameters to False) *Output: RC*

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

DsExcel .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. You can also choose whether to copy and paste the data in a hidden range in a worksheet. To know more, refer to Paste or Skip Data in Invisible Range.

In order to cut or copy data across multiple sheets, refer to Cut or Copy Across Sheets.

Copy Cell Range

DsExcel 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**.

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

Example	Description
Copy(sheet.Range["E5"])	This method copies data from cell range **B3:D12** and pastes the data to cell **E5** onwards.

Copy(sheet.Range["E5:G14"])	This method copies data from cell range **B3:D12** and pastes the data in cell range **E5:G14**. In case the range of cells copied does not fit into the destination cell range, the data is lost.

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

C#

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

Cut Cell Range

DsExcel 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**.

DsExcel provide the following different ways to use Cut method.

Example	Description
Cut(sheet.Range["E5"])	This method cuts the data from cell range **B3:D12** and pastes the data to cell **E5** onwards.
Cut(sheet.Range["E5:G14"])	This method cuts the data from cell range **B3:D12** and pastes the data in cell range **E5:G14**. In case the range of cells cut does not fit into the destination cell range, the data is lost.

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

C#

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

Cut or Copy Shape, Slicer, Chart and Picture

DsExcel 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", "cate1");
ISlicer slicer = cache1.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "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();
```

Paste or Ignore Data in Hidden Range

DsExcel allows you to choose whether to copy and paste the data in a hidden range. The parameters of overloaded Copy method can be used to specify the destination where the copied data needs to be pasted, whether to paste the data in a hidden range and various paste types.

The **pasteOption** parameter of **Copy** method belongs to the **PasteOption** class. This class provides the **AllowPasteHiddenRange** property which if true, will paste the data in a hidden range to the destination, else will ignore the hidden range. The default value is true.

The **PasteOption** class also provides **PasteType** property which can be used to specify various paste types by setting it to any **PasteType** enumeration value.

The below table explains different options which can be used to specify the paste type using **PasteType** enumeration:

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

You can also combine two different paste type options. For example, if you want to paste values and number formats concurrently in a worksheet, you can use any of the below combinations:

```
PasteType.Values | PasteType.NumberFormats
PasteType.Formulas | PasteType.NumberFormats
```

Refer to the following example code to ignore and paste data in a hidden range. It also uses the combination of paste options while copying and pasting data.

C#

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

```
object[,] data = new object[,]{
{"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}
};

worksheet.Range["A1:F6"].Value = data;
worksheet.Range["A:F"].ColumnWidth = 15;
worksheet.Range["1:1"].Hidden = true;
worksheet.Range["3:3"].Hidden = true;
worksheet.Range["5:5"].Hidden = true;

//Ignore pasting data in hidden range
var worksheet2 = workbook.Worksheets.Add();
worksheet.Range["A1:F6"].Copy(worksheet2.Range["A1:F6"], new PasteOption {
AllowPasteHiddenRange = false });

//Paste data in hidden range
var worksheet3 = workbook.Worksheets.Add();
worksheet.Range["A1:F6"].Copy(worksheet3.Range["A1:F6"]);

//Ignore pasting data in hidden range and use PasteType options
var worksheet4 = workbook.Worksheets.Add();
worksheet.Range["A1:F6"].Copy(worksheet4.Range["A1:F6"], new PasteOption {
AllowPasteHiddenRange = false, PasteType = PasteType.ColumnWidths | PasteType.Values });

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

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.

DsExcel .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 occurences 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.

DsExcel 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

DsExcel 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

DsExcel 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.

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

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

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

C#

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

// Insert the range of cells
worksheet.Range["A3:A5"].Insert();
```

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

C#

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

Delete cell range

DsExcel 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.

DsExcel 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

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

Insert rows and columns

DsExcel 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

DsExcel 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

DsExcel 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

DsExcel 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 Custom Objects to a Range

DsExcel allows you to set custom objects or their 1D and 2D arrays to a range by using **Value** property of **IRange** interface. Custom objects are not supported for Excel I/O though.

Behavior of Custom Objects with Below Operations

- **Json Serialization and Deserialization:** Custom objects are discarded while performing Json serialization and deserialization (except for built-in SpreadJS interop types). However, this behavior can be overridden by setting the Workbook.ValueJsonSerializer property.
- **Export to PDF, HTML or Image Formats:** While exporting worksheets with custom objects to PDF, HTML or image formats, you can use Convert.ToString and then export custom objects as string. If the cell is too narrow to fit the content, ##### is exported.
- **Cut and Copy:** While performing cut and copy operations, custom objects can be copied or moved to another range, worksheet and workbook. Custom objects are always copied by reference. Some of them are even structures.
- **Range.Text**: While using Range.Text with custom objects, Convert.ToString method should be used.
- **Built-in formulas:** While using Built-in formulas, range references are treated as custom objects, including array formula expressions, such as {=A1:D3}. The following behavior is observed while using built-in formulas with custom objects:
  - Range references of custom objects in pattern matching (lookup) formulas are skipped
  - Range references of custom objects in aggregation formulas (mainly SUM*, statistical and database formulas) are skipped if custom data types do not make sense
  - Custom objects in aggregation formulas are accepted if custom data types are acceptable. For example, custom objects are counted in the COUNTA function
  - The below formulas return the specified value:
    - ISERROR: Always returns FALSE
    - TYPE: Returns #VALUE!
    - ERROR.TYPE: Returns #N/A
  - In all other cases, custom objects are treated as #VALUE!

  While using below operators and formulas with custom objects, the mentioned methods should be used:

Operators	=	Use Object.Equals
	<>	Use Not =
Formulas	EXACT	Use Object.ReferenceEquals
	TEXT	Use Convert.ToString

  The following barcode formulas can handle custom objects:

  - BC_CODABAR

- BC_CODE128
- BC_CODE39
- BC_CODE49
- BC_CODE93
- BC_DATAMATRIX
- BC_EAN13
- BC_EAN8
- BC_GS1_128
- BC_PDF417
- BC_QRCODE

Refer to the following example code to set 2D array of custom objects to a range.

C#

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

var activeSheet = workbook.ActiveSheet;
IRange a1 = activeSheet.Range["A1"];
var dict = new Dictionary<string, object>()
{
    {"TempData1", 1},
    {"TempData2", "Temp value 2"},
    {"TempData3", 3},
    {"TempData4", "Temp value 4"}
};
// Set temporary data to a range
a1.Value = dict;

// Display the custom object later
var obj = (IReadOnlyDictionary<string, object>)a1.Value;
var row = 1;
foreach (var kv in obj)
{
    activeSheet.Range["B" + row].Value = kv.Key;
    activeSheet.Range["C" + row].Value = kv.Value;
    row += 1;
}

// Arrange
activeSheet.Columns.AutoFit();
activeSheet.Columns[0].Hidden = true;

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

Refer to the following example code to override JSON serialization behavior.

C#

```
 // The JSON converter class
class JsonNetConverter<T> : IJsonSerializer
{
    public static JsonNetConverter<T> Instance { get; } = new JsonNetConverter<T>();

    public object Deserialize(string json)
    {
        var jobj = JObject.Parse(json);
        if (jobj.TryGetValue("typeName", out JToken jtok) &&
            jtok is JValue jval &&
            jval.Type == JTokenType.String &&
            (string)jval.Value == typeof(T).Name)
        {
            return JsonConvert.DeserializeObject<T>(json);
        } // End If
        return null;
    }

    public string Serialize(object value)
    {
        var jObj = JObject.FromObject(value);
        jObj.Add("typeName", new JValue(typeof(T).Name));
        return jObj.ToString(Newtonsoft.Json.Formatting.None);
    }
} // End Class ' JsonNetConverter

public void overrideJSON()
{
    // Usage
    Workbook.ValueJsonSerializer = JsonNetConverter<ValueWithUnit>.Instance;
}
```

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

DsExcel .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, DsExcel .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 DsExcel .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 = "Documents for Excel";
worksheet.Range["A1"].Columns.AutoFit();

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

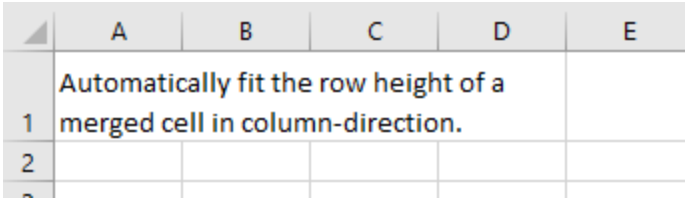
```
// Auto fit column width and row height of range 'C3'
worksheet.Range["C3"].Value = "Documents for Excel";
worksheet.Range["C3"].Font.Size = 32;
worksheet.Range["C3"].AutoFit();

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

You can also use the overloaded **AutoFit** method which provides **considerMergedCell** parameter. The parameter, when set to true, allows you to automatically fit the row height of a merged cell (in column-direction). Please note that the merged cell should not contain more than one row. The following example code showcases such a scenario.

C#

```
var workbook = new Workbook();
var sheet = workbook.ActiveSheet; sheet.Range["A1:D1"].Merge();
sheet.PageSetup.PrintGridlines = true;
sheet.Range["A1:D1"].WrapText = true;
sheet.Range["A1:D1"].Value = "Automatically fit the row height of a merged cell in
column-direction.";
sheet.Range["A1:D1"].EntireRow.AutoFit(true);
workbook.Save("AutoFitMergeRow.xlsx");
```

The output of above code will look like below in Excel:



Note: The Auto fit feature has the following limitations :

1) Apart from the above mentioned scenario, the AutoFit methods will not be applied in a merged cell. 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 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.

DsExcel 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 used range in selected 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;
```

Work with used range in selected range

To work with used range in a selected range, you can use **UsedRange** property and **GetUsedRange** method of the **IRange** interface. Once fetched, you can customize the used range using the properties of the **IRange** interface.

Note: In case of non-continuous selected range, the GetUsedRange method returns used range of the first range.

C#

```
worksheet.Range["B5"].Value = "Google";
```

```
worksheet.Range["D9"].Value = "Google";
worksheet.Range["F7"].Value = "Google";

// The used range1 is B5:D9
IRange usedRange1 =
workbook.Worksheets[0].Range["B3:E10"].GetUsedRange(UsedRangeType.Data);
usedRange1.Interior.Color = Color.Blue;

// The used range2 is F7
IRange usedRange2 = workbook.Worksheets[0].Range["E1:F8"].UsedRange;
usedRange2.Interior.Color = Color.Red;
```

After you get the used range of cells using any of the above methods, you can customize the fetched range. 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.

To view the code in action, see Get Used Range demo.

Measure Digital Width

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

DsExcel 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. DsExcel 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.

Set Default Values for Cell Range

When creating Excel documents or reports such as finance documents, sales reports, invoices, student status reports, forms, and others, it is often required to have cells with pre-filled text or fixed data rather than empty or blank cells that can be used in further processing and calculations.

The default value can help in such a scenario and provides a value or formula to be displayed and used in calculations like a normal cell value. DsExcel provides **DefaultValue** property in **IRange** interface that allows you to set the default value of a cell to avoid empty cell scenarios.

The default value has certain characteristics that are listed below:

- You can set the value and formula through the default value. The string starting with "=" will be considered a formula.
- If the cell is empty, its default value or formula will participate in recalculation.
- The logic behind the default value formula adjustment is the same as the cell formula.
- When exporting Excel, DsExcel exports the default value as a cell value when the cell is empty. If the default value is a formula, DsExcel discards it and only keeps the calculation result.
- You can clear the default value by setting null to the default value.
- Clear and UnMerge methods do not affect default values. The default values are filled after executing these methods.

Refer to the following example code to set the default value of cells:

C#

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

// Open defaultValue.xlsx.
workbook.Open("defaultValue.xlsx");
var worksheet = workbook.ActiveSheet;
```

```
// Set default value for standard reduction.
worksheet.Range["C4:C8"].DefaultValue = "=B4 - B4*0.12";

// Set normal value for specific percent reduction.
worksheet.Range["C6"].Formula = "=B6 - B6*0.08";
worksheet.Range["C8"].Formula = "=B8 - B8*0.05";

// Calculate total on default value and specific values.
worksheet.Range["C9"].Formula = "=SUM(C4:C8)";

// Save to an Excel file.
workbook.Save("DefaultValueOutput.xlsx");
```

Limitations

- You cannot set the default value for the entire row or column.
- DsExcel does not support the dynamic array when defaultValue sets a formula, so the cell default value will follow the implicit intersection policy when it is a reference and will get the top-left cell value when it is a dynamic array.
- You cannot copy the default value.

Set Cell Background Image for Cell Range

DsExcel enables you to set the cell background image and its layout for the cell range using **BackgroundImage** and **BackgroundImageLayout** properties in **IRange** interface. You can only export the cell background image to PDF, HTML, and IMG and can only view it in SpreadJS when saved in .sjs format. **BackgroundImageLayout** enumeration allows you to set the background image layout to Stretch (default), Center, Zoom, or None.

DsExcel only supports and exports the following image formats as the cell background image:

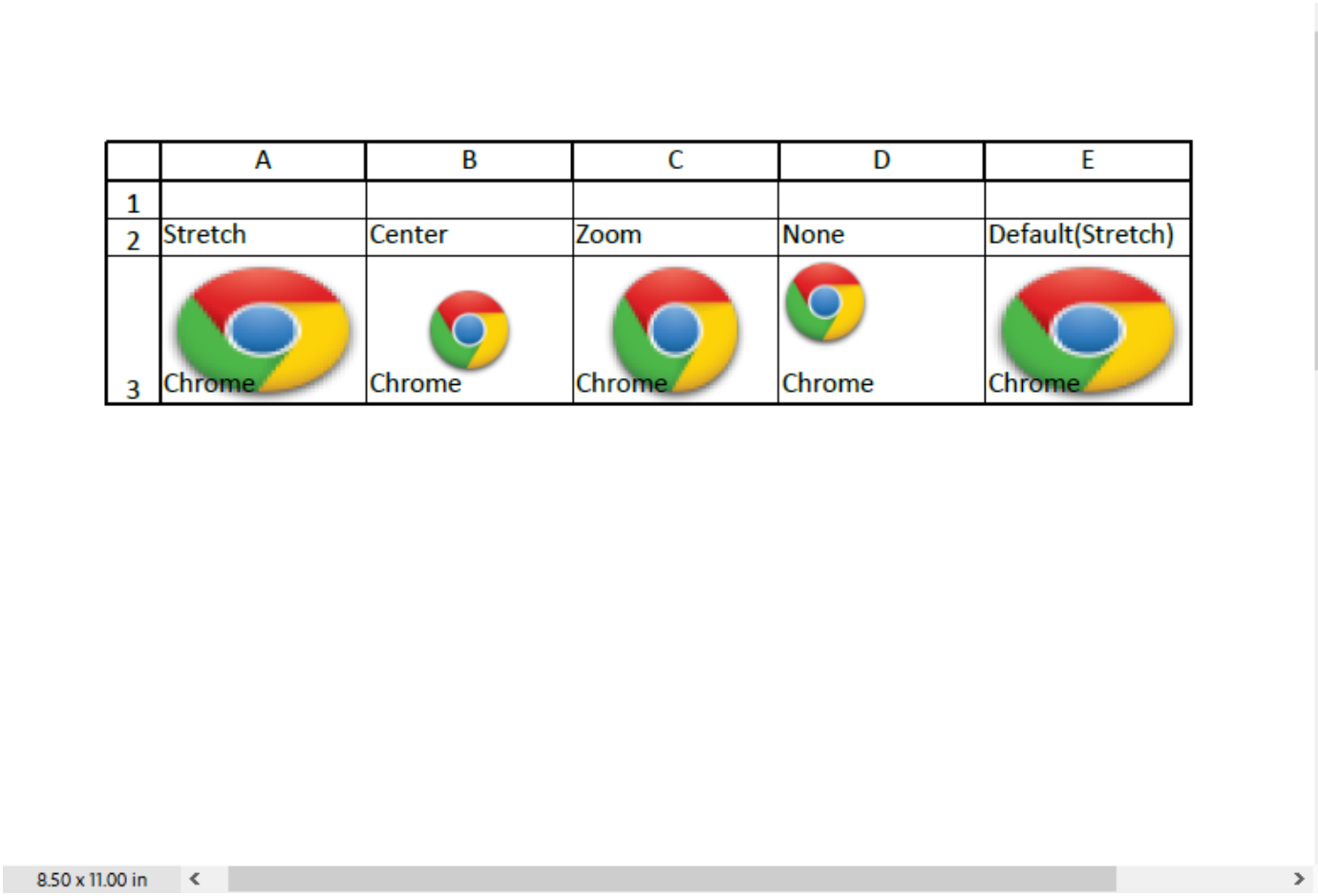
- PNG
- JPEG/JPG
- ICO
- SVG
- GIF

Refer to the following example code to add a cell background image in different layouts and export the workbook to a PDF file:

C#

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

IWorksheet worksheet = workbook.Worksheets[0];

// Load the image.
byte[] imageBytes = File.ReadAllBytes("Chrome_icon.png");

worksheet.Range["A2:E2"].Value = new string[] { "Stretch", "Center", "Zoom", "None",
```

```
"Default(Stretch)" };
worksheet.Range["A3:E3"].Value = "Chrome";
worksheet.Range["A3:E3"].RowHeightInPixel = 80;
worksheet.Range["A3:E3"].ColumnWidthInPixel = 100;

// Add cell background image.
worksheet.Range["A3:E3"].BackgroundImage = imageBytes;

// Set image layout.
worksheet.Range["A3"].BackgroundImageLayout = BackgroundImageLayout.Stretch;
worksheet.Range["B3"].BackgroundImageLayout = BackgroundImageLayout.Center;
worksheet.Range["C3"].BackgroundImageLayout = BackgroundImageLayout.Zoom;
worksheet.Range["D3"].BackgroundImageLayout = BackgroundImageLayout.None;

// Set PDF export options.
workbook.ActiveSheet.PageSetup.PrintGridlines = true;
workbook.ActiveSheet.PageSetup.PrintHeadings = true;

// Save to a PDF file.
workbook.Save("CellBackgroundImage.pdf");
```



Limitations

DsExcel does not support saving the background image to Excel; hence, you cannot view it in Excel.

Ignore Errors in Cell Range

Sometimes, when working with numbers and calculating formulas, Excel evaluates for any errors and points out the errors by showing the green triangle at the top-left corner of the cell. To avoid error evaluation and showing errors with the green triangle, DsExcel provides **IgnoredError** property in **IRange** interface and **IgnoredErrorType** enumeration to enable you to ignore errors such as invalid formula results, numbers stored as text, inconsistent formulas in adjacent cells, and others, and not show the green triangle at the top-left corner of the cell in a specific cell range in Excel.

IgnoredError property will not change when copying or cutting rows, columns, or cells, whereas it will move or delete when inserting or deleting rows, columns, or cells. The property will be copied or moved when copying or moving the sheet. IgnoredError property of the top-left cell of the first cell rect will be returned when getting IgnoredError.

DsExcel supports ignoring the following types of errors:

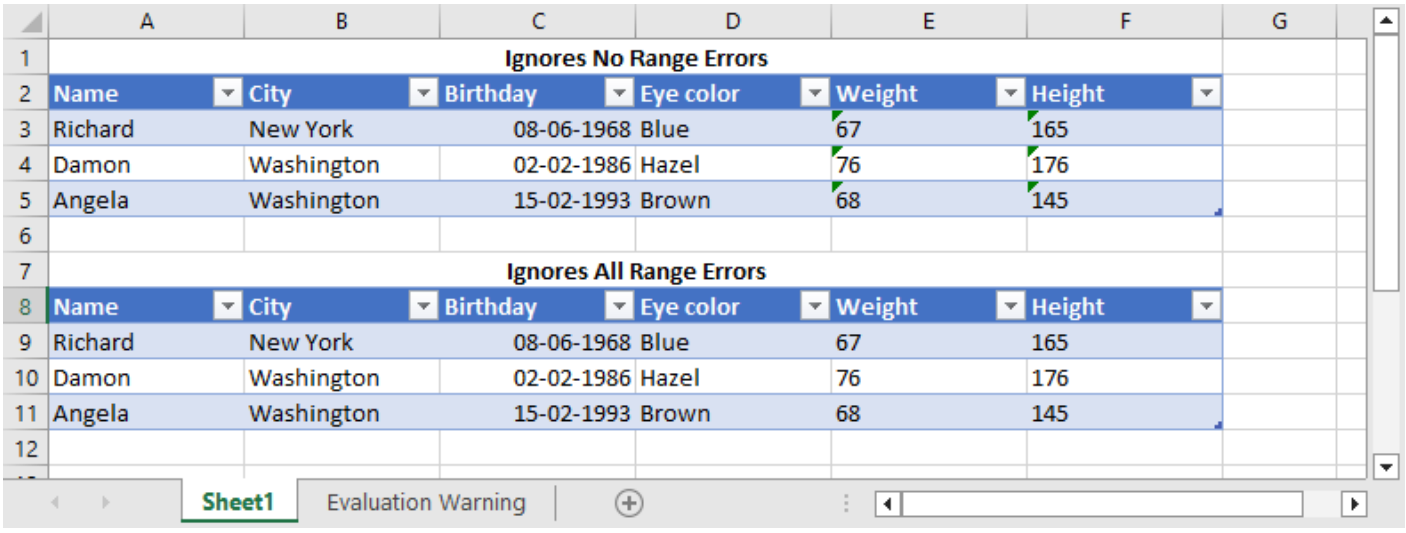
Error Type	Description
None	No errors are ignored.
InconsistentListFormula	Ignores the error of discrepancies in formulas within a calculated column.
InconsistentFormula	Ignores the error of discrepancies in formulas within a range.
OmittedCells	Ignores the error in cells containing formulas referring to a range that omits adjacent cells that could be included.
TextDate	Ignores the error when formulas contain text-formatted cells with years misinterpreted as the wrong century.
EmptyCellReferences	Ignores the error when a formula contains a reference to an empty cell.
ListDataValidation	Ignores the error of the cell value that does not comply with the Data Validation rule that restricts data to predefined items in a list.
EvaluateToError	Ignores the error of the formula result.
NumberAsText	Ignores the error in cells containing numbers stored as text or preceded by an apostrophe.
UnlockedFormulaCells	Ignores the error in unlocked cells containing formulas.
All	Ignores all types of errors.

Refer to the following example code to ignore all types of errors in the specified range:

C#

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

// Add data object.
object[,] data = new object[,]{
    {"Name", "City", "Birthday", "Eye color", "Weight", "Height"},
```

```
    {"Richard", "New York", new DateTime(1968, 6, 8), "Blue", "67", "165"},
    {"Damon", "Washington", new DateTime(1986, 2, 2), "Hazel", "76", "176"},
    {"Angela", "Washington", new DateTime(1993, 2, 15), "Brown", "68", "145"}
};

// No errors are ignored in this range.
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A:F"].ColumnWidth = 15;
worksheet.Range["A1:F1"].Merge();
worksheet.Range["A1:F1"].Value = "Ignores No Range Errors";
worksheet.Range["A1:F1"].Font.Bold = true;
worksheet.Range["A1:F1"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["A2:F5"].Value = data;
worksheet.Tables.Add(worksheet.Range["A2:F5"], true);

// Ignores all errors in this range.
worksheet.Range["A7:F7"].Merge();
worksheet.Range["A7:F7"].Value = "Ignores All Range Errors";
worksheet.Range["A7:F7"].Font.Bold = true;
worksheet.Range["A7:F7"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["A8:F11"].Value = data;
worksheet.Tables.Add(worksheet.Range["A8:F11"], true);

// Ignore error in range A8:F11.
worksheet.Range["A8:F11"].IgnoredError = IgnoredErrorType.All;

// Save Excel file.
workbook.Save("IgnoreRangeError.xlsx");
```



	A	B	C	D	E	F
1	Ignores No Range Errors					
2	Name	City	Birthday	Eye color	Weight	Height
3	Richard	New York	08-06-1968	Blue	67	165
4	Damon	Washington	02-02-1986	Hazel	76	176
5	Angela	Washington	15-02-1993	Brown	68	145
6						
7	Ignores All Range Errors					
8	Name	City	Birthday	Eye color	Weight	Height
9	Richard	New York	08-06-1968	Blue	67	165
10	Damon	Washington	02-02-1986	Hazel	76	176
11	Angela	Washington	15-02-1993	Brown	68	145
12						

Sheet1 | Evaluation Warning

Freeze Panes in a Worksheet

DsExcel 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

DsExcel 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 **UnfreezeTrailingPanes** 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.FreezeTrailingPanes(2, 3);

//unfreeze trailing pane
worksheet.UnfreezeTrailingPanes();

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

Customize Worksheets

DsExcel 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 DsExcel.

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 = "Documents";
worksheet.Range["C2"].Value = 4;
worksheet.Range["D1"].Value = "Excel";
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(@"Logo.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 wokrsheet
worksheet.StandardHeight = 20;
worksheet.StandardWidth = 40;
```

Worksheet Views

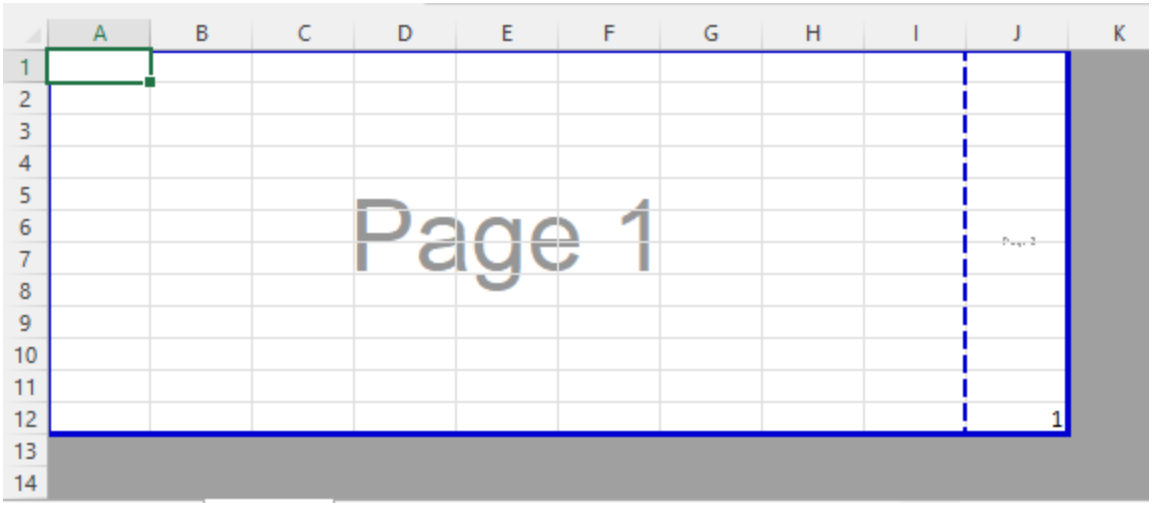
DsExcel offers various options to customize display settings that are applied to a worksheet. You can either choose from

pre-defined views or customize the view settings to get the preferred display. You can also save customized views in a workbook and apply them later.

Pre-defined Views

DsExcel.NET, similar to MS Excel, provides pre-defined views to make it easier for users to preview the page layout and page breaks before printing the document.

- Default View - Default view of the worksheet
- Page Layout View - Gives a preview of document to be printed by showing start and end of pages including headers and footers of the document.
- Page Break View - Displays position of page breaks in the document to be printed.



These pre-defined views can be set using **ViewType** property of the **IWorksheetView** interface.

C#

```
IWorkbook workbook = new Workbook();
IWorksheet worksheet = workbook.ActiveSheet;

worksheet.Range["J12"].Value = 1;

//Set the view mode of the worksheet to PageBreakPreview.
worksheet.SheetView.ViewType = ViewType.PageBreakPreview;

//Modify the zoom of the PageBreakPreview to 80%.
worksheet.SheetView.Zoom = 80;

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

View Settings

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

interface.

C#

```
//Set worksheet view

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

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

C#

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

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

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

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.

DsExcel.NET also lets you save custom views in the workbook. To learn more about custom views, see Custom Views.

Cell Types

DsExcel 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. DsExcel 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 = "MESCIUS, Inc. Website";
    hyperlinkCell.LinkColor = "Blue";
    hyperlinkCell.LinkToolTip = "MESCIUS, Inc. Website";
    hyperlinkCell.VisitedLinkColor = "Green";
    hyperlinkCell.Target = HyperLinkTargetType.Blank;

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

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

Range Template Cell

DsExcel 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		
Asset Type	Amount	Rate
Savings	$25,000	6.88%
Shares	$55,000	15.13%
Stocks	$15,000	4.13%
House	$250,000	68.78%
Bonds	$11,000	3.03%
Car	$7,500	2.06%
Total	$363,500	

Icey		
Asset Type	Amount	Rate
Savings	$30,000	14.08%
Shares	$45,000	21.13%
Stocks	$25,000	11.74%
House	$20,000	9.39%
Bonds	$18,000	8.45%
Car	$75,000	35.21%
Total	$213,000	

Walter		
Asset Type	Amount	Rate
Savings	$20,000	9.30%
Shares	$4,000	1.86%
Stocks	$95,000	44.19%
House	$30,000	13.95%
Bonds	$10,000	4.65%
Car	$56,000	26.05%
Total	$215,000	

Chris		
Asset Type	Amount	Rate
Savings	$70,000	25.93%
Shares	$85,000	31.48%
Stocks	$35,000	12.96%
House	$20,000	7.41%
Bonds	$15,000	5.56%
Car	$45,000	16.67%
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

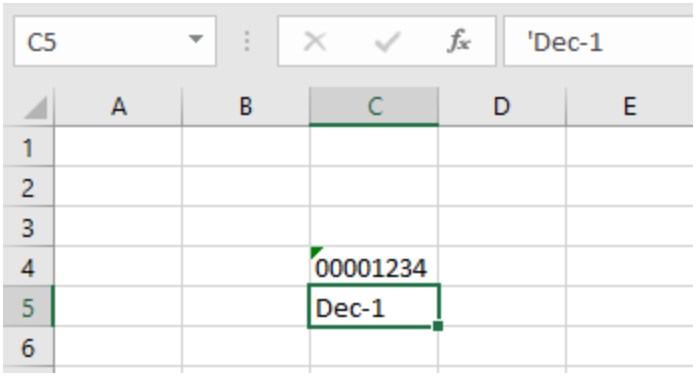
DsExcel library provides the Quote Prefix feature just like Microsoft Excel. You can add a single quote or an apostrophe as a prefix to handle the cell value as text. The quote prefix remains hidden and only the cell value is visible. The single quote prefix can be seen in the formula bar when the user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

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

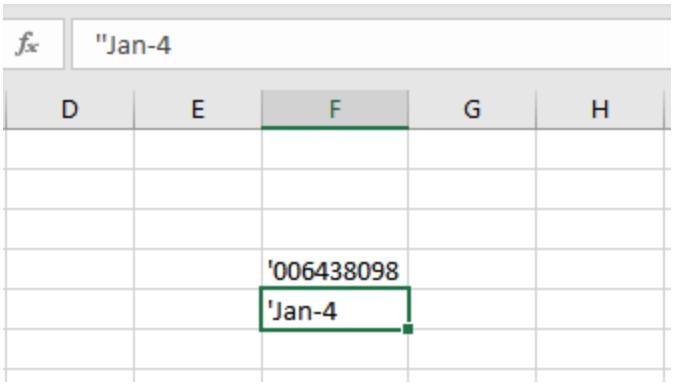
C#

```
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 DsExcel, 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;
```

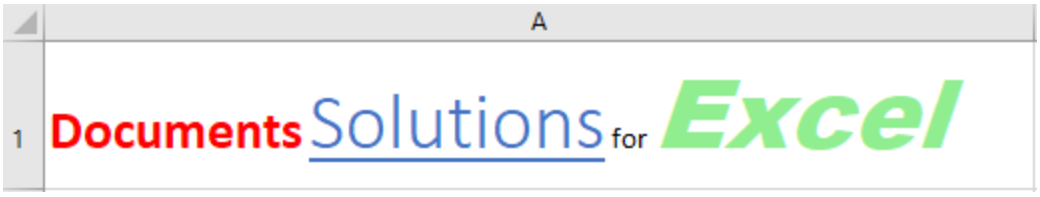
```
    public Student(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
}
internal class NetJsonSerializer : IJsonSerializer
{
    public object Deserialize(string json)
    {
        return Newtonsoft.Json.JsonConvert.DeserializeObject(json) as
Newtonsoft.Json.Linq.JObject;
    }

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

Rich Text

DsExcel.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 "Solutions" is formatted with a custom font size, underline style and blue color. Similarly, the text "Documents" and "Excel" has been formatted using multiple styles.



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

// Append string "Solutions" to IRichText object and apply formatting
ITextRun run2 = richText.Add("Solutions");
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 = "Documents Solutions for Excel";

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

// Formatting string "Documents"
ITextRun run1 = worksheet.Range["A1"].Characters(0, 9);
run1.Font.Color = Color.Red;
run1.Font.Bold = true;
run1.Font.Size = 20;

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

// Append string "Solutions" to IRichText object and apply formatting
ITextRun run2 = richText.Add("Solutions");
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 "Solutions" to run1 and apply formatting
ITextRun run2 = run1.InsertBefore("Solutions");
run2.Font.ThemeFont = ThemeFont.Major;
run2.Font.ThemeColor = ThemeColor.Accent1;
run2.Font.Size = 30;
run2.Font.Underline = UnderlineType.Single;

// Use InsertBefore() to add string "Documents " to run2 and apply formatting
ITextRun run3 = run2.InsertBefore("Documents ");
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.

DsExcel .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 DsExcel, 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 DsExcel.

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. The default value is false

DoNotAutoFitAfterOpened	Do not autofit the row height after loading the file. The default value 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;

//Don’t autofit row height
options1.DoNotAutoFitAfterOpened = 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
- .xltx
- .csv
- .json
- .ssjson
- .sjs

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);
```

Sometimes workbook may contain many unused styles, unused defined names, or empty cells with styles applied on it which increases the file size. The **XlsxSaveOptions** class provides you with options to save .xlsx files without these unnecessary items so that you can optimize the file size. The **ExcludeEmptyRegionCells** property of the class, when set to true, lets you exclude the empty cells outside the used data range. Similarly, you can exclude the unused names and styles of a workbook while exporting by setting the **ExcludeUnusedNames** and **ExcludeUnusedStyles** properties to **true**. The class also provides **IgnoreFormulas** property which lets you save the formula cells as value cells in the saved .xlsx file. You can even save the workbook in compact mode if you do not require extended features of the workbook after saving it by using the **IsCompactMode** property.

The sample code below shows how to save your workbook using various options:

C#

```
Workbook workbook = new Workbook();

//Add names and style for current workbook
for (int i = 0; i < 10000; i++)
{
  workbook.Names.Add($"name{i + 1}", $"=$A${i + 1}");
  workbook.Styles.Add($"style{i + 1}");
}

//Exclude the unused styles, names and empty cell region
XlsxSaveOptions option = new XlsxSaveOptions();
option.ExcludeUnusedStyles = true;
option.ExcludeUnusedNames = true;
option.ExcludeEmptyRegionCells = true;

//save file with and without options
workbook.Save("test_optimized.xlsx", option);
workbook.Save("test_no_optimized.xlsx");
```

Apart from saving files in .xlsx format, you can also save files in the below file formats by using the overloads of **Save** method in Workbook class:

- .xlsm
- .xltx
- .csv
- .html
- .pdf
- .json
- .ssjson
- .sjs

To view the code in action, see Option to optimize file size demo.

Protect Workbook

DsExcel 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

DsExcel 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

A workbook can be either modified by making changes in its data or by changing its structure and window.
Hence, DsExcel allows you to protect a workbook from modification in following two ways:

Protect Workbook from Modifying Data

In many cases, you may want to share workbook data only for reference and do not want anyone to carry out unauthorised editing, intentionally or accidentally. For instance, a workbook storing list of expenses incurred during the office set up needs to be shared with stake holders while protecting its data from any kind of tampering.

DsExcel .NET provides **WriteProtection** class of the **IWorkbook** interface which lets you set protect options while saving

the workbook. This class lets you recommend the user to open only in reading mode by using **ReadOnlyRecommended** property or you can set the "modification password" through **WritePassword** property. DsExcel automatically opens the document when user provides the correct modification password. However, the behavior can be customized using result of **ValidatePassword** property. The WriteProtection class also provides **WriteReserved** and **WriteReservedBy** property to fetch whether the workbook is protected and by whom.

Refer to following example code to protect the workbook data from modification.

C#

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

// Open an excel file.
var fileStream = GetResourceStream("xlsx\\Medical office start-up expenses 1.xlsx");
workbook.Open(fileStream);

// Specify the name of the user who reserves write permission on this workbook.
workbook.WriteProtection.WriteReservedBy = "Eric";

// Specify this workbook is saved as read-only recommended.
// Microsoft Excel displays a message recommending that you open the workbook as read-
only.
workbook.WriteProtection.ReadOnlyRecommended = true;

// Protects the workbook with a password so that other users cannot accidentally or
intentionally edit the data.
// The write-protection only happens when you open it with an Excel application.
workbook.WriteProtection.WritePassword = "Y6dh!et5";

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

Protect Workbook from Modifying Structure and Windows

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 DsExcel .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

DsExcel 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

DsExcel 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. You can also customize display settings of the workbook by using the properties such as **DisplayHorizontalScrollBar**, **DisplayHorizontalScrollBar**, **DisplayWorkbookTabs** and **TabRatio** of the **IWorkbookView** interface.

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;
```

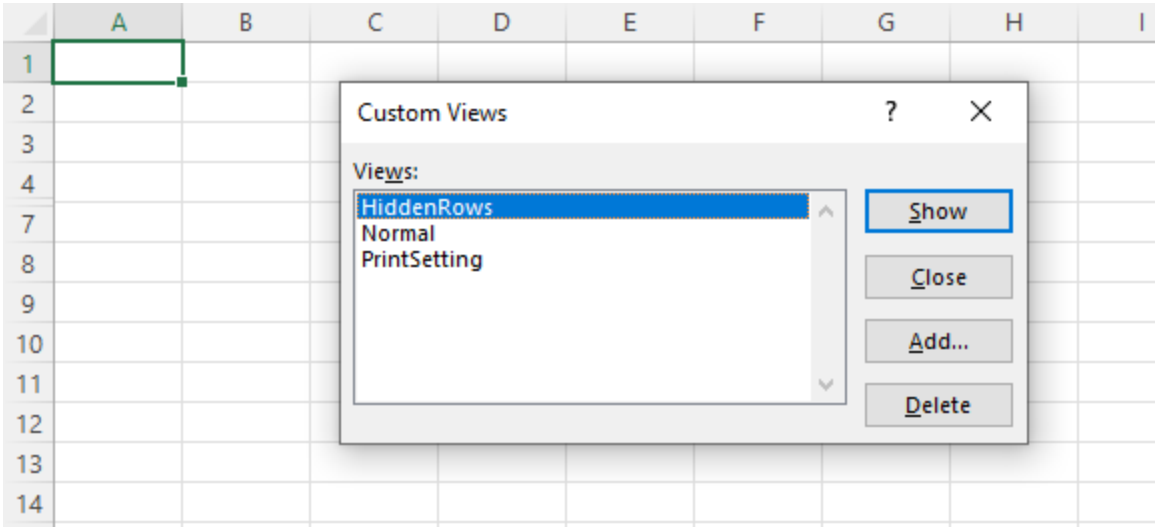
Custom Views

DsExcel.NET also lets you save the views created by customizing properties or methods as custom views. Collection of custom views stored in a workbook is represented by **IWorkbook.CustomViews**. Each custom view in the collection is

represented by **ICustomView** object and stores the view state of all worksheets in the current workbook. You can set **Name**, **RowColSettings** and **PrintSettings** properties for each custom view. To add a new custom view to a workbook, you can use the **Add** method. If a custom view name already exists in the collection, the new view overwrites the existing custom view. To apply a custom view to the workbook, you can use **ICustomView.Show** method. Similarly, you can delete custom views from the collection by calling **ICustomView.Delete** method.

Note: The Add method does not allow addition of custom view and throws an exception in following cases:

- When workbook contains a table or pivot table
- When name of the custom view is null or whitespace



C#

```
IWorkbook workbook = new Workbook();
IWorksheet worksheet1 = workbook.ActiveSheet;
IWorksheet worksheet2 = workbook.Worksheets.Add();

worksheet1.Range["J12"].Value = 1;
worksheet2.Range["J12"].Value = 2;

// Add the normal view.
workbook.CustomViews.Add("Normal", true, true);

worksheet1.PageSetup.PrintArea = "$C$4:$J$12";
worksheet2.PageSetup.PaperSize = PaperSize.A4;

// Add a new view which saves the current page settings of all worksheets.
workbook.CustomViews.Add("PrintSetting", true, false);

worksheet1.Range["5:6"].Hidden = true;
worksheet2.Range["5:6"].Hidden = true;

// Add a new view which saves the current hidden rows/columns, filter settings of all
```

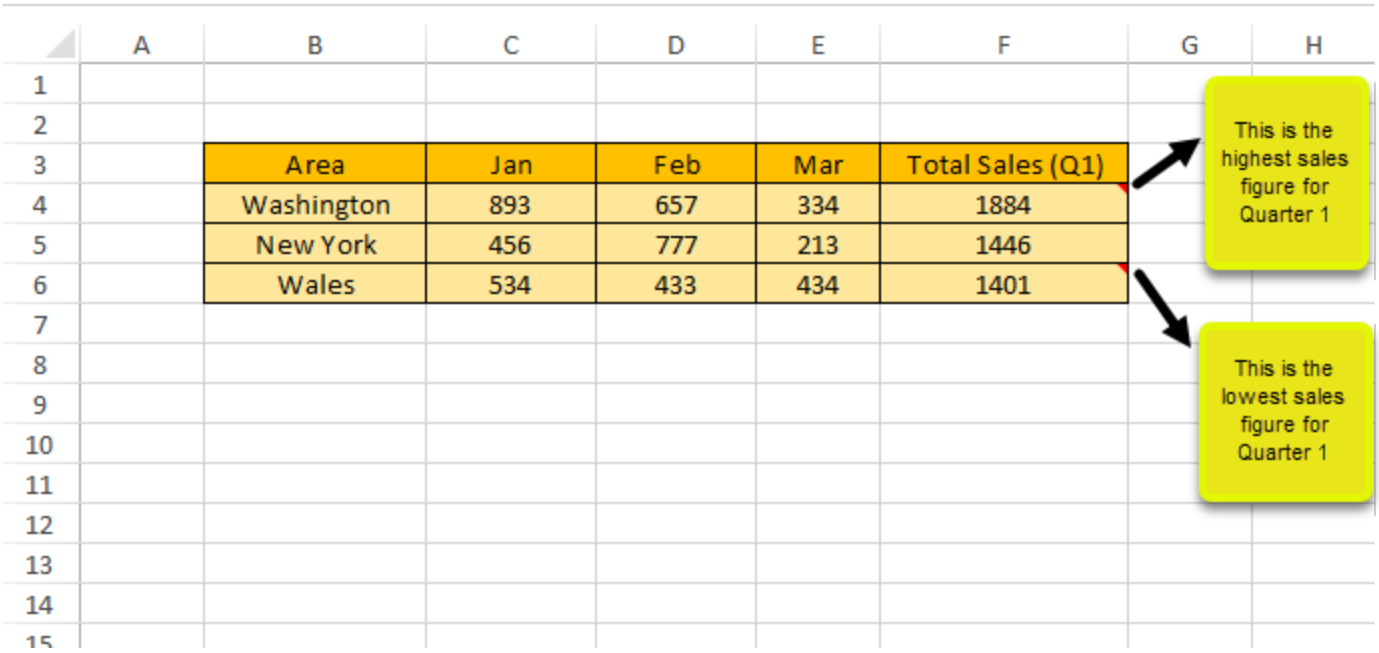
```
worksheets.
workbook.CustomViews.Add("HiddenRows", false, true);

// Apply PrintSetting custom view
workbook.CustomViews["PrintSetting"].Show();

// In the exported file, the user can switch between custom views.
workbook.Save("CustomViewAdd.xlsx");
```

Comments

DsExcel .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.



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 DsExcel .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.

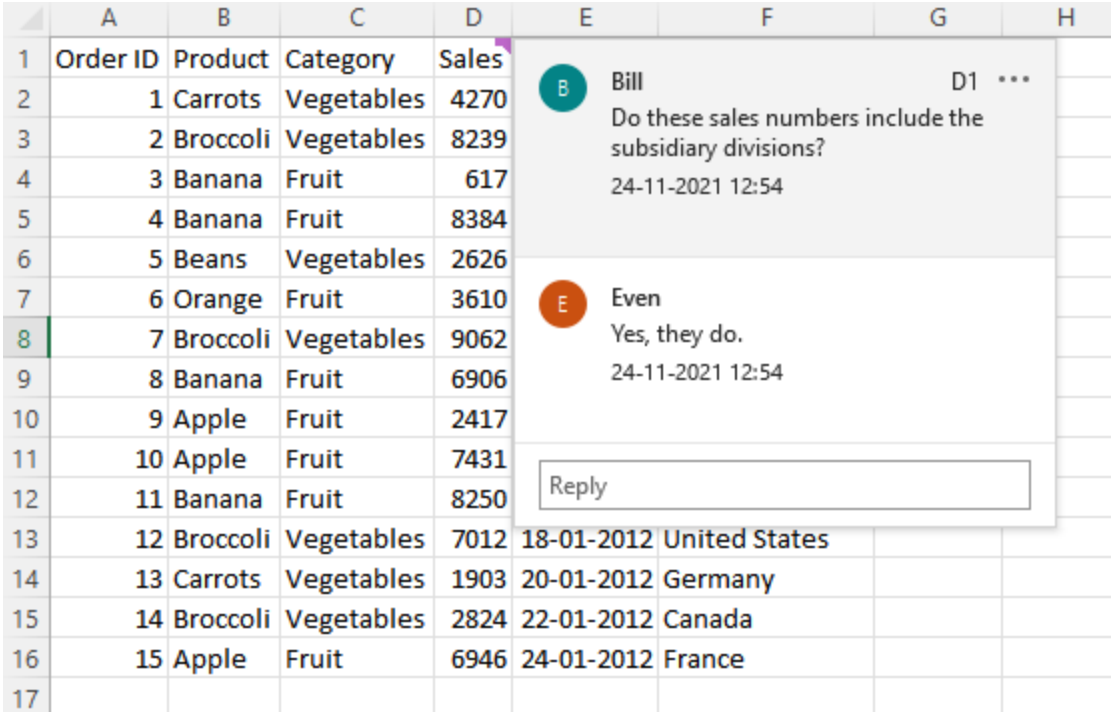
C#

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

Threaded Comments

Threaded comment refers to the spreadsheet comments which appear as a conversation or discussion. These comments provide a reply box which allows users to respond on a comment resulting into a conversation.

DsExcel.NET allows you to add threaded comments through **ICommentThreaded** interface. Each comment in the collection of threaded comments on a particular worksheet is represented by **ICommentsThreaded** object. The comments are stored in collection in the order of row and then column.



	A	B	C	D	E	F	G	H
1	Order ID	Product	Category	Sales				
2	1	Carrots	Vegetables	4270				
3	2	Broccoli	Vegetables	8239				
4	3	Banana	Fruit	617				
5	4	Banana	Fruit	8384				
6	5	Beans	Vegetables	2626				
7	6	Orange	Fruit	3610				
8	7	Broccoli	Vegetables	9062				
9	8	Banana	Fruit	6906				
10	9	Apple	Fruit	2417				
11	10	Apple	Fruit	7431				
12	11	Banana	Fruit	8250				
13	12	Broccoli	Vegetables	7012	18-01-2012	United States		
14	13	Carrots	Vegetables	1903	20-01-2012	Germany		
15	14	Broccoli	Vegetables	2824	22-01-2012	Canada		
16	15	Apple	Fruit	6946	24-01-2012	France		
17								

Note: A threaded comment is converted into a comment when the worksheet is exported to JSON.

Create Threaded Comments

To create threaded comment to a worksheet cell, you can use **AddCommentThreaded** method of the IRange interface. This method accepts comment text and author's name as its parameters.

C#

```
// Create threaded comment for range C3.
ICommentThreaded commentThreadedC3 = worksheet.Range["C3"].AddCommentThreaded("Do these sales numbers include the subsidiary divisions?", "Bill");

// Add a reply for commentThreadedC3.
ICommentThreaded Reply = commentThreadedC3.AddReply("Yes, they do.", "Even");
```

Add Reply to Threaded Comment

To add reply to a threaded comment, you can use **AddReply** method of the **ICommentsThreaded** interface. The method adds the reply to collection of replies if the threaded comment is a top-level threaded comment. In case the threaded comment is a reply, the method adds reply to the parent's collection of replies. The collection of replies can be fetched using **Replies** property.

C#

```
// Add replies for commentThreadedC3.
ICommentThreaded Reply_1 = commentThreadedC3.AddReply("Mark's reply", "Mark");
ICommentThreaded Reply_2 = commentThreadedC3.AddReply("Bill's reply", "Bill");
```

Modify Threaded Comment

To modify a threaded comment, you can set **Text** property of the threaded comment to a new string.

C#

```
// Sets a new text for commentTHreadedC3.
commentThreadedC3.Text = "New Content";

// Delete Reply_1
Reply_1.Delete();

// Sets a new text for Reply_2.
Reply_2.Text = "Bill's new reply";
```

Read Threaded Comment

DsExcel.NET provides various properties using which you can read the threaded comments and their attributes such as date stamp, author etc. To fetch the whole collection of threaded comments on a worksheet, you can use the **IWorksheet.CommentsThreaded** method, while a specific threaded comment can be obtained from **ICommentsThreaded** collection by using the **Index** property**.** You can also get the number of total comments on a worksheet by using the **Count** property. The ICommentsThreaded interface also provides **Parent** property to get parent if the threaded comment is a reply. To read the date stamp or author of a comment, you can use **Date** and **Author** properties. You can also fetch **Next** or **Previous** comments of a threaded comment in a worksheet.

C#

```
// Get a specific comment
Console.WriteLine("Get a specific comment : " + worksheet.CommentsThreaded[0].Text);

// Get replies count
Console.WriteLine("Replies Count : " + commentThreadedC3.Replies.Count);

// Get all replies on a specific comment
for (int i = 0; i < commentThreadedC3.Replies.Count; i++)
    Console.WriteLine("Get replies text[" + i + "] : " +
commentThreadedC3.Replies[i].Text);

// Get author of comment
```

```
Console.WriteLine("Author Name : " + commentThreadedC3.Author.Name.ToString());

// Get collection of threaded comments
Console.WriteLine("Get collection of threaded comments : " +
worksheet.CommentsThreaded.Count);

// Get Date stamp of comment
Console.WriteLine("Date of Comment : " + commentThreadedC3.Date.ToShortDateString());

// Get Time stamp of comment
Console.WriteLine("Time of Comment : " + commentThreadedC3.Date.ToShortTimeString());

// Get C3's next comment
Console.WriteLine("C3's next comment : " + commentThreadedC3.Next().Text);

// Get D3's previous comment
Console.WriteLine("D3's previous comment : " + commentThreadedD3.Previous().Text);

// Get parent of a reply
Console.WriteLine("Parent of Reply_1 : " + Reply_1.Parent.Text);
Console.WriteLine("Parent of Reply_2 : " + Reply_2.Parent.Text);
```

Below is the outcome displayed on console after above processing:

```
Get a specific comment : New Content
Replies Count : 2
Get replies text[0] : Evan's reply
Get replies text[1] : Steve's reply
Author Name : Angelina
Get collection of threaded comments : 2
Date of Comment : 11/17/2021
Time of Comment : 5:14 PM
C3's next comment : Range D3's threaded comment
D3's previous comment : New Content
Parent of Reply_1 : New Content
Parent of Reply_2 : New Content
```

Disable Threaded Comment

DsExcel.NET provides **IsResolved** property to set the resolve status of a threaded comment. Setting this property to true also means, that the threaded comment is disabled and user cannot edit or reply to that comment.

C#

```
//Disable threaded comment
commentThreadedC3.IsResolved = true;
```

Clear Threaded Comment

To delete a threaded comment and replies associated with that comment, you can use the **ICommentThreaded.Delete**

method. If the target CommentThreaded object is a top-level comment, this method removes the specified comment. However, if the target comment is a reply in one of the comment threads, this method removes that reply from reply collection of the parent comment. You can also clear threaded comments from a range of cells by using the **IRange.ClearCommentsThreaded** method.

C#

```
// Create threaded comment for range C3.
ICommentThreaded commentThreadedE3 = worksheet.Range["E3"].AddCommentThreaded("Range E3's threaded comment", "Even");

// Add a reply for commentThreadedC3.
worksheet.Range["F3"].AddCommentThreaded("Range F3's threaded comment", "Charles");
#region Delete
// Delete a single cell threaded comment.
commentThreadedE3.Delete();

// Clear a range of cells threaded comment.
worksheet.Range["F3:G4"].ClearCommentsThreaded();
```

Set Author Name

To set author of a comment, you can use **Name** property of the **IAuthor** interface.

C#

```
//Set author name
IAuthor author = commentThreadedC3.Author;
author.Name = "Alex";
```

Hyperlinks

DsExcel .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\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"],
                                    "https://developer.mescius.com/",
                                    null,
                                    "open MESCIUS, Inc. web site.",
                                    "MESCIUS, Inc.");
```

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@mescius.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: "https://developer.mescius.com/"	null
A range in this document	Example: null	"Sheet1!$C$3:$E$4"
Email Address	Example: "mailto: abc.xyz@mescius.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"],
"https://developer.mescius.com/");
worksheet.Range["J6"].Hyperlinks.Add(worksheet.Range["A1"],
"https://developer.mescius.com/");

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

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

Sort

DsExcel 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 DsExcel.

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, DsExcel 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.

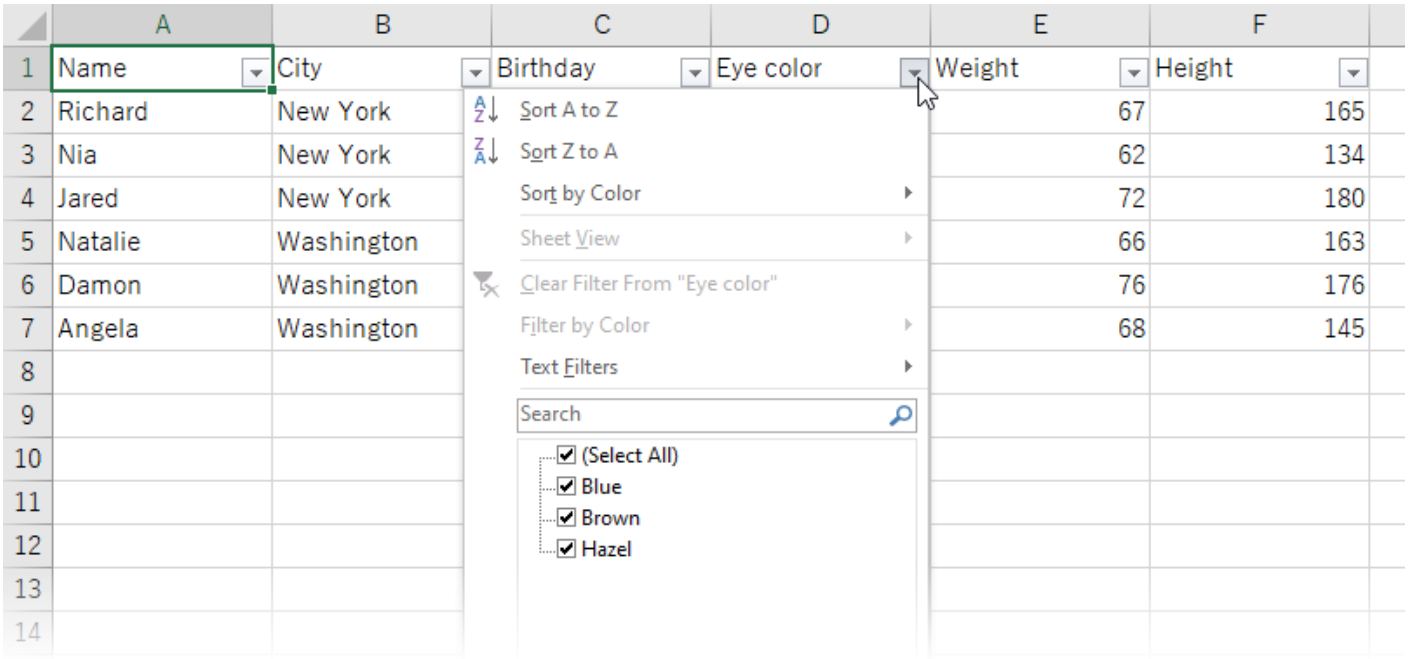
In DsExcel, you can apply filters to a selected range of data. For example, you can apply date type filter from C4 to C7 range. To filter data in a range of cells or a table, you need to set the auto filter mode for the worksheet to boolean true or false using **AutoFilterMode property** of the **IWorksheet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

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

- **Create filter without condition**
- **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**

Create filter without condition

DsExcel allows you to create a filter without condition by using the **IRange.AutoFilter** class method. This method creates an empty filter if no condition is set in the worksheet already. You can also create filter for a specific field by passing the optional *field* parameter to the method.



C#

```
//Create filters without condition.
worksheet.Range["A1:F7"].AutoFilter();
```

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 criteria1 = new DateTime(2008, 1, 1).ToString();
var criteria2 = new DateTime(2008, 8, 1).ToString();
worksheet.Range["D20:F29"].AutoFilter(2, ">=" + criteria1, 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

DsExcel .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 DsExcel .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
- Row or Column Group Information

> **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 DsExcel .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 DsExcel, 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#

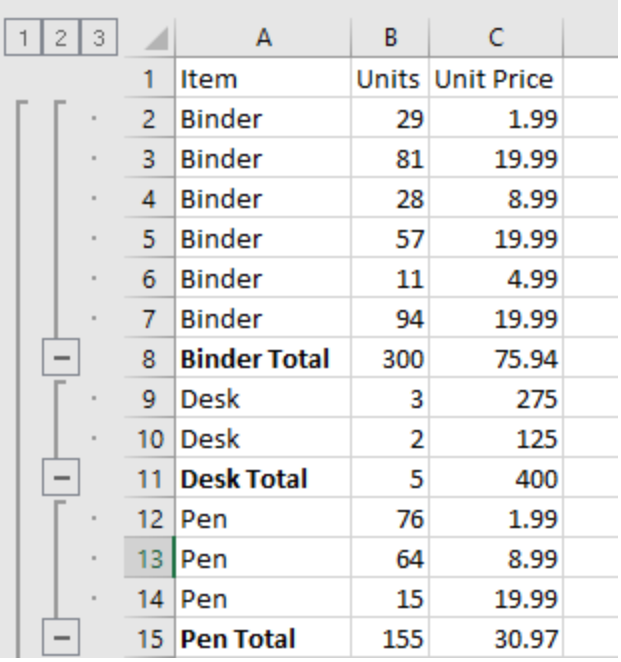
```
public void PopulateData(Workbook workbook)
{
    IWorksheet worksheet = workbook.Worksheets[0];

    // 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");
```



	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	**Binder Total**	300	75.94
9	Desk	3	275
10	Desk	2	125
11	**Desk Total**	5	400
12	Pen	76	1.99
13	Pen	64	8.99
14	Pen	15	19.99
15	**Pen Total**	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.

DsExcel 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

Row or Column Group Information

DsExcel allows you to retrieve the information of row or column groups by using the **RowGroupInfo** or **ColumnGroupInfo** property of **IOutline** interface.

You can identify the cell ranges where the grouping exists and can expand or collapse the groups by using the **Expand** and **Collapse** methods of **IGroupInfo** interface.

The **IGroupInfo** interface also provides **StartIndex**, **EndIndex**, **Level**, **Parent**, **Children** and **IsCollapsed** properties which can be used to retrieve grouping information such as start or end index, level, parent, children or collpased status of the group.

Get Row Group Information

Refer to the below example code which uses **RowGroupInfo** property to get the row group information, **Collapse** method to collapse groups and identifies the rows where row level is two:

C#

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

var sheet = workbook.ActiveSheet;
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}
};

sheet.Range["A1:F7"].Value = data;
sheet.Range["A:F"].ColumnWidth = 15;

sheet.Range["A:F"].Group();
sheet.Range["A:B"].Group();
sheet.Range["D:E"].Group();

//get group information and collapes some group.
var groupInfo = sheet.Outline.ColumnGroupInfo;
foreach (var item in groupInfo)
{
    if (item.Children != null)
    {
        foreach (var childItem in item.Children)
        {
            if (childItem.StartIndex > 2)
            {
                childItem.Collapse();
            }
        }
    }
}

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

Get Column Group Information

Refer to the below example code which uses **ColumnGroupInfo** property to get the row group information and **Collapse** method to collapse groups:

C#

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

var sheet = workbook.ActiveSheet;
IRange targetRange = sheet.Range["A1:C9"];
// Set data
targetRange.Value = new object[,]
{
{"Player", "Side", "Commander"},
{1, "Soviet", "AI"},
{2, "Soviet", "AI"},
{3, "Soviet", "Human"},
{4, "Allied", "Human"},
{5, "Allied", "Human"},
{6, "Allied", "AI"},
{7, "Empire", "AI"},
{8, "Empire", "AI"}
};

// Subtotal
targetRange.Subtotal(groupBy: 2, // group by 'Side' column
    subtotalFunction: ConsolidationFunction.Count,
    totalList: new[] { 2 } // displays total of 'Side' column
    );

targetRange.AutoFit();

//get group information and collapes some group.
var groupInfo = sheet.Outline.RowGroupInfo;
var rowInfo = new Dictionary<int, int>();
foreach (var item in groupInfo)
{
    if (item.Children != null)
    {
        foreach (var childItem in item.Children)
        {
            if (childItem.StartIndex > 3)
            {
                childItem.Collapse();
            }
            if (childItem.Level == 2)
            {
                rowInfo.Add(childItem.StartIndex, childItem.EndIndex);
```

```
            }
        }
    }
}

StringBuilder builder = new StringBuilder();
foreach (var item in rowInfo)
{
    builder.Append("row " + (item.Key + 1) + " to row " + (item.Value + 1) + ", ");
}

sheet.Range["A15"].Value = "The rows where the group level is 2 are: " +
builder.ToString();
sheet.Range["A15"].RowHeight = 25;
sheet.Range["A15"].Font.Color = Color.Red;
sheet.Range["A15"].Font.Size = 15;

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

Conditional Formatting

In order to highlight important information in rows or columns of a worksheet, DsExcel .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 occuring 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 = DataBarAxisPosition.Automatic;
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

DsExcel .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 DsExcel .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 DsExcel .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

DsExcel 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 JSON string, 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");
```

Bind sheet to a JSON string

Refer to the below example code to bind a sheet to a JSON string.

C#

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

// Get data from json file.
string jsonText = string.Empty;
using (Stream stream = GetResourceStream("json\\DataBinding.json"))
using (StreamReader reader = new StreamReader(stream))
{
    jsonText = reader.ReadToEnd();
}

// jsonText content.
// [
//   {"Area": "North America","City": "Chicago","Category": "Consumer
Electronics","Name": "Bose 785593-0050","Revenue": 92800},
//   {"Area": "North America","City": "New York","Category": "Consumer
Electronics","Name": "Bose 785593-0050","Revenue": 92800},
//   {"Area": "South America","City": "Santiago","Category": "Consumer
Electronics","Name": "Bose 785593-0050","Revenue": 19550}
// ]

IWorksheet worksheet = workbook.ActiveSheet;

worksheet.DataSource = new JsonDataSource(jsonText);

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

}
```

Cell Binding

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

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, JSON string 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.

DsExcel.NET also provides **ITable.ExpandBoundRows** property to handle how a bound table should respond to the changes in data source. When the property is set to true, the bound table automatically adjusts the number of rows to accommodate data source changes. When this property is set to false(default), table behaves like Excel and only add or delete cells instead of entire rows to reflect changes of data source.

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";

// Let bound table expand to accomodate data source changes
table.ExpandBoundRows = true;

// table2 does not expand with data source
table2.ExpandBoundRows = false;

// 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

DsExcel 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.

DsExcel 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

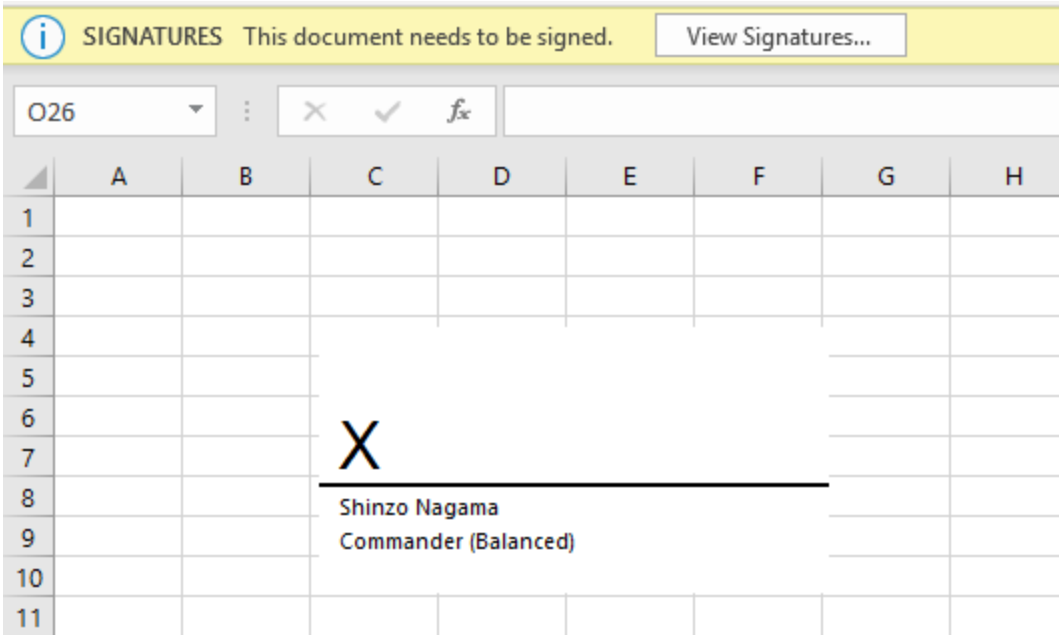
The **AddSignatureLine** method of **ISignatureSet** interface adds signature lines in a worksheet. You can also add information about the intended signer and instructions for the signer by using various properties of **ISignatureSetup** interface. When the workbook is opened again or sent to the intended 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 xlsm extension. A workbook containing digital signatures is 'marked as final' to discourage editing.

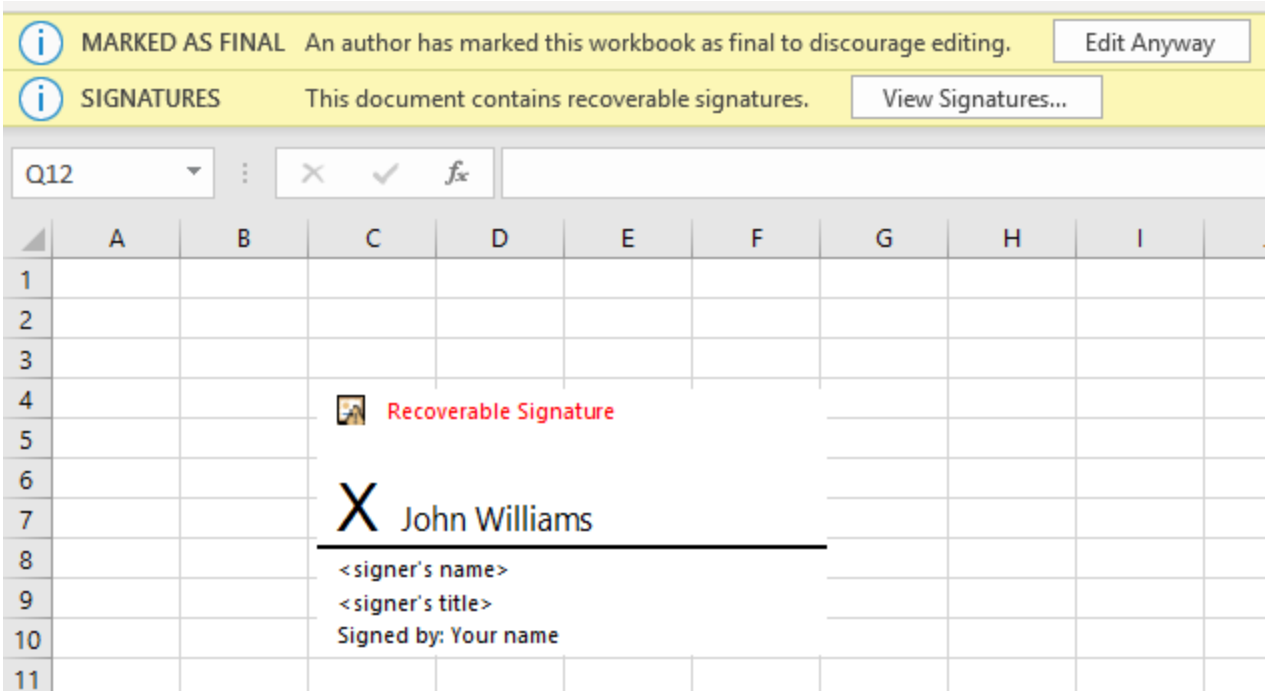
Refer to the following example code to add digital signatures in a worksheet.

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 DsExcel, 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 xlsm. 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

DsExcel 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

DsExcel 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

DsExcel .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 DsExcel .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 DsExcel .NET, managing formulas involves the following tasks.

- Formula Parser
- Formula Functions
- Set Formula to Range
- Set Table Formula
- Set Array Formula
- Dynamic Array Formulas
- Precedents and Dependents
- Iterative Calculation
- Calculation Mode
- Cross Workbook Formula
- Localized Formulas

Formula Parser

DsExcel 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 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 r1c1Option = new ParseContext { IsR1C1 = true };
var syntaxTree = FormulaSyntaxTree.Parse(r1c1Formula, r1c1Option);

// 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 = "'=" + r1c1Formula.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 parsed object for a workbook referenced from a web URI will look like below:

C#

```
var worbookRef = WorkbookReference.FromUri( "https://somesite.com/files/sample.xlsx");
var model = new ReferenceNode(
    new Reference
    {
        Workbook = worbookRef,
        WorksheetName = "sheet1",
        Row = 8,
        Column = 1,
    }
);
```

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!
- #CONNECT!
- #FIELD!
- #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

- GetHashCode method of FormulaSyntaxTree and SyntaxNode class are not supported. They return constant values because all fields are mutable.

- DsExcel does not support resolving workbook index defined in OpenXML or JSON file storage. They are treated as workbook reference by name.

Formula Functions

DsExcel.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 urity 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.
AGGREGATE	Math and Trigonometry	Returns an aggregate in a list or database.
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.
ASC	Text	Transforms full-width (double-byte) characters to half-width (single-byte) characters.
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 ß (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 In(x).
BESSELJ	Engineering	Returns the Bessel function Jn(x).
BESSELK	Engineering	Returns the modified Bessel function Kn(x).
BESSELY	Engineering	Returns the Bessel function Yn(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.
BYCOL	Logical	Applies a LAMBDA to each column and returns an array of the results.
BYROW	Logical	Applies a LAMBDA to each row and returns an array of the results.
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 a cell such as contents, formatting, location, etc. **Limitations** • If the argument reference is omitted, Excel uses the last modified cell reference, while DsExcel returns #VAULE!. • Since spreadjs does not support this function, json is not supported.
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.
CHOOSECOLS	Array Manipulation	Returns specified columns from array
CHOOSEROWS	Array Manipulation	Returns specified rows from array
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.
DBCS	Text	Transforms half-width (single-byte) characters to full-width (double-byte) characters.
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.
DROP	Array Manipulation	Drops rows or columns from the beginning or the end of the provided array
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.
EXPAND	Array Manipulation	Expands array to a specified dimension
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 trigonometry	Returns the double factorial of a number.
FALSE	Logical	Returns the logical value FALSE.
FDIST	Compatibility	Returns the F probability distribution.
FILTER	Lookup and reference	Filters a range of data based on criteria you define.
FILTERXML	Web	Returns specific data from the XML content using the specified XPath.
FIND	Text	Finds one text value within another (case-sensitive).

FINDB	Text	Finds one text value within another (case-sensitive). FINDB counts each double-byte character as 2 when you have enabled the editing of a language that supports DBCS and then set it as the default language. Otherwise, FINDB behaves the same as FIND, counting each character as 1.
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.
GETPIVOTDATA	Lookup and reference	Returns visible data from a pivot table.

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.
HSTACK	Array Manipulation	Stack arrays horizontally
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.

ISOMITTED	Information	Checks whether the value in a LAMBDA is missing and returns TRUE or FALSE.
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.
JIS	Text	Converts half-width (single-byte) letters within a character string to full-width (double-byte) characters.
KURT	Statistical	Returns TRUE if the value is text.
LAMBDA	Math and trigonometry	Returns custom reusable functions that can be called like any other function.
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.
LEFTB	Text	Returns the leftmost characters from a text value. LEFTB counts 2 bytes per character when a DBCS language is set as the default language. Otherwise behaves the same as LEFT, counting 1 byte per character.
LEN	Text	Returns the number of characters in a text string.
LENB	Text	Returns the number of characters in a text string. LENB counts 2 bytes per character when a DBCS language is set as the default language. Otherwise behaves the same as LEN, counting 1 byte per character.
LET	Logical	Assigns names to calculation results.
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.
MAKEARRAY	Logical	Returns a calculated array of a specified row and column size, by applying a LAMBDA.
MAP	Logical	Returns an array formed by mapping each value in the array(s) to a new value by applying a LAMBDA to create a new value.
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.
MIDB	Text	Returns a specific number of characters from a text string starting at the position you specify. MIDB counts each double-byte character as 2 when you have enabled the editing of a language that supports DBCS and then set it as the default language. Otherwise, MIDB behaves the same as MID, counting each character as 1.
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.
RANDARRAY	Lookup and reference	Returns an array of random numbers 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.
REDUCE	Logical	Reduces an array to an accumulated value by applying a LAMBDA to each value and returning the total value in the accumulator.

REPLACE	Text	Replaces characters within text.
REPLACEB	Text	Replaces characters within text. REPLACEB counts each double-byte character as 2 when you have enabled the editing of a language that supports DBCS and then set it as the default language. Otherwise, REPLACEB behaves the same as REPLACE, counting each character as 1.
REPT	Text	Repeats text a given number of times.
RIGHT	Text	Returns the rightmost characters from a text value.
RIGHTB	Text	Returns the rightmost characters from a text value. RIGHTB counts each double-byte character as 2 when you have enabled the editing of a language that supports DBCS and then set it as the default language. Otherwise, RIGHTB behaves the same as RIGHT, counting each character as 1.
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.
SCAN	Logical	Scans an array by applying a LAMBDA to each value and returns an array that has each intermediate value.
SEARCH	Text	Finds one text value within another (not case-sensitive).
SEARCHB	Text	Finds one text value within another (not case-sensitive). SEARCHB counts 2 bytes per character when a DBCS language is set as the default language. Otherwise behaves the same as SEARCH, counting 1 byte per character.
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.
SEQUENCE	Lookup and reference	Generates a list of sequential numbers in an array, such as 1, 2, 3, 4.
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.
SINGLE	Lookup and reference	Returns a single value, a single cell range or an error using the intersection logic.
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.
SORT	Lookup and reference	Sorts the contents of a range or array.
SORTBY	Lookup and reference	Sorts the contents of a range or array based on the values in a corresponding range or array.
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.
TAKE	Array Manipulation	Returns rows or columns of an array either from the beginning or the end of the provided array
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.
TEXTAFTER	Text Manipulation	Returns text present after the provided delimiting character
TEXTBEFORE	Text Manipulation	Returns text present before the provided delimiting character
TEXTSPLIT	Text Manipulation	Splits text between rows or columns using the delimiting character
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.
TOCOL	Array Manipulation	Returns the array as one column
TODAY	Date and time	Returns the serial number of today's date.
TOROW	Array Manipulation	Returns the array as one row
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 references by the given numeric value.

UNICODE	Text	Returns the number (code point) that corresponds to the first character of the text.
UNIQUE	Lookup and reference	Returns a list of unique values in a list or range.
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.
VSTACK	Array Manipulation	Stack arrays vertically
WEBSERVICE	Web	Returns data from a web service on the Internet or Intranet.
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.
WRAPCOLS	Array Manipulation	Wraps a column array in a 2D array
WRAPROWS	Array Manipulation	Wraps a row array in a 2D array
XIRR	Financial	Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.
XLOOKUP	Lookup and reference	Searches a range or an array, and then returns the item corresponding to the first match it finds. If no match exists, then XLOOKUP can return

		the closest (approximate) match.
XMATCH	Lookup and reference	Searches for a specified item in an array or range of cells, and then returns the item's relative position.
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. For more information on how to add custom names, see Defined Names.

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

DsExcel .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, DsExcel 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;
```

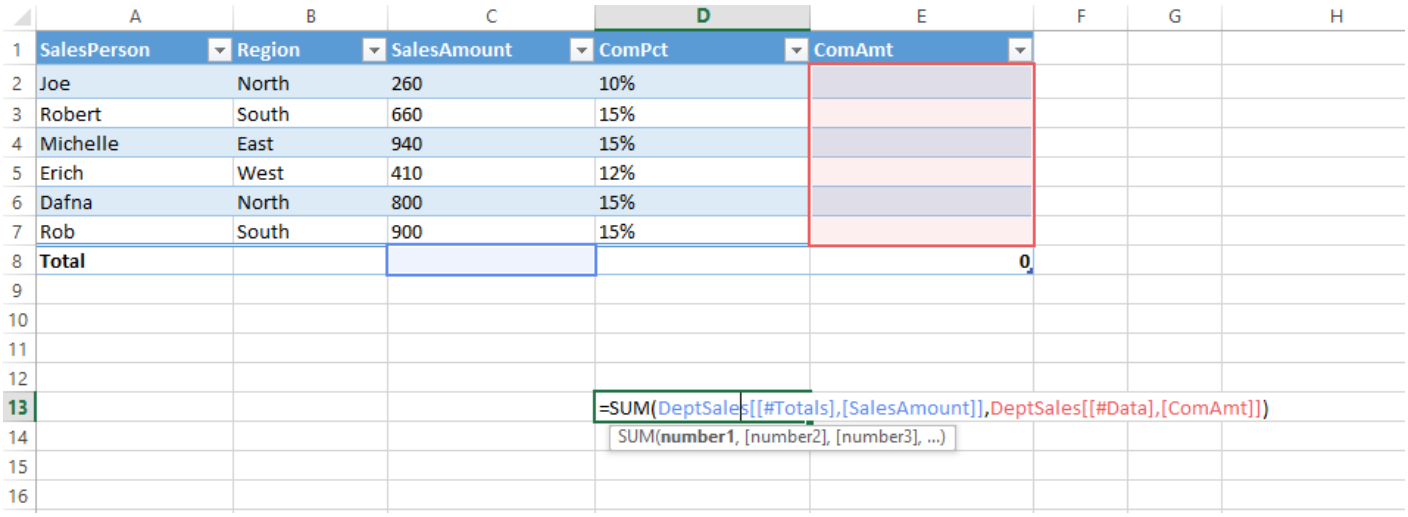
Limitation

When Workbook.DeferUpdateDirtyState is set to True, DsExcel 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.



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

Components	Description
Table Name	References the table data, without any header or total rows. You can use a default table name, such as Table1, or change it to use a custom name. Example: DeptSales is a custom table name in the table formula. For more information on how to add custom names, see Defined Names.
Column Specifier	Column specifiers use the names of the columns they represent. They reference column data without any column header or total row. Column specifiers must be enclosed in [] square brackets when they are written in the table formula. Example: [SalesAmount] and [ComAmt]
Item Specifier	Refers to a specific portions of the table such as total row. Example: [#Totals] and [#Data]
Table Specifier	Represents the outer portions of the structured reference. Outer references follow table names and are enclosed within the square brackets. Example: [[#Totals],[SalesAmount]],[[#Data],[ComAmt]]
Structures Reference	Represented by a string that begins with the table name and ends with the column specifier. Example: DeptSales[[#Totals],[SalesAmount]] and DeptSales[[#Data],[ComAmt]]

Reference operators

In DsExcel .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.

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

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

Refer to the following example code to set table formula in your spreadsheets.

C#

```
// Define Data
worksheet.Range["A1:E3"].Value = new object[,]
    {
        {"SalesPerson", "Region",  "SalesAmount", "ComPct", "ComAmt"},
        {"Joe", "North", 260, 0.10, null},
        {"Robert", "South", 660, 0.15, null},
    };

worksheet.Tables.Add(worksheet.Range["A1:E3"], true);
worksheet.Tables[0].Name = "DeptSales";
worksheet.Tables[0].Columns["ComPct"].DataBodyRange.NumberFormat = "0%";

//Use table formula in table range.
worksheet.Tables[0].Columns["ComAmt"].DataBodyRange.Formula = "=[@ComPct]*
[@SalesAmount]";

//Use table formula out of table range.
worksheet.Range["F2"].Formula = "=SUM(DeptSales[@SalesAmount])";
worksheet.Range["G2"].Formula = "=SUM(DeptSales[[#Data],[SalesAmount]])";
worksheet.Range["H2"].Formula = "=SUM(DeptSales[SalesAmount])";
worksheet.Range["I2"].Formula = "=SUM(DeptSales[@ComPct], DeptSales[@ComAmt])";
```

Set Array Formula

Array formula is a formula that can execute multiple calculations on individual cells or a range of cells to display a column or a row of subtotals. The array formula can consist of array of row of values, column of values or simply a combination of rows and columns of values that may return either multiple results or a single result.

Array formulas can be used to simplify the following tasks in a worksheet:

1. You can count the number of characters in a range of cells.
2. You can sum numeric values in cells that meet a specified criteria. For instance,the highest value in a range or values that fall between an upper and lower boundary.
3. You can sum every nth value in a range of cell values in a spreadsheet.

In DsExcel .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();
```

Dynamic Array Formulas

Dynamic Array Formulas are the formulas which return multiple values (in an array) to a range of cells on a worksheet. The neighbouring cells are hence populated with the results (calculated data) based on a single formula entered in one cell.

This behavior is called 'Spilling' and the range in which the results appear is called a 'Spill Range'. The spill range operator (#) can be used to reference the entire spill range.

DsExcel supports dynamic array formulas in worksheets by using the **IRange.Formula2** property which allows you to define dynamic array formula in a worksheet. It also lets you specify a formula without automatically adding the intersection operator (@). To enable use of the dynamic array formulas, you need to specify formula of IRange object through the Formula2 property.

> **Note:** In v5.0 release, the AllowDynamicArray property has been obsoleted. The property can currently be used along with IRange.Formula property to support compatibility with v4.2 version. However, we recommend using the new Formula2 property as the AllowDynamicArray property might be removed in future.

You can also use **CalcError** enumeration which specifies the type of calculation error:

- **Calc**: Occurs when calculation engine encounters a scenario it does not currently support.
- **Spill**: Occurs when a formula returns multiple results, but can't return these values to neighboring cells.

The below dynamic array functions are added in DsExcel:

Function	Category	Description
FILTER	Lookup and reference	Filters a range of data based on the defined criteria
RANDARRAY	Math and trigonometry	Returns an array of random numbers between 0 and 1
SEQUENCE	Math and trigonometry	Generatesa list of sequential numbers in an array, such as 1, 2, 3, 4
SINGLE	Lookup and reference	Returns a single value using logic known as implicit intersection
SORT	Lookup and reference	Sorts the contents of a range or array
SORTBY	Lookup and reference	Sorts the contents of a range or array based on the values in a corresponding range or array
UNIQUE	Lookup and reference	Returns a list of unique values in a list or range

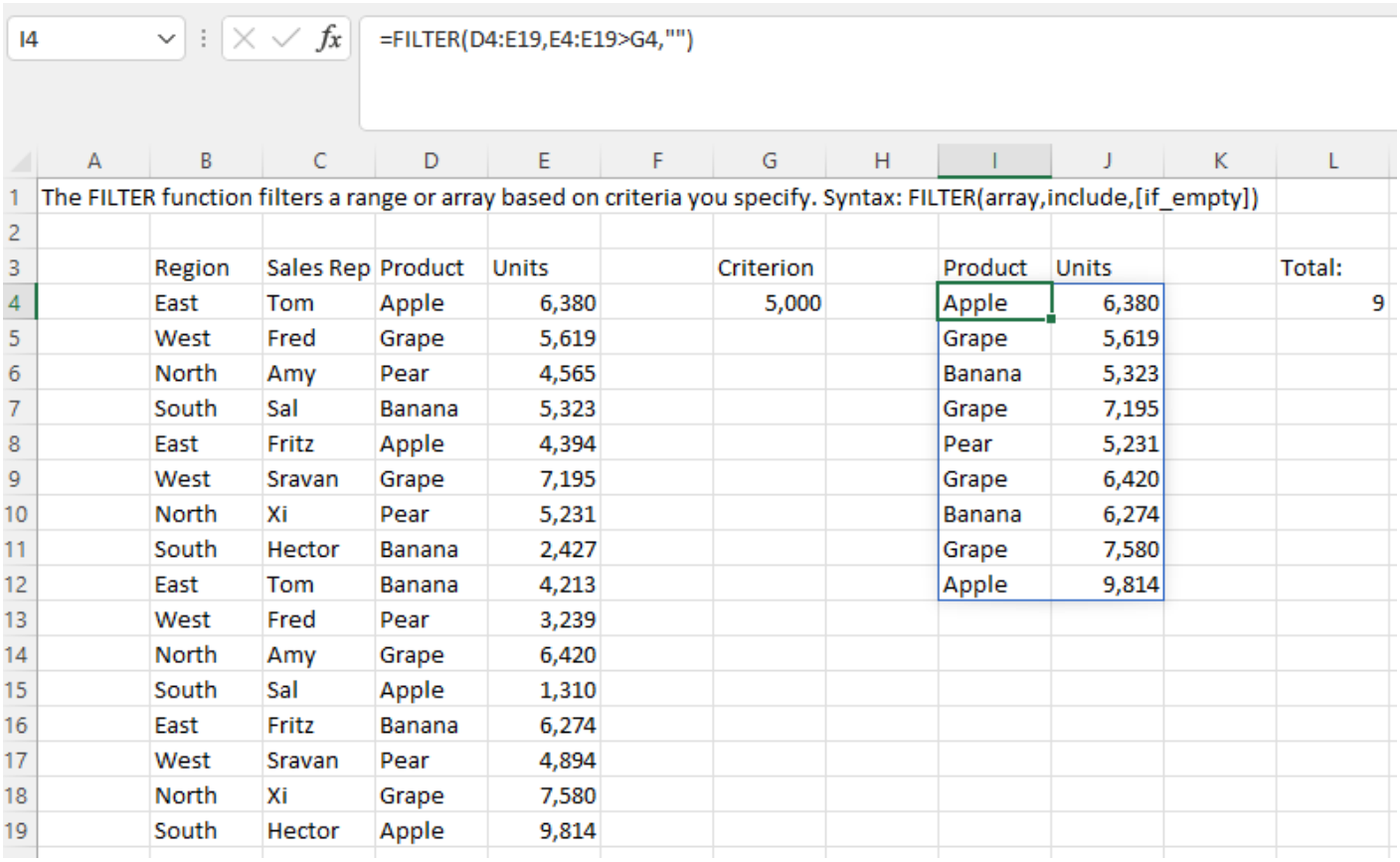
Refer to the following example code to enable dynamic array formula and use FILTER function by specifying a criteria

C#

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

var sheet = workbook.Worksheets[0];
sheet.Name = "FILTER";
sheet.Range["A1"].Value = "The FILTER function filters a range or array based on
criteria you specify. Syntax: FILTER(array,include,[if_empty])";
```

```
sheet.Range["B3:E19"].Value = new object[,] {
    { "Region", "Sales Rep", "Product", "Units" },
    { "East", "Tom", "Apple", 6380 },
    { "West", "Fred", "Grape", 5619 },
    { "North ", "Amy", "Pear", 4565 },
    { "South", "Sal", "Banana", 5323 },
    { "East", "Fritz", "Apple", 4394 },
    { "West", "Sravan", "Grape", 7195 },
    { "North ", "Xi", "Pear", 5231 },
    { "South", "Hector", "Banana", 2427 },
    { "East", "Tom", "Banana", 4213 },
    { "West", "Fred", "Pear", 3239 },
    { "North ", "Amy", "Grape", 6420 },
    { "South", "Sal", "Apple", 1310 },
    { "East", "Fritz", "Banana", 6274 },
    { "West", "Sravan", "Pear", 4894 },
    { "North ", "Xi", "Grape", 7580 },
    { "South", "Hector", "Apple", 9814 } };

sheet.Range["G3:L4"].Value = new object[,] { { "Criterion", "", "Product", "Units", "",
"Total:" }, { 5000, null, null,null,null,null } };

sheet.Range["I4"].Formula2 = "=FILTER(D4:E19,E4:E19>G4,\"\")";
sheet.Range["L4"].Formula2 = "=SUM(IF(E4:E19>G4,1,0))";

sheet.Range["E4:E19,G4,J4:J12"].NumberFormat = "#,##0";

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

The below image shows the output of above code where Filter function is applied in cell I4.



	A	B	C	D	E	F	G	H	I	J	K	L
1	The FILTER function filters a range or array based on criteria you specify. Syntax: FILTER(array,include,[if_empty])											
2												
3		Region	Sales Rep	Product	Units		Criterion		Product	Units		Total:
4		East	Tom	Apple	6,380		5,000		Apple	6,380		9
5		West	Fred	Grape	5,619				Grape	5,619		
6		North	Amy	Pear	4,565				Banana	5,323		
7		South	Sal	Banana	5,323				Grape	7,195		
8		East	Fritz	Apple	4,394				Pear	5,231		
9		West	Sravan	Grape	7,195				Grape	6,420		
10		North	Xi	Pear	5,231				Banana	6,274		
11		South	Hector	Banana	2,427				Grape	7,580		
12		East	Tom	Banana	4,213				Apple	9,814		
13		West	Fred	Pear	3,239							
14		North	Amy	Grape	6,420							
15		South	Sal	Apple	1,310							
16		East	Fritz	Banana	6,274							
17		West	Sravan	Pear	4,894							
18		North	Xi	Grape	7,580							
19		South	Hector	Apple	9,814							

Note: In addition, DsExcel also supports Array and Text manipulation functions. For more information, see Formula Functions.

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. DsExcel 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 directly or indirectly referred to, by the formulas in other cells
- **Dependents**: Cells or ranges which contain formulas that refer to other cells directly or indirectly

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.

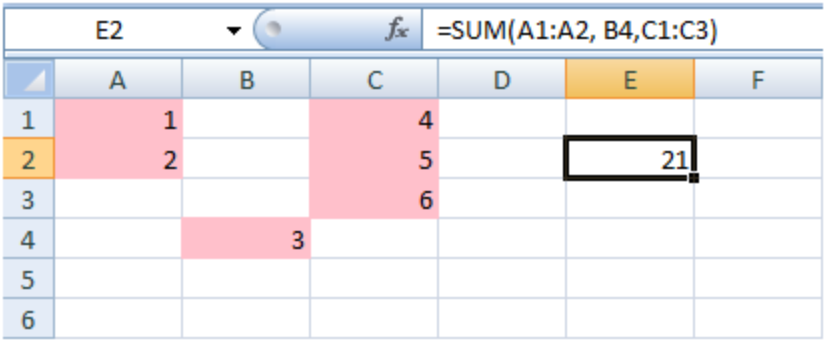
Direct Precedents

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

C#

```
public void DirectPrecedents()
{
    // 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 direct precedent ranges (highlighted in pink).



Direct Dependents

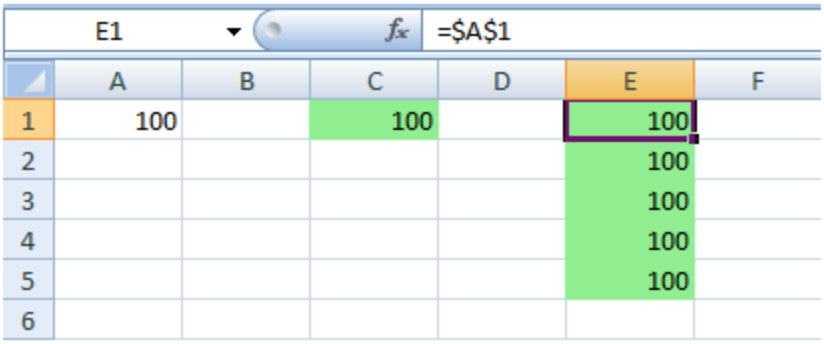
Refer to the following example code to get direct dependent ranges in a worksheet.

C#

```
public void DirectDependents()
```

```
{
    // 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).



Direct and Indirect Precedents

You can also identify the direct and indirect precedents by using the overloaded **GetPrecedents** method which provide the **includeIndirect** parameter. This parameter when set to true returns all the direct and indirect precedents. However, its default value is false which returns only direct precedents.

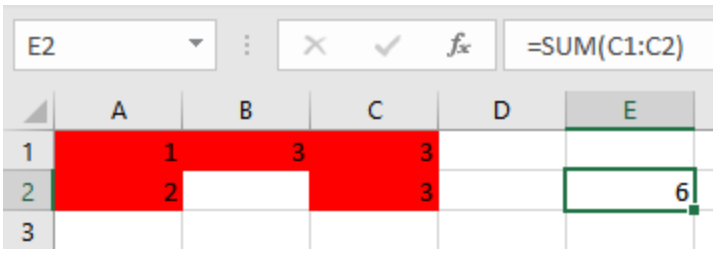
Refer to the following example code to get all the precedent ranges in a worksheet.

C#

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

```
        IWorksheet worksheet0 = workbook.Worksheets[0];
        // Set Formula in Cells
        worksheet0.Range["E2"].Formula = "=Sum(C1:C2)";
        worksheet0.Range["C1"].Formula = "=B1";
        worksheet0.Range["B1"].Formula = "=Sum(A1:A2)";
        // Set Value of Cells
        worksheet0.Range["A1"].Value = 1;
        worksheet0.Range["A2"].Value = 2;
        worksheet0.Range["C2"].Value = 3;

        List<IRange> list = new List<IRange>();
        foreach (var item in worksheet0.Range["E2"].GetPrecedents(true))
        {
            item.Interior.Color = Color.Red;
        }

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

The below image shows all the precedent ranges of cell E2.



Direct and Indirect Dependents

You can also identify the direct and indirect dependentsby using the overloaded **GetDependents** method which provide the **includeIndirect** parameter. This parameter when set to true returns all the direct and indirect dependents. However, its default value is false which returns only direct dependents.

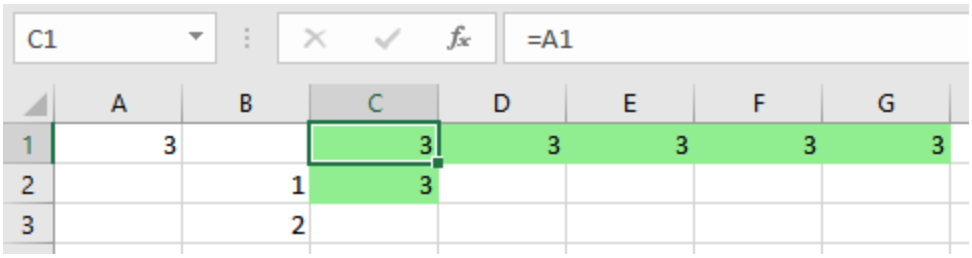
Refer to the following example code to get all the dependent ranges in a worksheet.

C#

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

    worksheet1.Range["C1"].Formula = "A1";
    worksheet1.Range["C2"].Formula = "A1";
    worksheet1.Range["D1"].Formula = "A1";
    worksheet1.Range["F1"].Formula = "A1";
```

```
    worksheet1.Range["G1"].Formula = "A1";
    worksheet1.Range["E1"].Formula = "D1";
    // Set Value of Cells
    worksheet1.Range["B2"].Value = 1;
    worksheet1.Range["B3"].Value = 2;

    foreach (var item in worksheet1.Range["A1"].GetDependents(true))
    {
        item.Interior.Color = Color.LightGreen;
    }
    //save to an excel file
    workbook.Save("getAllDependents.xlsx");
}
```

The below image shows all the dependent ranges of cell A1.



Iterative Calculation

Iterative calculation is performed to repeatedly calculate a function until a specific numeric condition is met. DsExcel 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, DsExcel 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");
```

Calculation Mode

Sometimes, opening a large workbook with many formulas can take a long time, as Excel recalculates all formulas before opening the workbook, which leads to a longer processing time. Moreover, when exporting particular worksheets containing formulas or cross-worksheet formulas, Excel requires significant time because it calculates all formulas before completing the export process. To enhance the speed of opening or exporting a large Excel workbook with extensive formulas, DsExcel provides **CalculationMode** property within the **IFormulaOptions** interface. This property allows you to choose from the **CalculationMode** enumeration options, providing control over how Excel calculates formulas before the workbook is opened or exported. CalculationMode enumeration provides the following three calculation modes:

Calculation Mode	Description
Automatic	In this mode, Excel calculates everything and recalculates whenever something is changed every time a workbook is opened.
Semiautomatic	In this mode, Excel calculates everything except Data Tables and Python formulas.
Manual	In this mode, Excel calculates nothing; it recalculates only when the user explicitly requests it by pressing F9 or CTRL+ALT+F9 or when the workbook is saved.

CalculationMode property does not impact the functioning of the internal calculation engine of DsExcel, and it only affects the calculation mode settings in Excel and SpreadJS I/O. This property does not affect the runtime state of DsExcel. If you want to disable the calculation for the current workbook, use **EnableCalculation** property.

Refer to the following example code to set the CalculationMode to 'Manual' for calculating the 'Total' value:

C#

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

// Add data for the table.
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}
};
```

```
// Add data to the range.
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A1:F7"].Value = data;
worksheet.Range["A:F"].ColumnWidth = 15;

// Add table.
worksheet.Tables.Add(worksheet.Range["A1:F7"], true);

// Show totals.
worksheet.Tables[0].ShowTotals = true;
worksheet.Tables[0].Columns[4].TotalsCalculation = TotalsCalculation.Average;
worksheet.Tables[0].Columns[5].TotalsCalculation = TotalsCalculation.Average;

// Add comment to notify the user to calculate the formula manually.
var comment = worksheet.Range["F8"].AddComment("Please press F9 to calculate the
formula.");
comment.Visible = true;

// Set calculation mode to manual.
workbook.Options.Formulas.CalculationMode = CalculationMode.Manual;

// Save the Excel file.
workbook.Save("CalculationModeOptions.xlsx");
```

	A	B	C	D	E	F	G	H	I
1	Name	City	Birthday	Eye color	Weight	Height			
2	Richard	New York	08-06-1968	Blue	67	165			
3	Nia	New York	03-07-1972	Brown	62	134			
4	Jared	New York	02-03-1964	Hazel	72	180			
5	Natalie	Washington	08-08-1972	Blue	66	163			
6	Damon	Washington	02-02-1986	Hazel	76	176			
7	Angela	Washington	15-02-1993	Brown	68	145			
8	Total								
9									
10									
11									

Please press F9 to calculate the formula.

Note: SpreadJS does not support "Partial" calculations. When exporting SSJSON and SJS files, it will be considered "Automatic."

Cross Workbook Formula

Cross-workbook formulas allow you to calculate values by referring to and using data from different workbooks. For example, if there are five workbooks for different subjects, you can add the marks of all five subjects to a worksheet by using cross-workbook formulas.

DsExcel supports using cross-workbook formulas by using the folder or web path for an external workbook. The **GetExcelLinkSources** method 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 the cross-workbook formula by using the folder path for the external workbook and updating 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("crossworkbookformulafolderpath.xlsx");
```

Refer to the following example code to use the cross-workbook formula by using the web path for an external workbook and updating the Excel links:

C#

```
//create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();
workbook.Worksheets[0].Range["B1"].Formula = "=
'https://developer.mescius.com/dsexcel/[SourceWorkbook.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 = 100;
// update the caches of external workbook data.
foreach (var item in workbook.GetExcelLinkSources())
{
    workbook.UpdateExcelLink(item, workbook2);
}
//save to an excel file
workbook.Save("crossworkbookformulawebpath.xlsx");
```

Refer to the following example code to use the cross-workbook formula by using the path for an external workbook with the table and updating the Excel links:

C#

```
// Create a new workbook.
```

```
var workbook = new Workbook();

// Add data to the cell.
workbook.Worksheets[0].Range["A1"].Value = "Total Sales";

// Set the table formula across workbook.
workbook.Worksheets[0].Range["B1"].Formula = "=SUM('[SalesTable.xlsx]'!Table1[Sales])";
workbook.Worksheets[0].Range["B1"].NumberFormat = "$#,##0.00";
workbook.Worksheets[0].Range["A:B"].ColumnWidth = 12;

// Create another new workbook as the intance of external workbook.
var workbook2 = new Workbook();

// Add data for the table.
workbook2.ActiveSheet.Range["A1:C6"].Value = new object[,]
{
{ "Product", "Type", "Sales" },
{ "Apple", "Fruit", 25000},
{ "Grape", "Fruit", 30000 },
{ "Carrot", "Vegetable", 28000 },
{ "Strawberry", "Fruit", 50000 },
{ "Onion", "Vegetable", 23000 }

};

// Create the table.
workbook2.ActiveSheet.Tables.Add(workbook2.ActiveSheet.Range["A1:C6"], true);
workbook2.ActiveSheet.Range["C2:C6"].NumberFormat = "$#,##0.00";
workbook2.ActiveSheet.Range["A:C"].ColumnWidthInPixel = 100;

// Update the caches of external workbook data.
foreach (var item in workbook.GetExcelLinkSources())
{
    workbook.UpdateExcelLink(item, workbook2);
}

// Save both Excel files.
workbook.Save("CrossWorkbookTableFormula.xlsx");
workbook2.Save("SalesTable.xlsx");
```

Note: You need to open the external link's workbook at the same time you open the workbook where the cross-workbook formula is used to recalculate and show the correct result.

Localized Formulas

DsExcel provides the **FormulaLocal** and **FormulaR1C1Local** properties in **IRange** interface which can be used to retrieve or set localized formulas in the cells of a worksheet. For example, if you set a function in English Excel by using these

properties and you open the Excel in Japanese culture, the equivalent localized formula will be used to display the result in Japanese Excel.

> **Note:** These properties support localized formulas only for Japanese and Chinese cultures.

Refer to the following example code which sets the workbook culture as Japanese, sets ASC and JIS functions to **FormulaLocal** properties and further retrieves the general and localized function names.

C#

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

workbook.Culture = CultureInfo.GetCultureInfo("ja-JP");
var sheet = workbook.ActiveSheet;

sheet.Range["$A$1:$A$3"].Value = new[,]
{
{"Original"},
{"ゴールドシップは1番人気です。"},
{"Halfwidth"}
};

sheet.Range["A5"].Value = "Fullwidth";

var a4 = sheet.Range["A4"];
var a6 = sheet.Range["A6"];

// Equivalent: Formula = "ASC(A2)"
// Because ASC doesn't have localized name in Japanese Excel.
a4.FormulaLocal = "=ASC(A2)";

// Equivalent: Formula = "DBCS(A2)"
// Because JIS is localized name of DBCS in Japanese Excel.
a6.FormulaLocal = "=JIS(A2)";

// Compare different formula properties.
sheet.Range["$B$1:$F$1"].Value = new[,]
{
{nameof(IRange.FormulaLocal), nameof(IRange.Formula),
nameof(IRange.FormulaR1C1Local), nameof(IRange.FormulaR1C1)}
};

sheet.Range["$B$4:$E$4"].Value = new[,]
{
{a4.FormulaLocal, a4.Formula, a4.FormulaR1C1Local, a4.FormulaR1C1}
};

sheet.Range["$B$6:$E$6"].Value = new[,]
```

```
{
{a6.FormulaLocal, a6.Formula, a6.FormulaR1C1Local, a6.FormulaR1C1}
};

// Arrange layout
sheet.UsedRange.Columns.AutoFit();
sheet.PageSetup.IsPercentScale = false;
sheet.PageSetup.FitToPagesWide = 1;
sheet.PageSetup.PrintHeadings = true;

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

The below image shows the output of above code:

	A	B	C	D	E
1	Original	FormulaLocal	Formula	FormulaR1C1Local	FormulaR1C1
2	ゴールドシップは1番人気です。				
3	Halfwidth				
4	ｺﾞｰﾙﾄﾞｼｯﾌﾟは1番人気です。	=ASC(A2)	=ASC(A2)	=ASC(R[-2]C)	=ASC(R[-2]C)
5	Fullwidth				
6	ゴールドシップは１番人気です。	=JIS(A2)	=DBCS(A2)	=JIS(R[-4]C)	=DBCS(R[-4]C)

Note: Excel does not support multi-byte string conversions. Hence, the output is saved to a PDF file.

Custom Functions

DsExcel 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 DsExcel .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 **Example 5** of 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.

Caching in Custom Functions

Custom functions in the same column store resultant value as cache. Hence, when a custom function in a column is called

subsequently with previous parameter, custom function uses the cached value instead of calculating it again. This feature helps in optimizing performance especially in case of repetitive use of the custom function in a single column.

However, to control this caching behavior of custom functions, DsExcel.Net provides **IsVolatile** property in the class inherited from **CustomFunction** class. The property lets you choose whether to recalculate a custom function for a column having same parameters every time or use the cached result. The default value of this property is **false**, which means custom function applied on a single column maintains its own cache and reuses it on a repeated call. For implementation, see **Example 6: Create Volatile Cache**.

Create Custom Function Using Code

Creating custom function in DsExcel.Net involves following three steps.

- 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\",
\" \", \"Excel\", \".\")";
worksheet.Range["A2"].Formula = "=MyConcatenate(A1, \"Documents.\")";
// Value of cell A1 is "I work with Excel."
var resultA1 = worksheet.Range["A1"].Value;
// Display result in cell C1
worksheet.Range["C1"].Value = resultA1;
// Value of cell A2 is "I work with Excel 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");
```

Example 6: Create Volatile Custom Function

Following example demonstrates how to create a custom function for generating GUID. To generate a unique GUID every time, custom function should not be using cache. Hence, example code sets the **IsVolatile** property to **true**, so that a new GUID is generated on every call.

C#

```
// Creating a new class GeneralID by inheriting the CustomFunction class
internal class GeneralID : CustomFunction
{
    public GeneralID() : base("GeneralID", FunctionValueType.Object)
    {
        // set IsVolatile to true to avoid using cached value
        this.IsVolatile = true;
    }

    public override object Evaluate(object[] arguments, ICalcContext context)
    {
        return Guid.NewGuid().ToString("N");
    }
```

```
}
```

C#

```
// Step2: Register the custom function using AddCustomFunction() method
//create a new workbook
var workbook = new GrapeCity.Documents.Excel.Workbook();

GrapeCity.Documents.Excel.Workbook.AddCustomFunction(new GeneralID());

IWorksheet worksheet = workbook.Worksheets[0];

// Step3- Implement the Custom Function
worksheet.Range["A1"].Formula = "GeneralID()";
var valueA1Before = worksheet.Range["A1"].Value;

Console.WriteLine(valueA1Before);
worksheet.Range["A2"].Formula = "GeneralID()";

// Observe A1's value has changed because it's not using cached value.
var valueA1After = worksheet.Range["A1"].Value;
Console.WriteLine(valueA1After);
```

Example 7: Create Asynchronous Custom Functions

Asynchronous function is any function that delivers its result asynchronously or concurrently. Asynchronous functions have a non-blocking architecture, so the execution of one task isn't dependent on another. Tasks can run simultaneously. Running asynchronous functions can improve performance by allowing several calculations to run at the same time. DsExcel enables functions to perform asynchronous calculations by deriving them from **AsyncCustomFunction** class. **EvaluateAsync** method calculates the function asynchronously. DsExcel also provides an enumeration value "Busy" in **CalcError** enumeration that indicates that a cell is calculating an async formula.

Refer to the following example code to add and use a custom Asynchronous function:

C#

```
internal class Program
{
    static void Main(string[] args)
    {
        // Register Async custom function.
        Workbook.AddCustomFunction(new MyAddFunction());

        // Implement the Async custom Function.
        Workbook workbook = new Workbook();
        var worksheet = workbook.Worksheets[0];
        worksheet.Range["A1"].Value = 1;
        worksheet.Range["B1"].Value = 2;

        // Add the cell values.
```

```
        worksheet.Range["C1"].Formula = "=MyAdd(A1,B1)";
        var value = worksheet.Range["C1"].Value;

        // Display result. The result will be "Busy".
        Console.WriteLine($"get value first time:{value}");
        Thread.Sleep(2000);
        value = worksheet.Range["C1"].Value;

        // Display result. The result will be "3".
        Console.WriteLine($"get value second time:{value}");
    }
}
// Define Async custom function: MyAddFunction.
public sealed class MyAddFunction : AsyncCustomFunction
{
    public MyAddFunction()
        : base("MyAdd", FunctionValueType.Number, new Parameter[] { new
Parameter(FunctionValueType.Number), new Parameter(FunctionValueType.Number) })
    {

    }

    async protected override Task<object> EvaluateAsync(object[] arguments, ICalcContext
context)
    {
        await Task.Delay(1000);
        return (double)arguments[0] + (double)arguments[1];
    }
}
```

Limitations

The AsyncCustomFunction's parameters do not accept any reference because the asynchronous function may run in another thread, and it will cause multi-thread conflicts if you use a reference. Similarly, using objects such as IWorksheet and IWorkbook is not allowed within asynchronous functions.

Shapes and Pictures

DsExcel .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 DsExcel 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 DsExcel, you can add connectors at specific coordinates or a specific range of a worksheet using the **AddConnector** method. You can also 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 code to connect general shapes using the connector format. You can add a Connector by providing the connector position in points, or add a connector directly to a range.

C#

```
// To config the connector shape.
// Add shapes using points
IShape shapeBegin = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 1, 1, 100, 100);
IShape endBegin = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 200, 200, 100,
100);

//Add connector for shapeBegin and endShape, by providing connector position in points
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);

// Add shape using range
IShape rectangle3 = worksheet.Shapes.AddShape(AutoShapeType.Rectangle,
worksheet.Range["B12"]);
IShape rectangle4 = worksheet.Shapes.AddShape(AutoShapeType.Rectangle,
worksheet.Range["D12"]);

//Add connector for rectangle3 and rectangle4, by adding connector directly to a range
IShape rangeConnectorShape = worksheet.Shapes.AddConnector(ConnectorType.Curve,
worksheet.Range["B12:D12"]);
```

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 DsExcel, 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.

To add shapes in a DsExcel worksheet, you can use **AddShape** method of the **IShapes** interface. The method provides overloads which allow you to add variety of shapes at a specified position or to a specified range.

Refer to the below example code to add shapes at a particular position and to a specific range:

C#

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

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

// Create shape with custom name at a specific position
IShape shape = worksheet.Shapes.AddShape("Balloon", AutoShapeType.Balloon, 50, 50, 100,
200);

// Add shape to a range
// IShape rangeShape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle,
worksheet.Range["F5:I10"]);

// 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 **AddPicture** method of the **IShapes** interface. The method allows you to add a picture at a specific location or to a specific range. The **IPictureFormat interface** in DsExcel allows users to customize and format pictures while working in a spreadsheet.

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

C#

```
// Add a picture through stream
string path = @"Images\flower.jpg";
FileStream stream = System.IO.File.Open(path, FileMode.Open);
IShape picture = worksheet.Shapes.AddPicture(stream, ImageType.JPG, 480, 10, 100, 100);

// Add a picture through file at specific location
// IShape picture = worksheet.Shapes.AddPicture(@"Images\flower.jpg", 480, 10, 100,
100);

// Add a picture to a specific range
// IShape pictureInRange = worksheet.Shapes.AddPicture("flower.jpg",
worksheet.Range["D3:F5"]);

// 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");
```

To view the code in action, see Add shape to range and Add picture to range demos.

Working with shapes and pictures in the DsExcel 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

Linked Picture

> **Note**:
>
> - DsExcel.NET also provides support for loading and saving SpreadJS JSON files with shapes. For more information, refer to Import and Export JSON Stream.
> - The targetRange and the shape to be added must exist in the same worksheet. Otherwise, it results into an InvalidOperationException.

Customize Shape Format and Shape Text

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

Shape Format

In DsExcel, 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 DsExcel. 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

DsExcel 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 arround 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;
```

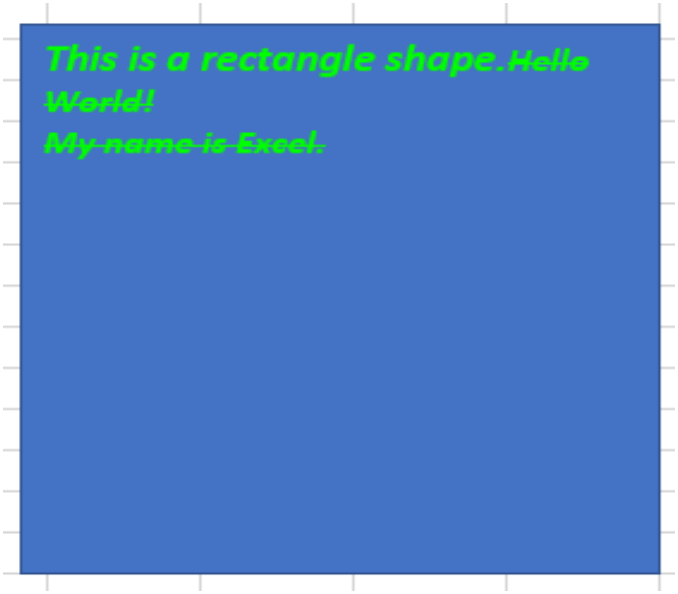
Shape Text

In DsExcel, 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.

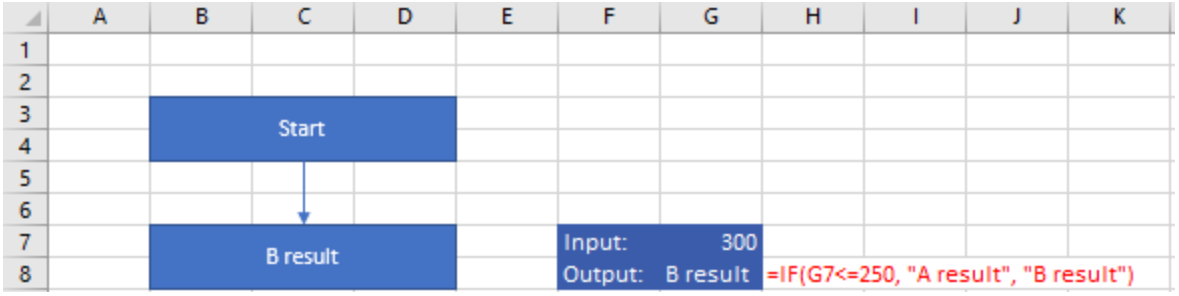
C#

```
// To config shape's text and text style.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 40, 40, 200, 200);
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 = 12;
shape.TextFrame.TextRange.Font.Strikethrough = true;

shape.TextFrame.TextRange.Paragraphs.Add("This is a rectangle shape.");
shape.TextFrame.TextRange.Paragraphs.Add("My name is Excel.");
shape.TextFrame.TextRange.Paragraphs[0].Runs.Add("Hello World!");

shape.TextFrame.TextRange.Paragraphs[0].Runs[0].Font.Strikethrough = false;
shape.TextFrame.TextRange.Paragraphs[0].Runs[0].Font.Size = 14;
```



Set Formula for Shape Text

You can set formula for a shape by using **Formula** property of the **IShape** interface. This property configures a formula that refers to text of the range or a defined name. When you set a shape formula for the first time, the shape acquires text and font style of the first cell of the reference. Once shape text has been set, any kind of changes in content of the referenced cell updates value of the shape text also. However, font style remains the same.



C#

```
// set shape formula to G8
IShape shapeResult = worksheet.Shapes.AddShape(AutoShapeType.Rectangle,
worksheet.Range["B7:D8"]);
shapeResult.Formula = "=G8";
```

You can remove shape reference by setting the **Formula** property to **null**. On removing reference, the shape text becomes a custom normal text; shape content gets text of the first cell of removed reference and the font style is the default style. If shape text is removed from the shape, cell reference stops having any affect on the shape text.

Further, you can also retain formula of the referenced shape when exporting to JSON IO, DsExcel API, PDF, HTML, or an image.

To view the feature in action, see Set Shape Formula demo.

Set Alignment of Shape Text

You can align the text in a shape to the left, right, center, distribute, and justify using the **TextAlignment** property. Also, you can secure the position of the text frame containing the text at the center using the **HorizontalAnchor** property and at the top, middle, and bottom using the **VerticalAnchor** property.

These different alignments and positions of text in a shape can also be exported to PDF documents.

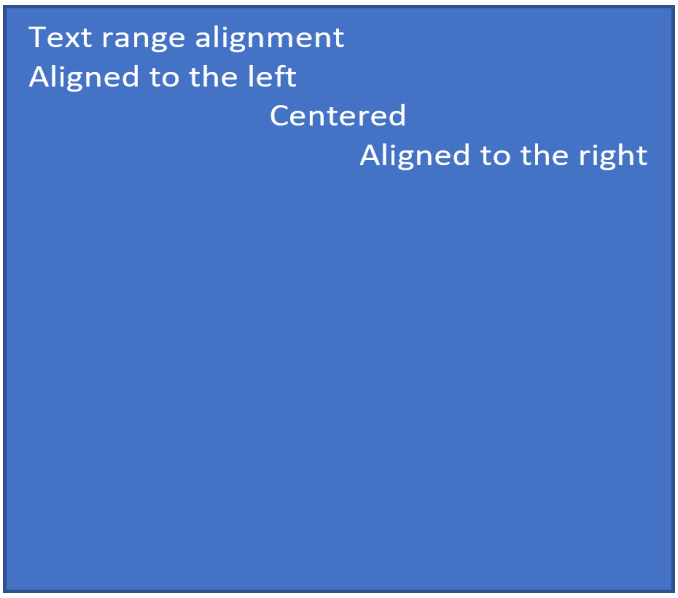
Align Text

The **TextAlignment** property in **ITextRange** interface allows you to set the alignment of a text range or a paragraph in a shape using **TextAlignmentAnchor** enumeration. This property sets the text alignment to left, right, center, distribute, and justify.

Refer to the following example code to set the alignment of text range and paragraphs in a shape:

C#

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

// Add a shape.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 10, 10, 200, 200);

// Add text range and two paragraphs for the shape.
shape.TextFrame.TextRange.Text = "Text range alignment";
shape.TextFrame.TextRange.Paragraphs.Add("Aligned to the left");
shape.TextFrame.TextRange.Paragraphs.Add("Centered");
shape.TextFrame.TextRange.Paragraphs.Add("Aligned to the right");

// Align text range to the left.
shape.TextFrame.TextRange.TextAlignment = TextAlignmentAnchor.Left;

// Align paragraph to the center.
shape.TextFrame.TextRange.Paragraphs[2].TextAlignment = TextAlignmentAnchor.Center;

// Align paragraph to the right.
shape.TextFrame.TextRange.Paragraphs[3].TextAlignment = TextAlignmentAnchor.Right;

// Save the workbook in XLSX and PDF formats.
workbook.Save("Alignment.xlsx");
workbook.Save("Alignment.pdf");
```



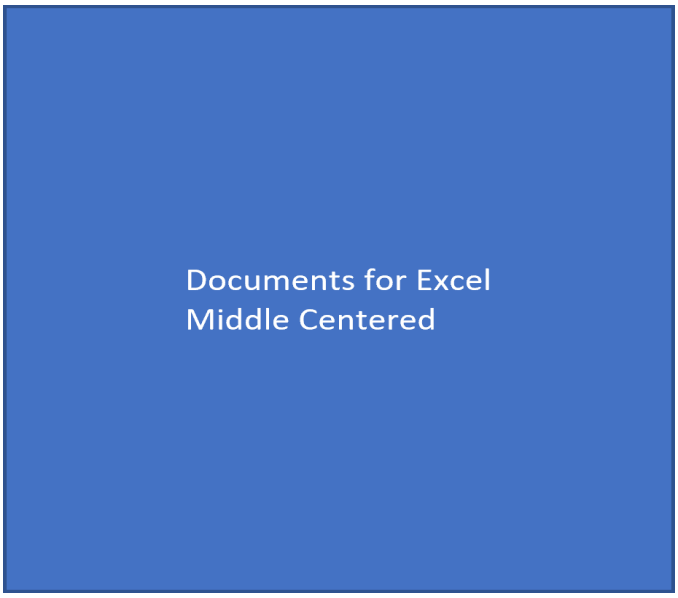
Anchor Text

The text frame (or text body) contains the text or paragraph you add to a shape. The **HorizontalAnchor** and **VerticalAnchor** properties of **ITextFrame** interface allow you to set the horizontal and vertical anchors of a text frame in a shape using **HorizontalAnchor** and **VerticalAnchor** enumerations. The text frame can be positioned horizontally at the center or vertically at the top, middle, or bottom.

Refer to the following example code to anchor the text frame in a shape:

C#

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

//Add a shape.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 10, 10, 200, 200);

//Add two paragraphs for the shape.
shape.TextFrame.TextRange.Paragraphs.Add("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");
```



You can also set the alignment of a text range and a paragraph, along with the anchor of the text frame in a shape. Refer to the following example code to align and anchor a paragraph at the bottom right:

C#

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

// Add a shape.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 10, 10, 400, 200);

// Add a paragraph for the shape.
shape.TextFrame.TextRange.Paragraphs.Add("Aligned and anchored to bottom right");

// Anchor the text frame to the bottom vertically.
shape.TextFrame.VerticalAnchor = VerticalAnchor.AnchorBottom;

// Align paragraph to the right.
shape.TextFrame.TextRange.Paragraphs[0].TextAlignment = TextAlignmentAnchor.Right;

// Save the workbook in XLSX and PDF formats.
workbook.Save("Alignment.xlsx");
workbook.Save("Alignment.pdf");
```

Note: The **TextAlignmentAnchor.Mixed** is a special enumeration value returned for a shape having different alignments applied to paragraphs in a text range. If you set the alignment of a text or paragraph in Shape using TextAlignment.Mixed, this will throw an exception.

Set Direction of Shape Text

You can set the direction of the text in shape to horizontal, vertical, rotated (to 90 or 270 degree), and stacked (with text reading left-to-right or right to left). The **Direction** property in **ITextFrame** interface allows you to set the direction of the text frame in shape using **TextDirection** enumeration.

Refer to the following example code to set the text direction to vertical:

C#

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

// Add a shape.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 10, 10, 200, 200);

// Add paragraph for the shape.
shape.TextFrame.TextRange.Paragraphs.Add("Documents for Excel");

// Set the direction of text frame to vertical.
shape.TextFrame.Direction = TextDirection.Vertical;

// Save the workbook.
workbook.Save("TextDirection.xlsx");
```

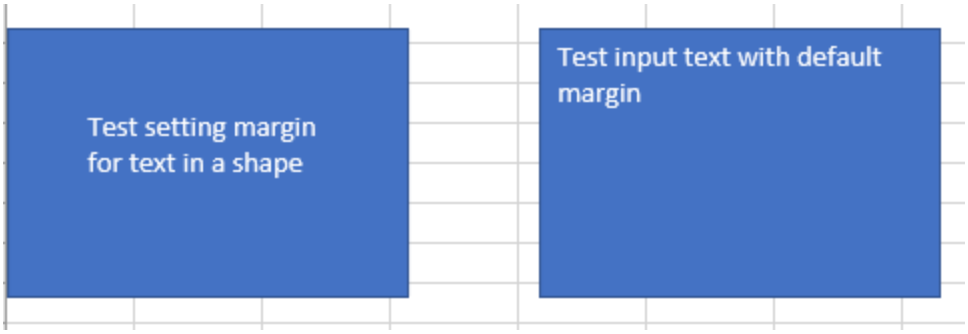
Set Margin for Shape Text

You can set the margin of text in a shape in the bottom, left, right and top directions. The **MarginBottom**, **MarginLeft**, **MarginRight** and **MarginTop** properties of ITextFrame interface can be used to achieve the same.

Refer to the following example code which configures the text margins in first shape and keeps it as default in the other.

C#

```
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 1, 10, 150, 100);
IShape shape2 = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 200, 10, 150, 100);

//set the margin of text
shape.TextFrame.MarginBottom = 30;
shape.TextFrame.MarginLeft = 30;
shape.TextFrame.MarginRight = 30;
shape.TextFrame.MarginTop = 30;

shape.TextFrame.TextRange.Paragraphs[0].Runs.Add("Test setting margin for text in a
shape");
shape2.TextFrame.TextRange.Paragraphs[0].Runs.Add("Test input text with default
margin");
```



Hyperlink on Shape

In DsExcel, 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: "https://developer.mescius.com/"	null
A range in this document	Example: null	"Sheet1!$C$3:$E$4"
Email Address	Example: "mailto: abc.xyz@mescius.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\mescius-logo.jpg", 1, 1, 100,
100);
//Add Hyperlink
worksheet.Hyperlinks.Add(picture, "https://developer.mescius.com/", null , "Click to
Open", "MESCIUS, Inc.");
//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@mescius.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://developer.mescius.com/", null, "Click to Open", "MESCIUS, Inc.");

//Delete hyperlink1.
hyperlink1.Delete();

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

Group or Ungroup Shapes

DsExcel 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 DsExcel, 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.

DsExcel 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 adjustent values for different adjustement types are described below:

Adjustment type	Valid values
Linear (horizontal or vertical)	Value 0.0 represents the left or top edge of the shape. Value 1.0 represents the right or bottom edge of the shape. 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. Values lesser than 0.0 and greater than 1.0 are also valid. The valid values for the adjustment correspond to the valid adjustments that can be made to shapes in Excel by extending the adjustment points. 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.
Radial	Value 1.0 represents the shape width. Hence, the maximum value for radial adjustment is 0.5, which is half way across the shape.
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

DsExcel 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 = "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.BackgroundPicture = 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

As a common use case scenario, you may want to render image at a particular position.

DsExcel allows you to fetch the exact position and size of the specified cell range in pixels by using **CellInfo.GetAccurateRangeBoundary** method that returns **RectangleF** object of type double. You can then use the **AddPictureinPixel** method to add an image at the fetched location. For more information about adding pictures, see Shapes and Pictures topic.

Following example code shows how to get the absolute location and size of the range and insert an image to that location.

C#

```
// Get the absolute location and size of the Range["F1:G1"] in the worksheet.
IRange range = worksheet.Range["F1:G1"];
// Get the RectangleF object of GetAccurateRangeBoundary method
RectangleF rect = GrapeCity.Documents.Excel.CellInfo.GetAccurateRangeBoundary(range);
// Add the image to the Range["F1:G1"]
worksheet.Shapes.AddPictureInPixel("logo.png", rect.X, rect.Y, rect.Width, rect.Height);
```

Note: Version v5.2 onwards, we obsoleted **CellInfo.GetRangeBoundary** method and replaced it with the **GetAccurateRangeBoundary** method which gives a more accurate position and size of the specified range.

Image Transparency

DsExcel 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. DsExcel 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 DsExcel 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.

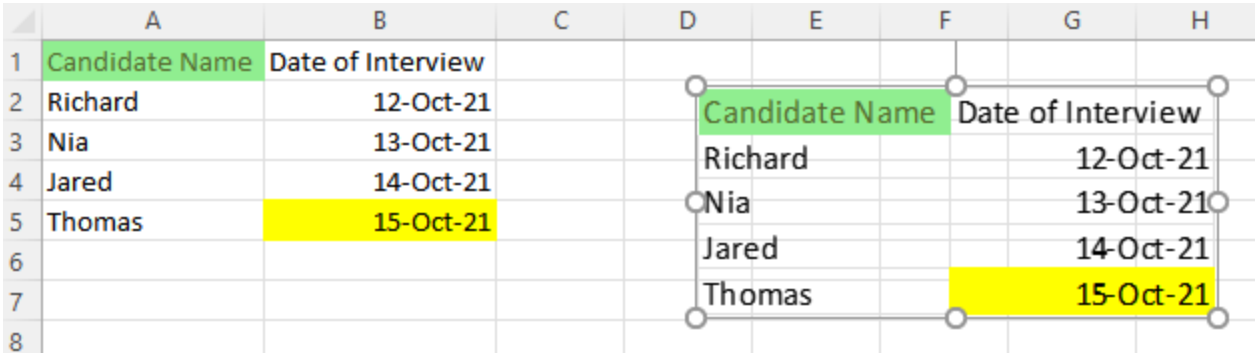
C#

```
// 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");
```

Linked Picture

Linked Picture, also known as Camera shape or picture, refers to a real-time dynamic snapshot of the copied range. That is, as values in the copied cell range change, the same is automatically reflected in its snapshot as well. In Microsoft Excel, this feature is provided through Special Paste option named "Linked Picture". This feature is especially useful in case of dashboards and reports as you can display dynamically changing images and can even resize them according to the available space.



DsExcel allows you to create these dynamic linked pictures through **AddCameraPicture** method of the **IShapes** interface. This method accepts the source range of the linked picture and position coordinates of the target range with respect to the document as parameters. You can also specify the cell range where you want to add the linked picture using another overload of this method. As the linked picture is also just another picture, it can be resized and formatted similar to any other picture.

C#

```
Workbook workbook = new Workbook();
workbook.ActiveSheet.Range["A1"].Interior.Color = Color.Blue;
workbook.ActiveSheet.Range["B5"].Interior.Color = Color.Yellow;
```

```
//Add linked picture at a specific position(coordinates)
IShape shape = workbook.ActiveSheet.Shapes.AddCameraPicture("=$A$1:$B$4", 100, 100);

//Add linked picture at a specific cell range
//IShape shapeRange = workbook.ActiveSheet.Shapes.AddCameraPicture("=$A$1:$B$5",
worksheet.Range["G1:H5"]);

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

Note: The targetRange and the linked picture to be added must exist in the same worksheet. Otherwise, it results into an **InvalidOperationException**.

Format Linked Picture

You can set whether the linked picture should display with a transparent background by setting the **IPictureFormat.TransparentBackGround** property to true.

C#

```
// Set transparent background
shape.PictureFormat.TransparentBackground = true;
// Set degree of transparency
shape.PictureFormat.Transparency = 0.8;
```

You can also convert a linked picture to a usual static picture by setting the **IPictureFormat.Reference** property to null. Whereas, when this property is set to a reference for a normal picture, it is converted into a linked picture. For detailed implementation of converting a normal picture to a linked picture or vice versa, see online demo.

Document Properties

Document properties can be used to identify some useful details about a document like the author, subject, category, comments, created date, last saved time etc. DsExcel supports storing built-in and custom document properties which can be exported and viewed in Excel documents. A few built-in properties can also be exported to PDF documents like Author, Title, Subject, Comments, CreatedDate and LastSavedTime.

Note: To know more about supported document properties in PDF, refer Support Document Properties.

The **IDocumentProperty** interface represents the document property and provides following properties:

- **Type**: Indicates the type of document property value
- **LinkToContent:** Indicates whether the property is connected to the content
- **LinkSource:** Indicates the linked content source
- **Name**: Indicates the name of the property
- **Value**: Indicates the value of the property

The **BuiltInDocumentPropertyCollection** of **IWorkbook** interface is a collection of built-in properties. The built-in properties can be set by using **BuiltInDocumentProperties** property and setting the value of any built-in property like Author, Subject etc.

The **CustomDocumentPropertyCollection** of **IWorkbook** interface provides overloaded **Add** methods and an

AddLinkToContent method. The overloaded **Add** methods can be used to create new custom document properties with string, boolean, datetime, double or integer values. The **AddLinkToContent** method can be used to create a new document property which is linked to named cells. It provides 'name' and 'source' parameters where the 'source' is a named range. If the named range is deleted, the last value that was saved in the custom property is stored. If the named range refers to more than one cell then the value of the cell in the top left corner is used.

Refer to the following example code to set Author and Company (built-in properties) and add a custom document property.

C#

```
IWorkbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A1"].Value = "hello";

// Add a defined name.
workbook.Names.Add("Headings", "=Sheet1!$A$1");

//Set values for built-in document properties
workbook.BuiltInDocumentProperties.Author = "John";
workbook.BuiltInDocumentProperties.Company = "Google";

// Add a custom document property.
 IDocumentProperty property =
workbook.CustomDocumentProperties.AddLinkToContent("Detail", "Headings");
var value = property.Value; // The value is "Hello".
var type = property.Type; // The Type is String.

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

Limitations

The following document properties are not supported in DsExcel.

- Number of Characters
- Number of Bytes
- Number of Lines
- Number of Paragraphs
- Number of Slides
- Number of Notes
- Number of Hidden Slides
- Number of Multimedia Clips
- Number of Characters (with spaces)

Styles

DsExcel .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 DsExcel .NET are listed below:

Category	Description	Properties
Number Format	Cell number format.	**IRange.NumberFormat**
Alignment	Horizontal and vertical alignment of cell content, indentation,text wrap, text rotation and text shrinking.	**IRange.AddIndent** **IRange.IndentLevel** **IRange.WrapText** **IRange.ShrinkToFit** **IRange.MergeCells** **IRange.ReadingOrder** **IRange.Orientation**
Font	IRange.Font(IFont)	**IRange.Font**(IFont)
Borders	Cell border line styles and colors.	**IRange.Borders**(IBorders)
Fill	Cell pattern fill or gradient fill.	**IRange.Interior**(IInterior)
Protection	Cell protection options (Locked and Hidden)	**IRange.Locked** **IRange.FormulaHidden**

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 = "Excel";
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 DsExcel .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

DsExcel .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 DsExcel .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 DsExcel.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.

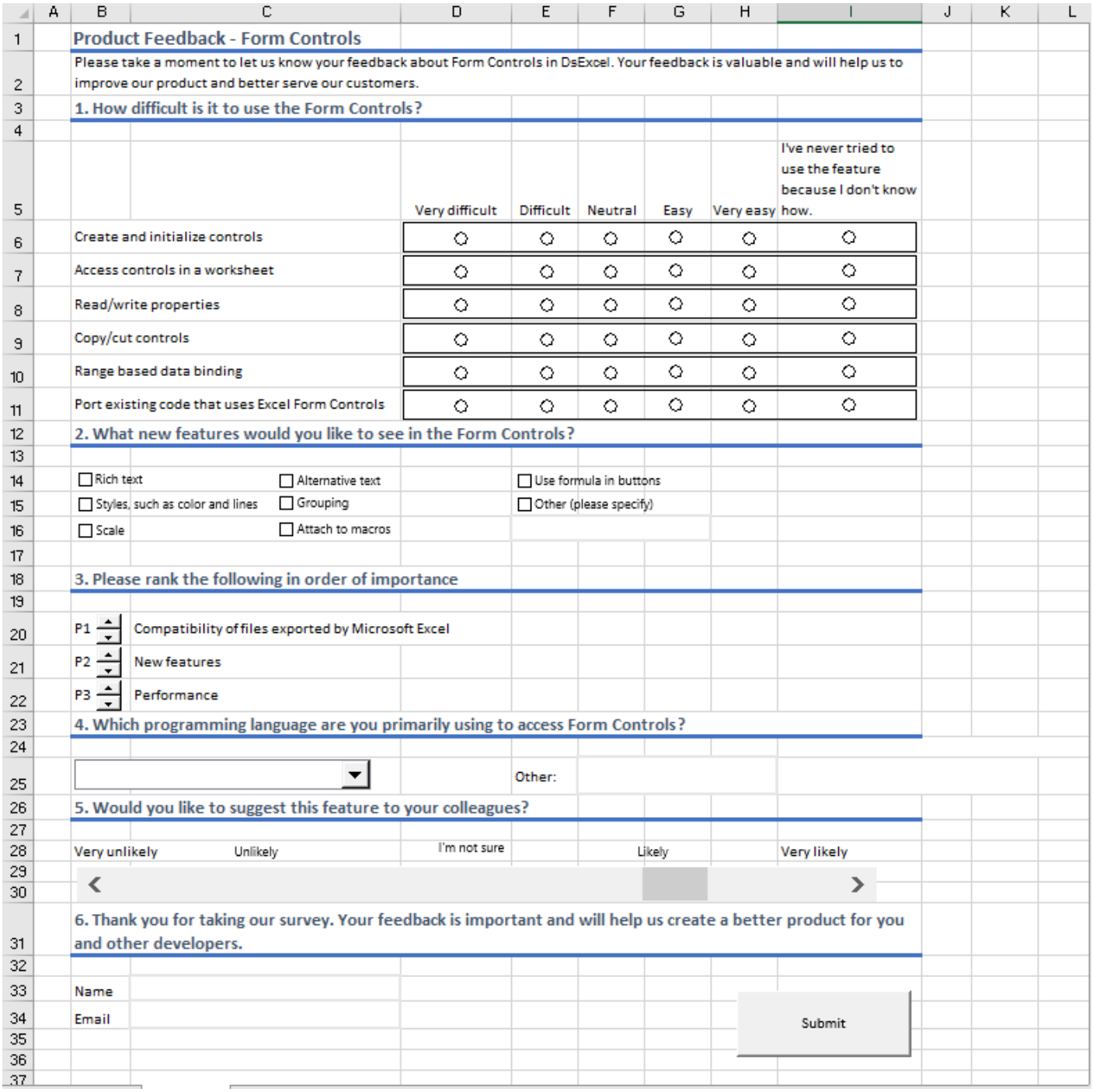
C#

```
// 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");
```

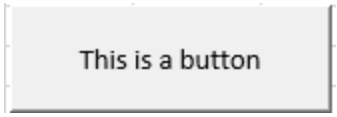
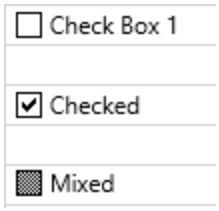
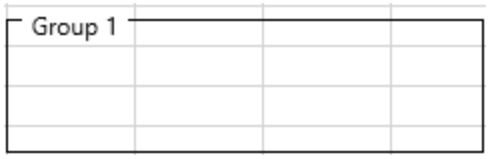
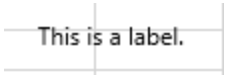
Form Controls

Form controls are objects which can be added to a worksheet to enable interaction with a cell or a data range available in the worksheet. You can seek input from the end-user or provide him with options to choose from by using these form controls. Hence, these controls are apt to create forms such as feedback forms or consent forms.



DsExcel provides nine form controls through **Grapecity.Documents.Excel.Forms** namespace which contains classes and interfaces for each supported form control.

The table below lists the supported form controls and their images.

Form Control	Snapshots
Button	
Dropdown	
Checkbox	
Spinner	
Listbox	
Option button	
Group box	
Label	
Scrollbar	

All the form controls possess some common features which are provided by **IControl** interface of the Grapecity.Documents.Excel.Forms namespace. You can disable these controls by setting the **Enabled** property to **false**, so that user cannot bring focus to that control. There is an option to even lock the controls from accepting user input by setting the **Locked** property to true. To define how a control is attached to the underlying cells, you can use the **Placement** property. You can also change ZOrder of the controls, bring form controls to front or send them to back by using the **BringToFront** and **SendToBack** properties.

Add and Remove Form Controls

DsExcel allows you to add or remove the form controls to a worksheet by using **Controls** property of the **IWorksheet**

interface. To add a form control to worksheet, you can use **Add<ControlName>** method of the **IControlCollection** interface. For instance, **AddButton** method adds the button form control and **AddDropdown** method adds the dropdown control to worksheet. So, there are nine such methods, one for each form control and all of them accept location coordinates, width, and height of the form control as parameters.

DsExcel provides **Delete** method of the **IControl** interface to remove a particular form control from worksheet. To remove all the controls from worksheet, you can use **Clear** method of the **IControlCollection** interface.

The code below demonstrates how to add or delete form controls to or from a worksheet:

C#

```
var workbook = new Workbook();
IWorksheet ws = workbook.Worksheets["Sheet1"];

// Add two controls
var lblResolution = ws.Controls.AddLabel(12.6, 20.4, 49.2, 18.6);
lblResolution.Text = "Resolution";
lblResolution.PrintObject = true;

var btnNative = ws.Controls.AddButton(199.8, 21, 127.8, 17.4);
btnNative.Text = "Use native resolution";
btnNative.PrintObject = true;

// Remove the first control
ws.Controls[0].Delete();

// Remove all the controls
// ws.Controls.Clear()
```

Link Form Controls to a Cell Range

The selection-based form controls, that are **Checkbox**, **Option button**, **Listbox**, **Dropdown**, and **Scrollbar** provide **LinkedCell** property of the **ICellLinkControl** interface that enables a two-way binding between value of the form control and the linked cell range. Linked cell range allows you to have a definite set of values in form control to avoid invalid data input from the end-user. You can update values of the form control by simply editing value in the linked cell range or vice-versa.

The code below shows how to link cell values to the Checkbox form control:

C#

```
// Link a check box
var checkBox1 = ws.Controls.AddCheckBox(54, 13.2, 64.2, 18);
checkBox1.LinkedCell = ws.Range["$A$2"];
```

Find Form Controls

DsExcel uses zero-based indexing while placing the form controls on a worksheet. You can find a form control in the

worksheet using its name or its type. To find an excel form control by its name, you can use **nameOf** operator to look for the specified name. While to find a control using its type, you can use the **FormControlType** enumeration.

See the code below to find the control by its name:

C#

```
// add the control
var lblResolution = ws.Controls.AddLabel(12.6, 20.4, 49.2, 18.6);
lblResolution.Text = "Resolution";
lblResolution.PrintObject = true;
lblResolution.Name = nameof(lblResolution);

// find the control by name
Console.WriteLine(ws.Controls[nameof(lblResolution)].Name);
```

See the code below to find the control by its type:

C#

```
// Get control by type and change width (without type conversion)
foreach (var ctl in ws.Controls)
{
 switch (ctl.FormControlType)
  {
  case FormControlType.Button:
            ctl.Width = 70;
            break;
 case FormControlType.CheckBox:
           ctl.Width = 60;
           break;
  }
}
```

Export Form Controls

Worksheets with form controls can be exported to PDF, XLSX, XLSM, HTML, .sjs, or SSJSON formats using **Save** method of the Workbook class and to PNG, SVG, JPG, or GIF formats using **ToImage** method of the IWorksheet interface. DsExcel provides **Visible** property in **IControl** interface that enables you to include or exclude the form controls while exporting. If you set Visible property of a form control to false, then that form control is not exported to either PDF, XLSX, XLSM, .sjs, SSJSON, HTML, PNG, SVG, JPG, or GIF formats.

Refer to the following example code to exclude a form control from exporting:

C#

```
// Add dropdown.
var dropDown = ws.Controls.AddDropDown(28.8, 81.8, 103.8, 31.4);
dropDown.PrintObject = true;
dropDown.Items.Add(new DropDownItem("Item 1"));
```

```
dropDown.Items.Add(new DropDownItem("Item 2"));
dropDown.Items.Add(new DropDownItem("Item 3"));
dropDown.SelectedIndex = 0;

// Set Visible to false.
dropDown.Visible = false;
```

Note: DsExcel exports the form controls as static images in the PNG, SVG, JPG, GIF, HTML, or PDF format.

For exporting form controls to interactive form fields in PDF, see Export Form Controls to Form Fields.

Form Control Shapes

DsExcel form controls are also shapes. Hence, to recognize whether a particular shape is a form control, **ShapeType** enumeration provides a **FormControl** member. To add onto this, if a shape is a form control, you can get the form control associated with the shape using the **Control** property of the **IShape** interface. Also, you can get shape associated with a form control using **ShapeRange** property of the IControl interface.

Refer to the following code to use form control as shape:

C#

```
// Add form control
var button1 = ws.Controls.AddButton(50, 100, 120, 40);
var buttonShape1 = button1.ShapeRange[0];

// Duplicate
buttonShape1.Duplicate();

// Size and move
buttonShape1.Left = 66.6;
buttonShape1.Top = 22.8;
buttonShape1.Width = 155.4;
buttonShape1.Height = 49.2;

// Delete
buttonShape1.Delete();
```

Add Option Button Group

In DsExcel, two or more option buttons can be grouped in a group box so that you can select one choice from several related but mutually exclusive choices. DsExcel groups the option buttons using the **GroupBox** property (read only) of **IOptionButton** interface which is identified by the boundaries of option buttons and group boxes. The GroupBox property is the first matched group box if an option button lies entirely within a group box. If there are no matching group boxes, the option button is in the default group, which is the worksheet.

When two or more option buttons are in the same group, they affect the selection state of other option buttons and allow you to select only one option button at a time in the same group. They also share the **LinkedCell** property, which means you can define LinkedCell for one option button in the group, and other option buttons in the same group can use the LinkedCell value.

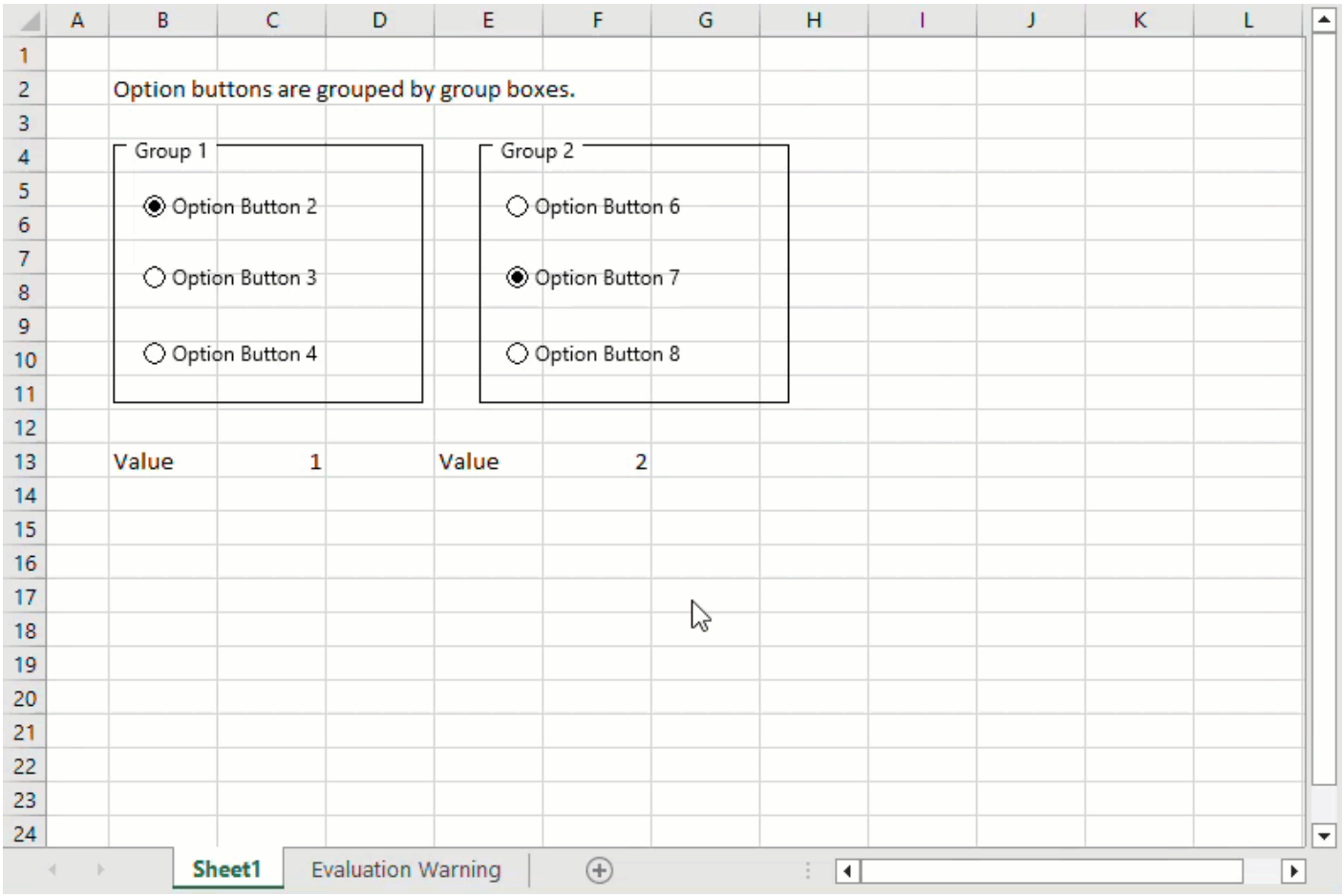
Note: Explicitly recalculate option button groups by calling **Cut(Left,Top)** on each group box control when a group box or option button loses focus.

Refer to the following example code to add two separate group boxes, each with respective linked option buttons:

C#

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

// Create a worksheet.
IWorksheet ws = workbook.Worksheets["Sheet1"];

// Add option buttons and group boxes to the worksheet.
var rngB2 = "Option buttons are grouped by group boxes.";
ws.Range["B2"].Value = rngB2;
ws.Range["B13:C13"].Value = new object[,] {
    { "Value", 1d}
};

ws.Range["E13:F13"].Value = new object[,] {
    { "Value", 2d}
};

ws.Range["A:A"].ColumnWidthInPixel = 37d;

// Add first group box.
var group1 = ws.Controls.AddGroupBox(29.75, 48.2, 136.5, 113.99);
group1.Text = "Group 1";

// Add option buttons.
var optionButton2 = ws.Controls.AddOptionButton(39.45, 67.1, 98, 15.60);

// Set linked cell.
optionButton2.LinkedCell = ws.Range["C13"];
optionButton2.IsChecked = true;

ws.Controls.AddOptionButton(39.45, 97.5, 98, 17.40);

ws.Controls.AddOptionButton(39.45, 131.2, 98, 17.5);

// Add second group box.
var group2 = ws.Controls.AddGroupBox(191.95, 48.2, 136.5, 113.99);
group2.Text = "Group 2";

// Add option buttons.
ws.Controls.AddOptionButton(200.35, 65.7, 117.6, 18.5);
```

```
var optionButton7 = ws.Controls.AddOptionButton(200.35, 95.99, 117.6, 21.28);

// Set linked cell.
optionButton7.LinkedCell = ws.Range["F13"];
optionButton7.IsChecked = true;

ws.Controls.AddOptionButton(200.35, 129.2, 117.6, 21.40);

// Save the workbook.
workbook.Save("OptionButtonsBasicUsage.xlsx");
```



Note: Adding option buttons to overlapping group boxes may not be the same as in Microsoft Excel.

Limitations

- Export of rich text and styles in DsExcel form controls is not supported.
- Getting or setting macro names in form controls requires VBA project.
- VBA project and COM interoperability is required for invoking macro when OnAction.
- LinkedObject and 'Copy to Clipboard' features require Window OLE automation interoperability.
- Spell check in Excel form controls requires Windows language pack interoperability.
- There is loss of some nodes during Excel I/O of alternate content in drawing Xml and Vml.

- In the option button group, some option buttons will join a different group when the measuring result differs from Excel.

Barcodes

DsExcel 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 DsExcel:

- QRCode
- EAN-13
- EAN-8
- Codabar
- Code39
- Code93
- Code128
- GS1-128
- Code49
- PDF417
- Data Matrix

Since Excel does not support barcode formulas, DsExcel provides **ConvertBarcodeToPicture** method in **IWorkbook** interface and **Workbook** class that enables a user to convert the calculated barcodes to pictures in all worksheets and save the file as an XLSX file. ConvertBarcodeToPicture method also allows the user to specify the image type using **ConvertBarcodeToPicture(ImageType imageType = ImageType.JPG)** overload. **ImageType** enumeration of **GrapeCity.Documents.Excel.Drawing** namespace sets the image type. The default image type is SVG if the image type parameter is not specified. The conversion does not support EMF and WMF image types, and the method will throw an unsupported exception.

Refer to the following example code to convert the resulting barcodes to pictures:

C#

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

// Access first worksheet.
IWorksheet worksheet = workbook.Worksheets[0];

// Set worksheet layout and add data.
worksheet.Range["A:A"].ColumnWidth = 2;
worksheet.Range["B:C"].ColumnWidth = 15;
worksheet.Range["D:G"].ColumnWidth = 25;
worksheet.Range["4:14"].RowHeight = 57;
worksheet.Range["B3"].Value = "Type";
worksheet.Range["C3"].Value = "Data";
worksheet.Range["B2"].Value = "Barcode";
worksheet.Range["B2:G2"].Merge(true);
worksheet.Range["D3:G3"].Value = new object[,]{
    {"Default", "Change color", "Change showLable", "Change lablePosition"}
```

```
};
worksheet.PageSetup.PrintTitleColumns = "$A:$C";
worksheet.Range["B4:C14"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B4:C14"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B2:G3"].HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["B2:G3"].VerticalAlignment = VerticalAlignment.Center;
worksheet.Range["B4:C14"].Value = new object[,]
{
    {"QR code", "Policy:411"},
    {"Data Matrix", "Policy:411"},
    {"PDF417", 6935205311092},
    {"EAN-8", 4137962},
    {"EAN-13", 6920312296219},
    {"Code39", 3934712708295},
    {"Code93", 6945091701532},
    {"Code49", 6901668002433},
    {"Code128", 465465145645},
    {"Codabar", 9787560044231},
    {"gs1128", 235465143135}
};
string[] types = {
    "BC_QRCODE",
    "BC_DataMatrix",
    "BC_PDF417",
    "BC_EAN8",
    "BC_EAN13",
    "BC_CODE39",
    "BC_CODE93",
    "BC_CODE49",
    "BC_CODE128",
    "BC_CODABAR",
    "BC_GS1_128"
};

// Add barcodes using barcode formula.
for (var i = 0; i < types.Length; i++)
{
    string columnD = "D" + (i + 4);
    string columnE = "E" + (i + 4);
    worksheet.Range[columnD].Formula = "=" + types[i] + "(C" + (i + 4) + ")";
    worksheet.Range[columnE].Formula = "=" + types[i] + "(C" + (i + 4) +
",\"#fff\",\"#000\")";
}

for (var i = 3; i < types.Length; i++)
{
    string columnD = "F" + (i + 4);
    string columnE = "G" + (i + 4);
```

```
    worksheet.Range[columnD].Formula = "=" + types[i] + "(C" + (i + 4) + ",,,0)";
    worksheet.Range[columnE].Formula = "=" + types[i] + "(C" + (i + 4) + ",,,,\"top\")";
}

// Convert resulting barcodes to pictures and set the image type to JPG.
workbook.ConvertBarcodeToPicture(GrapeCity.Documents.Excel.Drawing.ImageType.JPG);

// Save the workbook.
workbook.Save("ConvertBarcodeToPicture.xlsx");
```

DsExcel 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.

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	Defult	ge errorCorrectionL	Change model	Change version	Change mask
	Police	911					
	Travel Info Call 511	511					
		veloper.mescius.co					

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 DsExcel.

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", "Defult", "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"},
    { "", "developer.mescius.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. 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	hange addOnLabelPositio	Explain
	Medicine	692031229621	6 920312 296219	6 920312 296219 22	6 920312 296219 22	
	Pen	6945091701532	6 945091 701532	6 945091 701532 22	6 945091 701532 22	
	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 DsExcel.

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", "Defult", "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.

The below image displays EAN-8 barcode in a PDF document.

	EAN-8					
	Name	Number	Default	Change showLable	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.

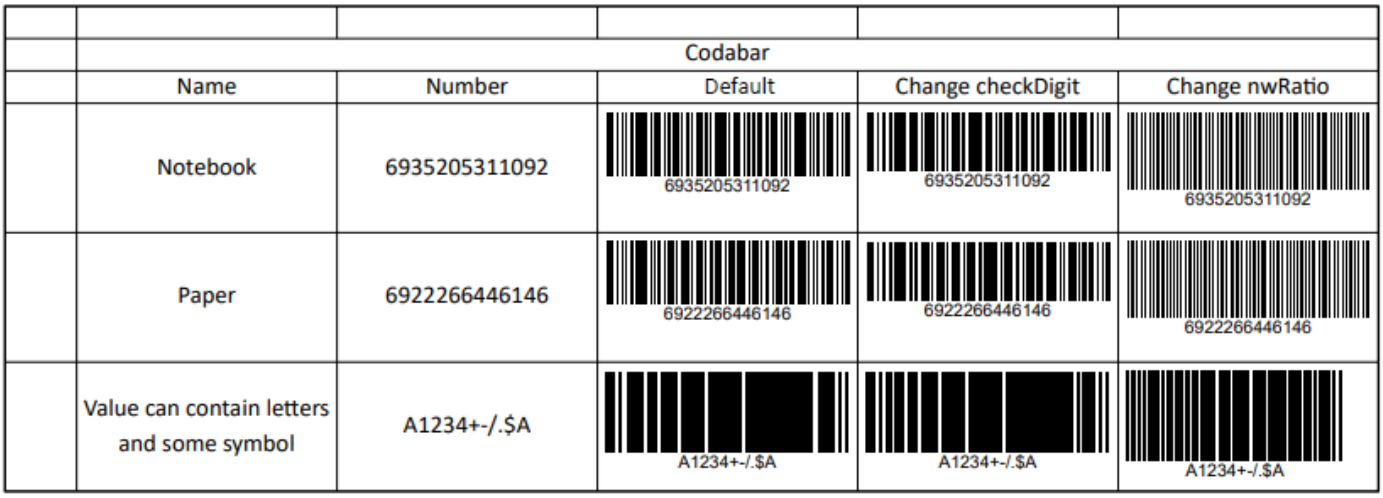
C#

```
// 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", "Defult", "Change showLable", "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.



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", "Defult", "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	ge labelWithStartAndStopChara	Change checkDigit
	Paper	6922266446146	6922266446146	6922266446146	6922266446146
	Book	9787560044231	9787560044231	9787560044231	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", "Defult", "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, backgroudColor, 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.

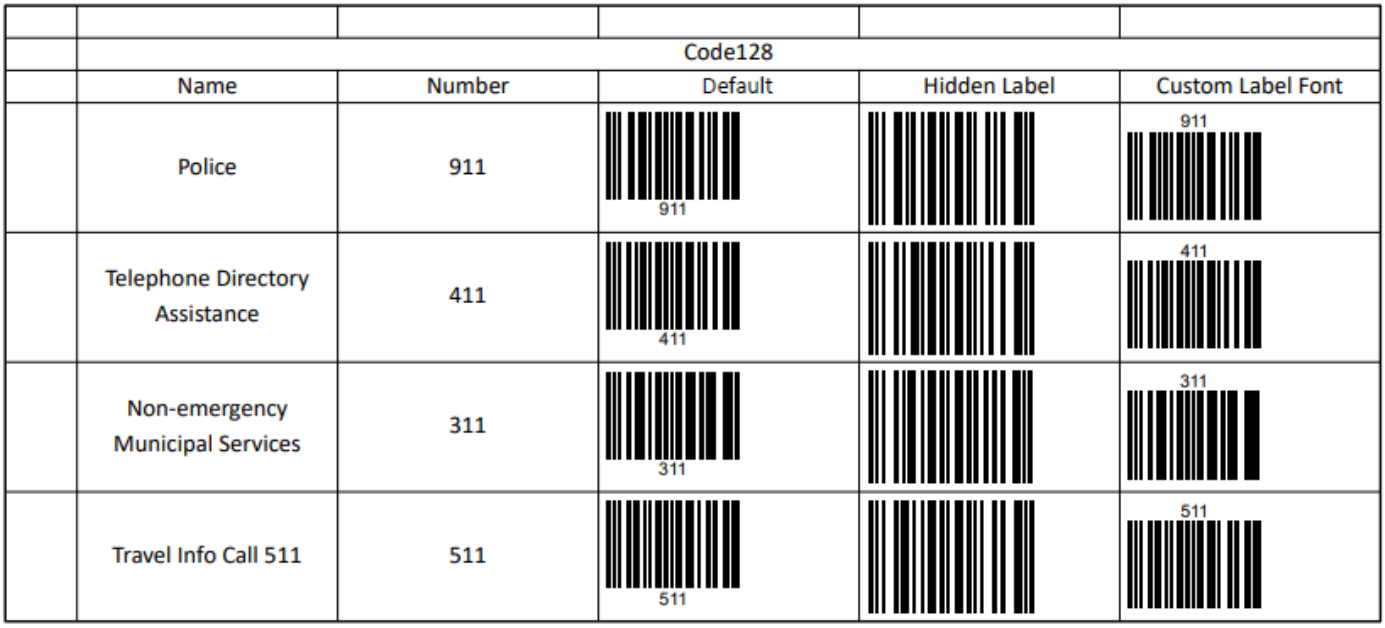
C#

```
// 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", "Defult", "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.



Formula definition

You can set Code128 in a worksheet using the following formula:

=BC_CODE128(value, color, backgroudColor, 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", "Defult", "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", "Defult", "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 encodes 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, backgroudColor, 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 Alphanimeric 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", "Defult", "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 mm2 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, backgroudColor, eccMode, ecc200SymbolSize, ecc200EndcodingMode, ecc00_140Symbole, structureAppend, structureNumber, fileIdentifier, 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'.

ecc200EndcodingMode	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", "Defult", "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

DsExcel .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 DsExcel, 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, DsExcel 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.

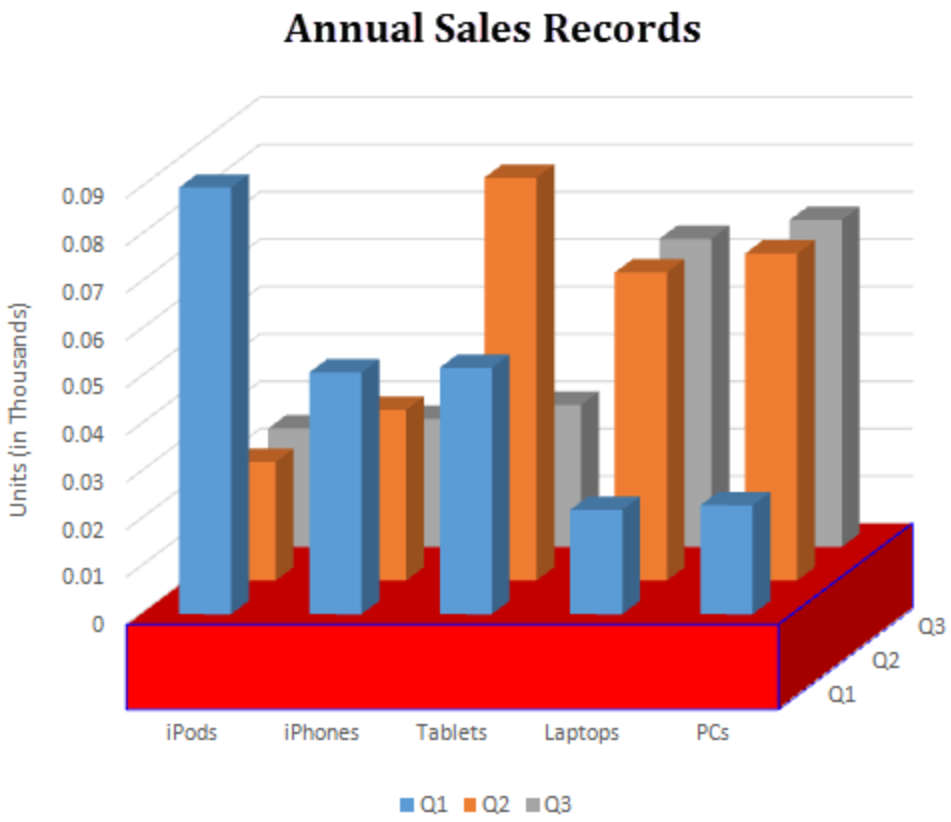
Refer to the following example code to add a custom theme and apply it to the workbook.

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

DsExcel .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

DsExcel .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 **IChart interface**

Create Chart

You can create chart in a worksheet by using **AddChart method** of the **IShapes** interface. Using this method, you can add a chart at a particular position by providing position coordinates of the target range. The method has another overload that lets you add a chart directly to the target range. You can use **Add** method of the **ISeriesCollection** class which lets you reflect data over the chart.

Refer to the following example code to create a chart.

C#

```
// Add Data
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}
};

// Add Chart at a specified position
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 300, 100, 300, 300);
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

// Add Chart at a particular range
// IShape shapeRange = worksheet.Shapes.AddChart(ChartType.ColumnClustered,
worksheet.Range["A8:E20"]);
// shapeRange.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);
```

Note: The targetRange and the chart to be added must exist in the same worksheet. Otherwise, it results into an **InvalidOperationException**.

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 DsExcel .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 DsExcel .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 Direction of Chart Title**
- **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 Direction of Chart Title

You can set the direction of the chart title to horizontal, vertical, rotated (to 90 or 270 degree), and stacked (with text reading left-to-right or right to left). The **Direction** property in **IChartTitle** and IChartTitle.**ITextFrame** interfaces allows you to set the direction of the chart title using **TextDirection** enumeration.

Refer to the following example code to set vertical chart title:

C#

```
// Create chart.
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

// Display the chart title.
shape.Chart.HasTitle = true;

// Set the chart title name.
```

```
shape.Chart.ChartTitle.Text = "Chart Title";

// Set the direction of chart title to vertical.
shape.Chart.ChartTitle.TextFrame.Direction = TextDirection.Vertical;

// OR
shape.Chart.ChartTitle.Direction = TextDirection.Vertical;
```

Set Text Angle for Chart Title

You can also set the text angle for chart title by using the **Orientation** property of **IChartTitle** interface.

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;
```

Note: The Orientation property only applies if the value of Direction property is Horizontal. However, the direction and orientation of the chart title can be exported or imported into a JSON file.

Chart Area

In DsExcel .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 DsExcel .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;
```

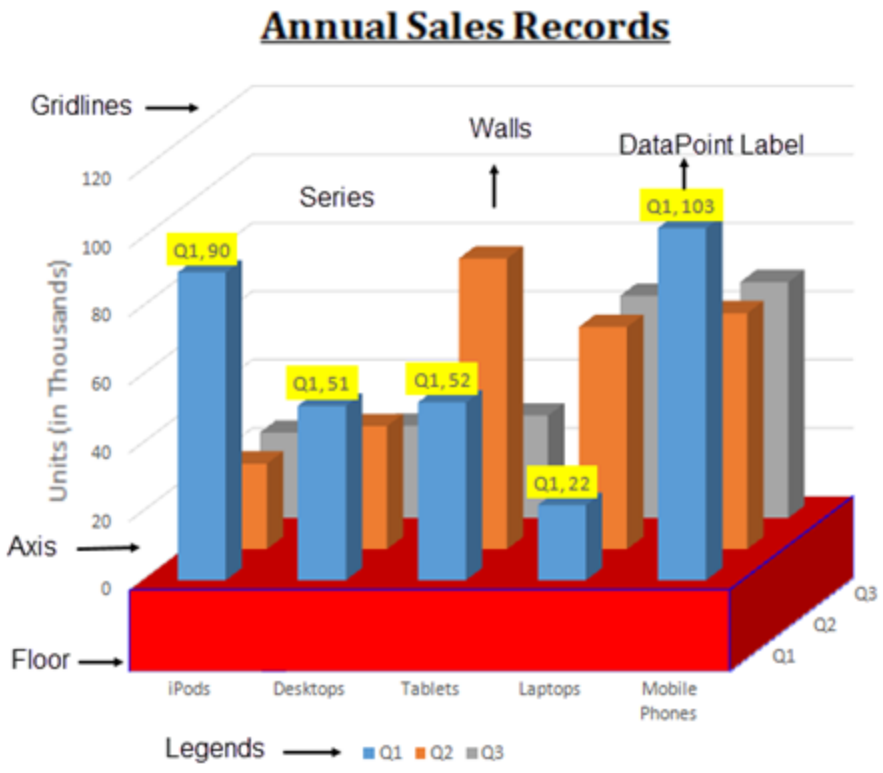
Customize Chart Objects

In DsExcel .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 DsExcel .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 DsExcel .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;
   //Iussue 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 DsExcel .NET, you can configure chart series using the following in your spreadsheet:

- **DataPoint**
- **DataLabel**
- **Trendline**
- **ChartGroup**
- **DropLine,HiLoLine and SeriesLine**
- **Up-Down Bars**
- **Smooth Line**

DataPoint

The Points collection in DsExcel .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 DsExcel .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;
```

Note: With version 7.0, the return value of **Parent** property of **IDataLabel** interface is changed from **IPoint** to **Object**, which will cause the compilation failure or runtime error in your existing projects. To avoid this, you must add explicit conversion of the object to IPoint as follows:

C#

```
// Following will cause a compilation error.
IPoint pt = series1.DataLabels[0].Parent;
// Add explicit conversion to IPoint.
IPoint pt = (IPoint)series1.DataLabels[0].Parent
```

Trendline

The Trendlines collection in DsExcel .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 set the trendline's name in DsExcel 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 DsExcel.

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");
```

Set trendline's label format

You can also format the trendline equation label and export it to a PDF document, HTML file, or image using **DataLabel** property in **ITrendline** interface, which gets the data label associated with the trendline. DataLabel returns value only when **DisplayEquation** or **DisplayRSquared** of ITrendline interface is true. If both of them are false, the DataLabel property will return null.

You can use **Font**, **Format**, **NumberFormat**, **Orientation**, **Direction**, and **AutoText** properties of **IDataLabel** interface to format the trendline equation label. DsExcel also provides **Delete** method to delete the trendline equation label.

Refer to the following example code to format the data label of the trendline:

C#

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

// Create a worksheet.
IWorksheet worksheet = workbook.Worksheets[0];

// Add XYScatter chart.
IShape shape = worksheet.Shapes.AddChart(ChartType.XYScatter, 250, 20, 360, 230);
worksheet.Range["A1:C11"].Value = new Object[,] {
    { null, "Mktng Exp", "Revenue" },
    { "Company 1", 1849, 2911 },
    { "Company 2", 2708, 5777 },
    { "Company 3", 3474, 8625 },
    { "Company 4", 4681, 9171 },
    { "Company 5", 5205, 10308 },
    { "Company 6", 5982, 11779 },
    { "Company 7", 8371, 12138 },
    { "Company 8", 8457, 17074 },
    { "Company 9", 9554, 15729 },
    { "Company 10", 9604, 19610 }
    };
shape.Chart.SeriesCollection.Add(worksheet.Range["B1:C11"], RowCol.Columns, true, true);
ISeries series1 = shape.Chart.SeriesCollection[0];

// Add Trendline.
ITrendline trendline = series1.Trendlines.Add();
trendline.Type = TrendlineType.Linear;

// Display equation for the trendline.
trendline.DisplayEquation = true;

// Format datalabel for trendline.
IDataLabel trendlineDataLabel = trendline.DataLabel;
```

```
trendlineDataLabel.Font.Color.RGB = Color.Purple;
trendlineDataLabel.Font.Size = 11;
trendlineDataLabel.Format.Fill.Color.ObjectThemeColor = ThemeColor.Accent4;
trendlineDataLabel.Format.Line.Color.ObjectThemeColor = ThemeColor.Accent2;

// Set paper size for PDF export.
worksheet.PageSetup.PaperSize = PaperSize.A3;

// Save the workbook.
workbook.Save("DataLabelTrendline.xlsx");

// Export the workbook as a PDF document.
workbook.Save("DataLabelTrendline.pdf");
```

Parent property (ITrendline.DataLabel.Parent) will return the parent object of the specified trendline. Its return value type is an object with ITrendline as the return value.

C#

```
ITrendline trendline = (ITrendline)trendline.DataLabel.Parent;
```

Note: The trendline equation label does not support the following properties of IDataLabel interface; hence, calling them will throw a NotSupportedException:

- Position
- Separator
- ShowBubbleSize
- ShowCategoryName
- ShowLegendKey
- ShowPercentage
- ShowSeriesName
- ShowValue
- NumberFormatLinked
- TextFrame

Limitations

SpreadJS only supports the default equation and R-value; therefore, DsExcel cannot export the trendline data format to JSON and SJS.

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 **IChartGroup 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;
```

Smooth Line

You can use **Smooth** property of **ISeries** Interface to smoothen the curves of line and scatter charts.

Configure Smooth Property for Line Chart

Refer to the following example code to configure Smooth property for the line chart:

C#

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

// Get the active sheet.
var worksheet = workbook.ActiveSheet;
```

```
// Set data for the chart.
var data = new object[,]
{
    { null , "2017", "2018", "2019", "2020", "2021", "2022", "2023" },
    { "Mobile Phones",0.9,0.13,0.15,0.18,0.17,0.18,0.04 },
    { "Tablets", 0.05, 0.08, 0.12, 0.13, 0.15, 0.17, 0.54 },
    { "Household items",0.43,0.35,0.23,0.13,0.13,0.15,0.16 },
    { "Vehicles",0.51,0.55,0.45,0.55,0.08,0.45,0.46 },
    { "Groceries",0.51,0.55,0.25,0.77,0.05,0.45,0.56 },
    { "Personal care",0.35,0.2,1,0.23,0.33,0.5,1 },
};

// Add data to the range.
worksheet.Range["A1:H7"].Value = data;

// Set style of the range.
worksheet.Range["A1:H7"].Style.HorizontalAlignment = HorizontalAlignment.Center;
worksheet.Range["A1:A7"].ColumnWidth = 18;

// Create line chart.
GrapeCity.Documents.Excel.Drawing.IShape shape =
worksheet.Shapes.AddChart(GrapeCity.Documents.Excel.Drawing.ChartType.Line, 10, 150,
400, 200);
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:H7"],
GrapeCity.Documents.Excel.Drawing.RowCol.Columns, true, true);
shape.Chart.ChartTitle.Text = "Product Sales";

// Set line curve setting.
foreach (var item in shape.Chart.SeriesCollection)
{
    // Set Smooth property to true.
    item.Smooth = true;
}

// Save the workbook in xlsx and pdf formats.
workbook.Save("SmoothLineChart.xlsx");
workbook.Save("SmoothLineChart.pdf");
```

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.

DsExcel 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

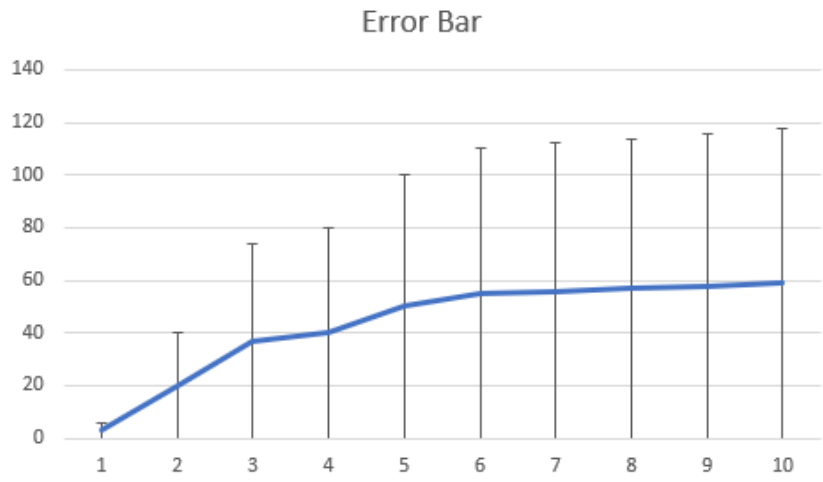
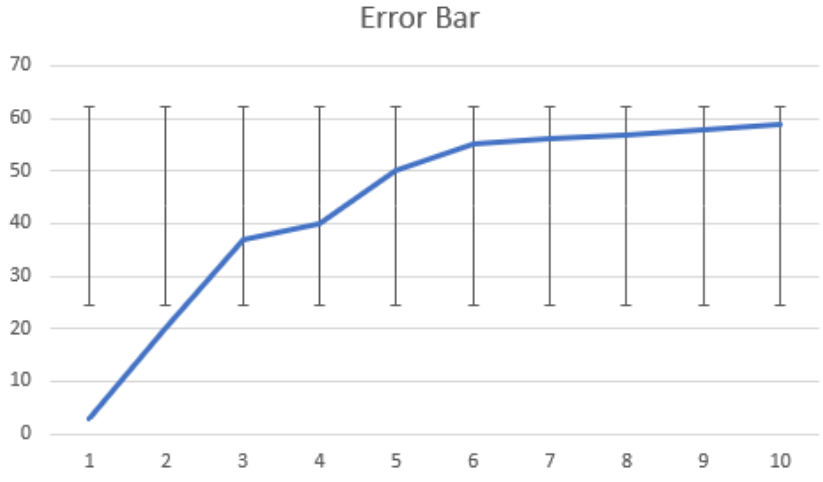
Type	Snapshot	Description
Plus		Error bar depicts only the positive values. C# `series.YErrorBar.Type = ErrorBarInclude.Plus;`
Minus		Error bar depicts only the negative values. C# `series.YErrorBar.Type = ErrorBarInclude.Minus;`

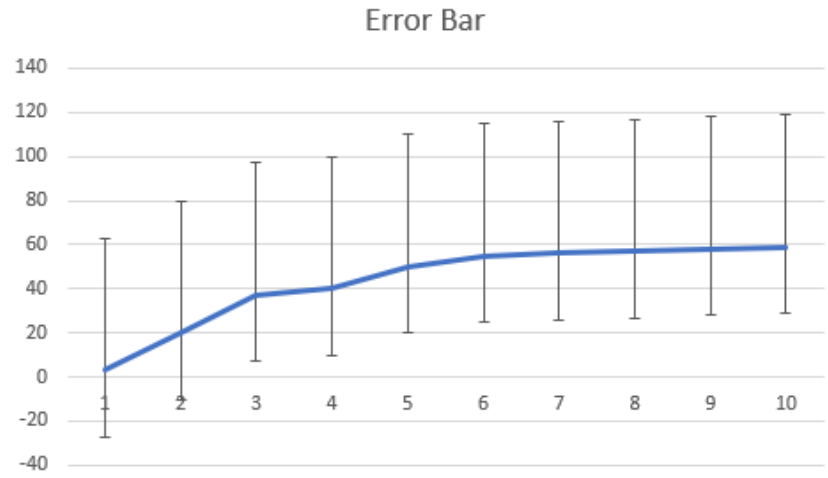
Both		Error bar depicts positive and negative values at the same time C# `series.YErrorBar.Type = ErrorBarInclude.Both;`

Error Bar End Styles

Type	Snapshot	Description
Cap		Error bar displays caps at the end of error bar lines. C# `series.YErrorBar.EndStyle = EndStyleCap.Cap;`
No Cap		Error bar does not display caps at the end of error bar lines. C# `series.YErrorBar.EndStyle = EndStyleCap.NoCap;`

Error Bar Value Types

Type	Snapshot	Description
Fixed Value		Error bar represents the error as an absolute value. C# `series1.YErrorBar.ValueType = ErrorBarType.FixedValue;`
Percentage		Error bar represents the error as a percentage of data value in the same direction axis. C# `series1.YErrorBar.ValueType = ErrorBarType.Percentage;`
Standard Deviation		Error bar represents the error as a calculating value which depends on the set deviation and chart data values. C# `series1.YErrorBar.ValueType = ErrorBarType.StDev;`

Standard Error		Error bar represents the error as a calculating value which only depends on the chart data values. C# `series1.YErrorBar.ValueType = ErrorBarType.StError;`
Custom		Error bar represents the error values that are set with positive and negative values respectively by formulas or fixed values. C# `series1.YErrorBar.ValueType = ErrorBarType.Custom;`

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 DsExcel and Excel.

Display Empty Cells

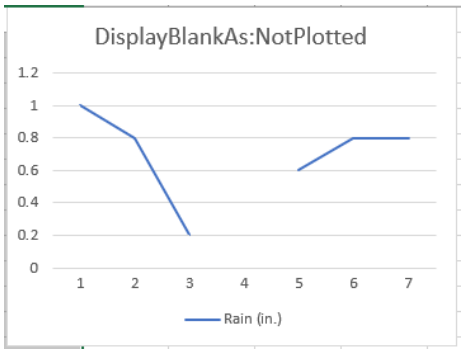
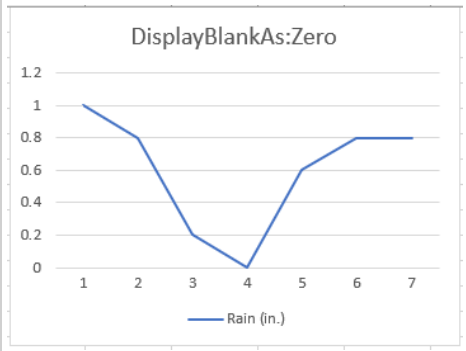
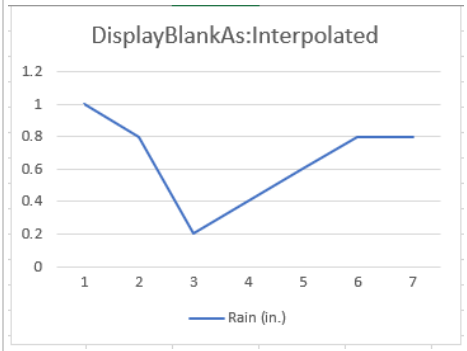
While plotting data on charts, you may have to deal with visualization of blank or empty cells on them. Depending on the requirement, you may want to plot empty values as gaps, zero or connected lines on the chart. This feature is helpful in plotting the missing data in a chart and hence facilitates users in quickly identifying and correcting any issues with the data.

Similar to Excel, DsExcel provides support for displaying empty cells and null values in charts using **DisplayBlanksAs** property. The property accepts values from **DisplayBlankAs** enumeration and can have following three values:

- **Interpolated**- Blank values are shown as 'interpolated' in the chart
- **NotPlotted**- Blank values are shown as 'gaps' in the chart
- **Zero** - Blank values are shown as 'zero' in the chart

This property is especially useful when **DisplayNaAsBlank** property is set to true. For more information, see Display #N/A Values.

Below images demonstrate how a daily rainfall chart is plotted with value against Day 4 as blank when DisplayBlankAs property is set to NotPlotted, Zero or Interpolated.

DisplayBlankAs = NotPlotted	DisplayBlankAs = Zero	DisplayBlankAs = Interpolated
		

Using Code

Below code shows how to set DisplayBlankAs property to plot blank values as zero.

C#

```
// Set show blank as zero.
shape.Chart.DisplayBlanksAs = DisplayBlanksAs.Zero;
```

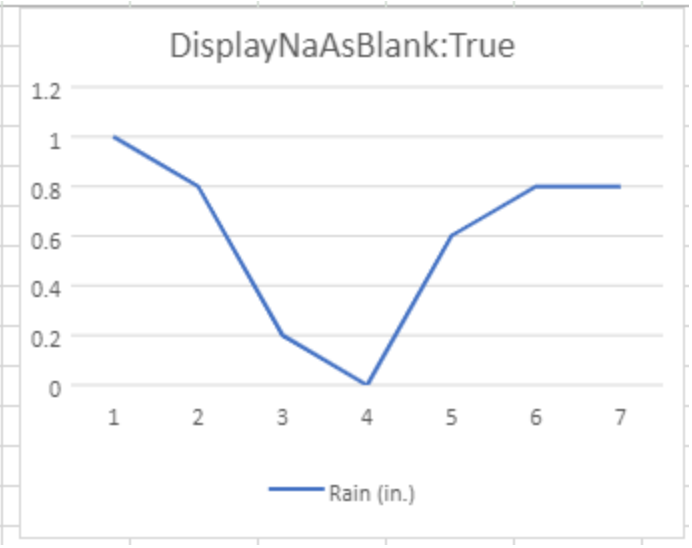
Display #N/A Values

DsExcel.NET provides support to display #N/A cells as empty cells in charts. The **DisplayNaAsBlank** property of **IChart** interface, when set to true, considers the cells containing #N/A values as empty cells. When DisplayNaAsBlank property is set to **true**, the display of empty cells, when plotted on a chart depends on the value of **DisplayBlanksAs** property. The property accepts values from **DisplayBlankAs** enumeration and can have following three values:

- **Interpolated**- #N/A values are shown as 'interpolated' in the chart
- **NotPlotted**- #N/A values are shown as 'gaps' in the chart
- **Zero** - #N/A values are shown as 'zero' in the chart

The below screenshot depicts a chart created from data containing #N/A cells which is displayed as zero because value of DisplayBlankAs is set to 'Zero'.

Day	Rain (in.)
1	1
2	0.8
3	0.2
4	#N/A
5	0.6
6	0.8
7	0.8



Following code demonstrates a line chart showing values plotted with **DisplayNaAsBlank** set to true.

C#

```
// Set show blank as zero.
shape.Chart.DisplayBlanksAs = DisplayBlanksAs.Zero;

// Set show #N/A as empty cell.
shape.Chart.DisplayNaAsBlank = true;
shape.Chart.ChartTitle.Text = "DisplayNaAsBlank:True";
```

When DisplayNaAsBlank property is set to **false**, chart plots the #N/A values depending on the chart type.

Walls

A wall refers to an area or a plane which is present behind, below or beside a chart.

DsExcel .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;
```

Axis and Other Lines

Axis is one of the charting elements meant for displaying the scale for a single dimension of a plot area. In DsExcel .NET, an axis in a chart can have a title, major tick mark, minor tick mark, tick mark labels, major gridlines and minor gridlines.

There are three types of axes in charts:

1. Category axis - Displays categories generally in the horizontal axis for all types of charts. An exception to this is the bar chart, where categories are shown along the y-axis that is, the vertical axis.
2. Value axis - Displays series values in vertical axis. An exception to this is the bar chart, where series values are shown along the x-axis that is, the horizontal axis.
3. Series axis - Displays data series for 3-dimensional charts including 3-D column chart, 3-D area chart, 3-D line chart, and surface charts.

Typically, a two-dimensional chart is comprised of two axes - category axis and value axis. While the category axis is also known as horizontal axis (x-axis) and is used to represent arguments, the value axis is also known as vertical axis (y-axis) and it represents the data values for rows and columns in a worksheet. However, in a three-dimensional chart, there is one more axis apart from the horizontal and vertical axis. This axis is known as the series axis.

You can use the properties of the **IAxis Interface** to configure category axis, value axis and series axis in a chart.

Refer to the following example code to configure axis in your chart.

C#

```
//Use IAxis.CategoryType to set category axis's scale type
IShape shape1 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 50, 300, 300);

worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {new DateTime(2015, 10, 21), 10, 25, 25},
    {new DateTime(2016, 10, 25), -51, -36, 27},
    {new DateTime(2017, 12, 20), 52, -85, -30},
    {new DateTime(2018, 5, 5), 22, 65, 65},
    {new DateTime(2019, 10, 12), 23, 69, 69}
};

shape1.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);
worksheet.Range["A2:A6"].NumberFormat = "m/d/yyyy";
IAxis category_axis = shape1.Chart.Axes.Item(AxisType.Category);
```

```
category_axis.CategoryType = CategoryType.AutomaticScale;
//Category axis's category type is automatic scale.
var categorytype = category_axis.CategoryType;
//Category axis's actual category type is time scale.
var actualcategorytype = category_axis.ActualCategoryType;


workbook.Worksheets.Add();
IWorksheet worksheet1 = workbook.Worksheets[1];
worksheet1.Range["A1:D6"].Value = new object[,]
 {
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
 };
//Set Category axis and Value axis's format.
IShape shape2 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 50, 300,
300);
shape2.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis category_axis1 = shape2.Chart.Axes.Item(AxisType.Category);
IAxis value_axis = shape2.Chart.Axes.Item(AxisType.Value);
//set category axis's format.
category_axis1.Format.Line.Color.RGB = Color.Green;
category_axis1.Format.Line.Weight = 3;
category_axis1.Format.Line.Style = LineStyle.ThickBetweenThin;
//set value axis's format.
value_axis.Format.Line.Color.RGB = Color.Red;
value_axis.Format.Line.Weight = 8;
value_axis.Format.Line.Style = LineStyle.ThinThin;

//Config time scale category axis's units.
worksheet1.Range["A8:A12"].NumberFormat = "m/d/yyyy";
worksheet1.Range["A7:D12"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {new DateTime(2015, 10, 21), 10, 25, 25},
    {new DateTime(2016, 10, 25), -51, -36, 27},
    {new DateTime(2017, 12, 20), 52, -85, -30},
    {new DateTime(2018, 5, 5), 22, 65, 65},
    {new DateTime(2019, 10, 12), 23, 69, 69}
};
IShape shape3 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 450, 300,
300);
shape3.Chart.SeriesCollection.Add(worksheet1.Range["A7:D12"], RowCol.Columns, true,
```

```
true);
IAxis category_axis2 = shape3.Chart.Axes.Item(AxisType.Category);
category_axis2.MaximumScale = new DateTime(2019, 10, 1).ToOADate();
category_axis2.MinimumScale = new DateTime(2015, 10, 1).ToOADate();
category_axis2.BaseUnit = TimeUnit.Years;
category_axis2.MajorUnitScale = TimeUnit.Months;
category_axis2.MajorUnit = 4;
category_axis2.MinorUnitScale = TimeUnit.Days;
category_axis2.MinorUnit = 60;

//Config value axis's units.
IShape shape4 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 800, 300,
300);
shape4.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis category_axis3 = shape4.Chart.Axes.Item(AxisType.Category);
IAxis value_axis1 = shape4.Chart.Axes.Item(AxisType.Value);
value_axis1.MaximumScale = 150;
value_axis1.MinimumScale = 50;
value_axis1.MajorUnit = 20;
value_axis1.MinorUnit = 5;

//Set axis crosses at.
IShape shape5 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 1150, 300,
300);
shape5.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis value_axis2 = shape5.Chart.Axes.Item(AxisType.Value);
value_axis2.Crosses = AxisCrosses.Maximum;

//Set axis's scale type.
IShape shape6 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 1500, 300,
300);
shape6.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis value_axis3 = shape6.Chart.Axes.Item(AxisType.Value);
value_axis3.ScaleType = ScaleType.Logarithmic;
value_axis3.LogBase = 5;

//Set axis's tick mark.
IShape shape7 = worksheet1.Shapes.AddChart(ChartType.ColumnClustered, 200, 1850, 300,
300);
shape7.Chart.SeriesCollection.Add(worksheet1.Range["A1:D6"], RowCol.Columns, true,
true);
IAxis category_axis4 = shape7.Chart.Axes.Item(AxisType.Category);
category_axis4.Format.Line.Color.RGB = Color.Green;
category_axis4.MajorTickMark = TickMark.Inside;
category_axis4.MinorTickMark = TickMark.Cross;
```

```
category_axis4.TickMarkSpacing = 2;
```

Configure Chart Axis

In DsExcel .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;
```

You can also set the direction of the axis title to horizontal, vertical, rotated (to 90 or 270 degree), and stacked (with text reading left-to-right or right to left). The **Direction** property in **IAxisTitle** and IAxisTitle.**ITextFrame** interfaces allows you to set the direction of the axis title using **TextDirection** enumeration.

Refer to the following example code to set the axis title direction to stacked:

C#

```
// Create chart.
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);

// Display the axis title.
category_axis.HasTitle = true;

// Set the name of axis title.
category_axis.AxisTitle.Text = "Category";

// Set direction of axis title to stacked.
category_axis.AxisTitle.TextFrame.Direction = TextDirection.Stacked;

// OR
category_axis.AxisTitle.Direction = TextDirection.Stacked;
```

DsExcel also allows you to configure the text angle of axis titles by using the **Orientation** property of IAxisTitle interface.

Refer to the following example code to set the text angle of the axis title:

C#

```
// Create chart.
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);

// Display the axis title.
category_axis.HasTitle = true;

// Set the name of axis title.
category_axis.AxisTitle.Text = "Category";

// Set axis title orientation to 45 degrees.
category_axis.AxisTitle.Orientation = 45;
```

The direction and orientation of the axis title can also be exported or imported into a JSON or PDF document.

> **Note:** The Orientation property only applies if the value of Direction property is Horizontal.

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#

```
// Create chart.
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 set the direction of the tick labels to horizontal, vertical, rotated (to 90 or 270 degree), and stacked (with text reading left-to-right or right to left). The **Direction** property in **ITickLabels** interface allows you to set the direction of the axis title using **TextDirection** enumeration.

Refer to the following example code to set the vertical tick labels:

C#

```
// Create chart.
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

// Set category axis.
var categoryAxis = shape.Chart.Axes.Item(AxisType.Category);

// Set category tick labels to vertical.
categoryAxis.TickLabels.Direction = TextDirection.Vertical;
```

DsExcel also allows you to configure the text angle of tick-mark labels by using the **Orientation** property of ITickLabels interface.

Refer to the following example code to set the text angle of tick mark label:

C#

```
// Create chart.
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");
```

The direction and orientation of the tick labels can also be exported or imported into a JSON or PDF document.

> **Note:** The Orientation property only applies if the value of Direction property is Horizontal.

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 DsExcel .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

DsExcel .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 lables' text.
series2.DataLabels.ShowCategoryName = true;
series2.DataLabels.ShowSeriesName = true;
series2.DataLabels.ShowLegendKey = true;
```

You can also set the direction of the data labels to horizontal, vertical, rotated (to 90 or 270 degree), and stacked (with text reading left-to-right or right to left). The **Direction** property in **IDataLabels**, **IDataLabel**, IDataLabels.**ITextFrame**, and IDataLabel.**ITextFrame** interfaces allows you to set the direction of the data labels using **TextDirection** enumeration.

Refer to the following example code to set the direction of the data labels to stacked and vertical:

C#

```
// Create chart.
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

// Add data labels to series 1.
ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

// Set direction of all data lables to stacked.
series1.DataLabels.Direction = TextDirection.Stacked;

// Set direction of first data label to vertical.
series1.Points[0].DataLabel.TextFrame.Direction = TextDirection.Vertical;

// Set direction of second data label to vertical.
series1.Points[1].DataLabel.Direction = TextDirection.Vertical;
```

DsExcel also allows you to configure the text angle for data labels by using the **Orientation** property of IDataLabel interface.

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");
```

The direction and orientation of the data labels can also be exported or imported into a JSON file.

Note: The Orientation property only applies if the value of Direction property is Horizontal.

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 DsExcel .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;
```

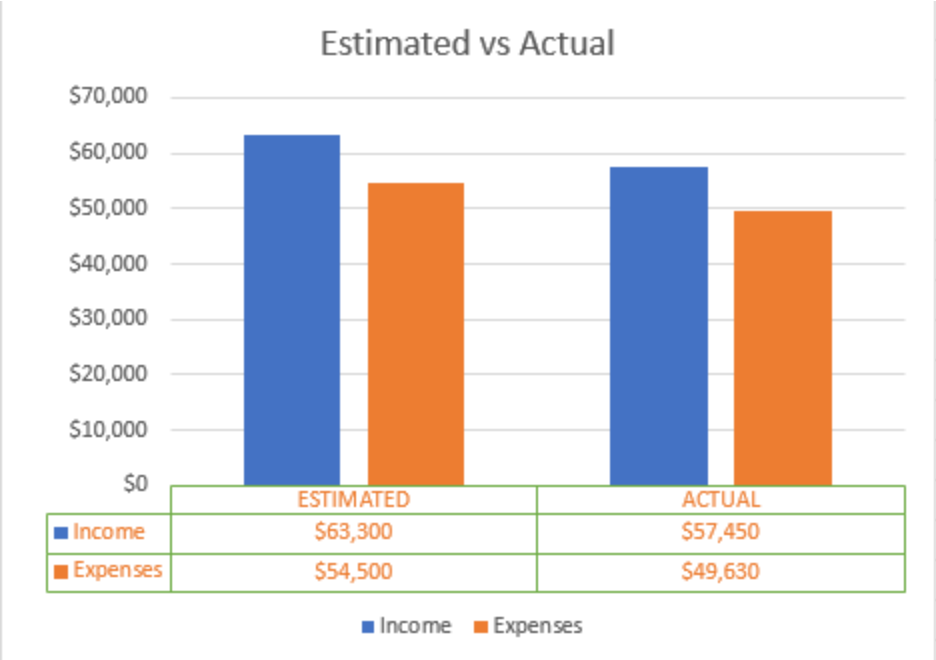
Refer to the following example code if you want to delete the legend or a specific legend entry from your chart.

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();
```

Data Table

Chart data table refers to a grid displaying source data of the chart and is drawn beneath the chart. The data table along with the chart is an organized form for reading exact values especially when data labels are difficult to read or are not set.



DsExcel .NET allows you to insert data table beneath chart by using **HasDataTable** property of the **IChart** interface. You can display data tables for the Column, Line, Bar, and Area chart by setting the HasDataTable property to **true**. However, for any other chart types, the property returns false.

Once a data table is added, you can configure the same by using various properties or methods of the **IDataTable** interface. The interface provides **HasBorderHorizontal**, **HasBorderVertical** and **HasBorderOutline** properties to display the cell borders and table outline respectively. You can configure the fill and line style of data table by using the **Format** property.

To remove the data table of a chart, you can call **Delete** method of the IDataTable interface.

Refer to the following example code to add and configure a chart data table in DsExcel:

C#

```
//Create chart.
GrapeCity.Documents.Excel.Drawing.IShape shape =
worksheet.Shapes.AddChart(GrapeCity.Documents.Excel.Drawing.ChartType.ColumnClustered,
250, 0, 350, 250);
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:C3"]);
shape.Chart.ChartTitle.Text = "Estimated vs Actual";

//Display the data table.
shape.Chart.HasDataTable = true;

//Config the data table.
```

```
GrapeCity.Documents.Excel.Drawing.IDataTable datatable = shape.Chart.DataTable;
datatable.Format.Line.Color.ObjectThemeColor = ThemeColor.Accent6;
datatable.Font.Color.ObjectThemeColor = ThemeColor.Accent2;
datatable.Font.Size = 9;
```

Limitation

Chart data table cannot be exported to JSON, PDF, HTML or image format.

Chart Types

DsExcel 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
Area Charts • Area • Area3D • AreaStacked • AreaStacked100 • AreaStacked1003D • AreaStacked3D		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.
Bar Charts • BarClustered • BarClustered3D • BarStacked • BarStacked100 • BarStacked1003D • BarStacked3D		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.
Column Charts • Column3D • ColumnClustered • ColumnClustered3D • ColumnStacked • ColumnStacked100 • ColumnStacked1003D • ColumnStacked3D		Unlike bar charts, Column charts use vertical columns/bars for representing data. These charts are generally used to plot data easily on X-axis.

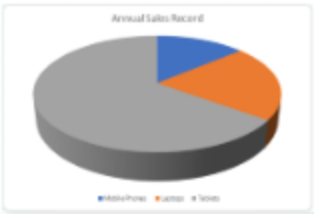
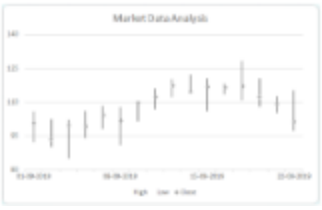
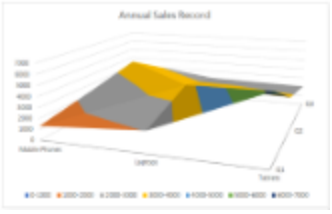
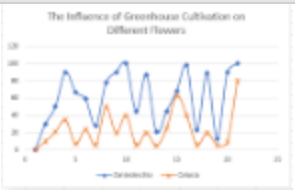
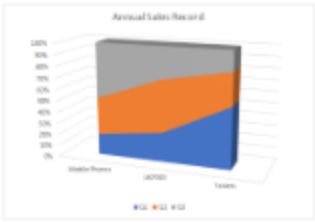
Chart Type	Chart Snapshot	Use Case
Combo Chart • Combo		The combo of two or more different charts can be used in the same plot area to compare the different data sets that are related to each other.
Line Charts • Line • Line3D • LineMarkers • LineMarkersStacked • LineMarkersStacked100 • LineStacked • LineStacked100		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.)
Pie Charts • Pie • Pie3D • PieExploded • PieExploded3D • PieOfPie • BarOfPie • Doughnut • DoughnutExploded		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).
Stock Charts • StockHLC • StockOHLC • StockVHLC • StockVOHLC		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.
Surface Charts • Surface • SurfaceTopView • SurfaceTopViewWireframe • SurfaceWireframe		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.

Chart Type	Chart Snapshot	Use Case
XY (Scatter) Charts • XYScatter • XYScatterLines • XYScatterLinesNoMarkers • XYScatterSmooth • XYScatterSmoothNoMarkers • Bubble • Bubble3DEffect		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.
Radar Charts • Radar • Radar Filled • Radar Markers		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.
Statistical Charts • Box and Whisker • Histogram • Waterfall • Pareto		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.
Specialized Charts • Sunburst • Treemap • Funnel		Specialized chart types provided by DsExcel 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.

DsExcel 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.
Aread3D		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.

DsExcel 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.

Chart Type	Chart Snapshot	Use Case
BarClustered3D		BarClustered3D chart is used to display the chart demonstration in 3D,which is a modification of 2DBarClustered 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.

DsExcel supports the following types of column charts.

Chart Type	Chart Snapshot	Use Case
Column3D		Column3D chart is used to display the chart demonstration in 3Dwhich is a modification of 2DColumn chart. It does not have a third dimension, it only looks volumetric in appearance.
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.

Chart Type	Chart Snapshot	Use Case
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.

DsExcel supports the following types of line charts.

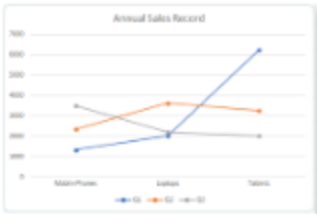
Chart Type	Chart Snapshot	Use Case
Line		Line chart is used to depict the data values plotted over time to display the trends. It shows continuous data over time on an evenly scaled Axis.
Line3D		Line3D chart is used to display the chart demonstration in 3D, which is a modification of 2D Line chart.
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.
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.

Chart Type	Chart Snapshot	Use Case
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
```

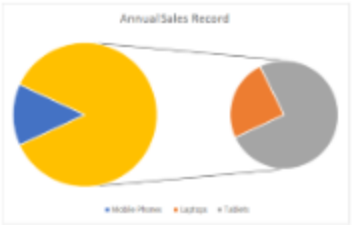
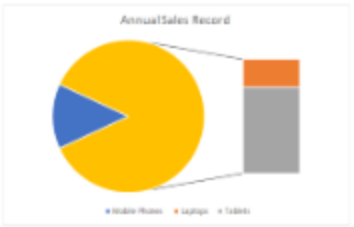
```
        lineChartshape.Chart.SeriesCollection.Add(worksheet.Range["A1:D4"],
RowCol.Columns, true, true);

        // 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.

DsExcel 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.
PieExploded		PieExploded chart is used to pull all of the slices out of a pie chart and view the sectors separately in pieces.

Chart Type	Chart Snapshot	Use Case
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.

DsExcel supports the following types of Stock charts.

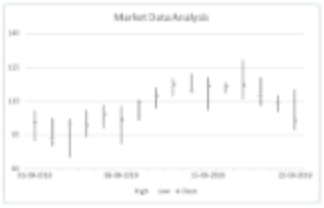
Chart Type	Chart Snapshot	Use Case
StockHLC		A high-low-close chart displays the data values organized in the order: high, low, close with the close value lying in between the high and low values.

Chart Type	Chart Snapshot	Use Case
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.

DsExcel 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.

DsExcel supports the following types of Scatter charts.

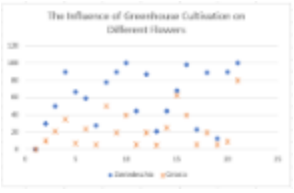
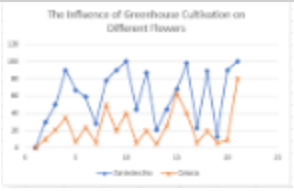
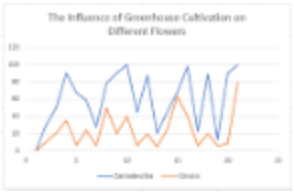
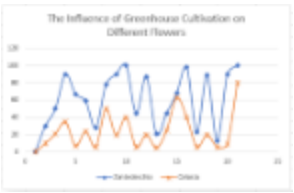
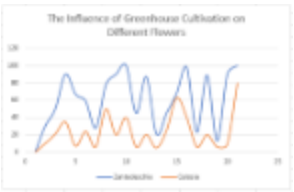
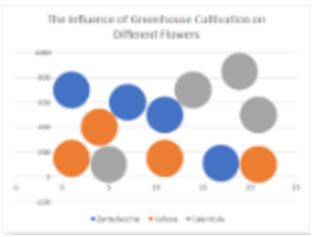
Chart Type	Chart Snapshot	Use Case
XYScatter		A clustered Scatter chart displays the data points based on a selected data range. This helps the users to analyze and determine the relationship between x and y variables.

Chart Type	Chart Snapshot	Use Case
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 2DBubble 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()
```

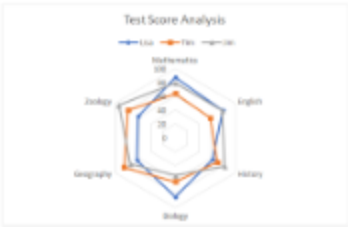
```
{
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.Worksheets[0];

    // 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.

DsExcel 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. DsExcel supports statistical chart types like Box and Whisker, Histogram, Waterfall and Pareto. Such chart types add visual meaning to the represented data.

DsExcel supports the following types of Statistical chart types:

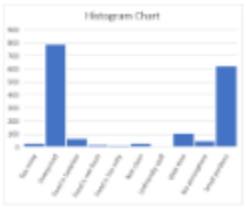
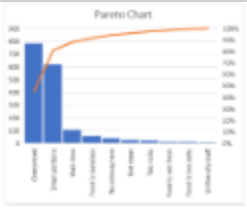
Chart Type	Chart Snapshot	Use Case
Box&Whisker		Box and Whisker charts are often used in Marketing Analysis, Statistical Analysis and General Analysis.
Histogram		Histogram is a common chart used in statistics. It can be used in scenarios, such as analysis of distribution/sales of books in a book store.
Waterfall Chart		Waterfall charts finds application in analyzing project gains including the number of contracts carried forwarded each year, contracts cancelled, tasks completed etc.

Chart Type	Chart Snapshot	Use Case
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

DsExcel 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

DsExcel 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, DsExcel 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 DsExcel are listed in the table below:

Methods/Properties	Description
Add(SheetType.Chart)	The **Add** method in **IWorksheets** interface has an overload with '**SheetType**'. Hence, for adding a Chart sheet, you need to use SheetType.Chart.
AddChart	The **AddChart** method in **IShapes** interface adds a chart for the Chart sheet. **Note:** Each Chart sheet should have a chart. Otherwise, it will throw an exception while saving the file.
AddShape	The **AddShape** method in **IChart** 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.
SheetType	The **SheetType** Property in **IWorksheet** interface gets the type of current sheet (Worksheet or Chart sheet).
Delete	The **Delete** method in **IShape** interface deletes the Chart from the Chart sheet, or deletes the shape from the Chart.
Copy	The **Copy** method in **IWorksheet** interface copies a new Chart sheet.
Move	The **Move** method in **IWorksheet** 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.

DsExcel .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 DsExcel .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 DsExcel .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();
```

Note: DsExcel also supports defined names, which can be used to name the table. For more information, see Defined Names.

Convert Table to Range

While a range refers to a group of cells, a table is nothing but a dynamic range of cells that is pre-formatted and organized. Just like converting a range into a table, you can also convert a table into a range. This feature is helpful when you want to use features such as dynamic arrays, which are not supported inside a table but, you want to have a table like formatting.

To handle such cases, DsExcel.Net provides **ConvertToRange**() method of the **ITable** interface to convert table into a range of cells without losing the table style. Converting table into a range of cells retains the cell data, table formatting and formulas but removes the table functionality. Note that on converting a table into a range, table reference used in formulas or Defind Names are converted to cell reference.

Refer to the following example code to convert table into range of cells in DsExcel.NET:

C#

```
// Add table
ITable table = worksheet.Tables.Add(worksheet.Range["A9:D13"], true);

// Convert table to range
table.ConvertToRange();
```

Modify Tables

While working with tables in DsExcel .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

DsExcel .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

DsExcel .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.

DsExcel .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 DsExcel .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 DsExcel .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 DsExcel .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

DsExcel .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

DsExcel .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#

```
//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");

// 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. DsExcel 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 comprehendible 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 DsExcel, 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.

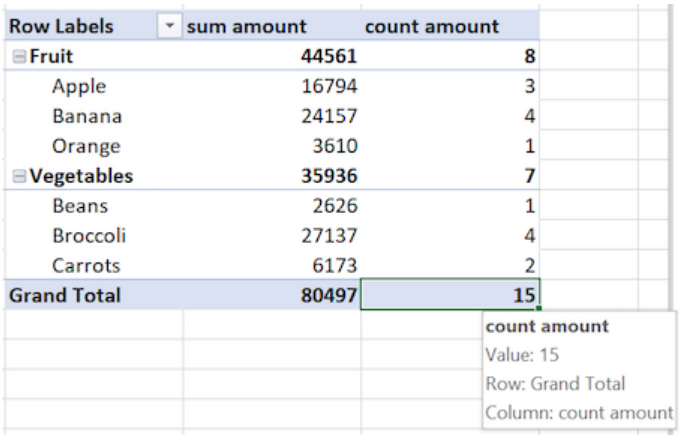
Refer to the following example code to add a pivot table field as multiple data fields by applying different calculation functions.

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
Fruit	44561	8
Apple	16794	3
Banana	24157	4
Orange	3610	1
Vegetables	35936	7
Beans	2626	1
Broccoli	27137	4
Carrots	6173	2
Grand Total	80497	15

Calculated Fields

Calculated fields in pivot table refer to the data fields created by applying additional logic or formula on existing data fields of the underlying data source. These fields are especially useful when summary functions and custom calculations do not generate the desired output. For instance, an employee database of a company holds data about existing salary and performance rating of each employee. At year end, one can easily calculate the raised salary of employees by creating calculated field using salary and the rating field.

In DsExcel, **CalculatedFields** property represents the collection of all calculated fields in a particular pivot table. You can use **Add** method of the **ICalculatedFields** interface to create a new calculated field in the pivot table. The Add method accepts fieldname and **IPivotField.Formula** property as its parameters to generate the calculated field. To remove a calculated field from the collection you can use the **Remove** method which takes target field name as its parameter.

Refer to the following code to create a calculated field in the pivot table:

C#

```
IWorksheet calculatedFieldSheet = workbook.Worksheets.Add();
calculatedFieldSheet.Name = "CalculatedField";

// Add pivot table.
IPivotCache pivotCache = workbook.PivotCaches.Create(worksheet.Range["A1:F71"]);
IPivotTable calculatedFieldTable = calculatedFieldSheet.PivotTables.Add(pivotCache, calculatedFieldSheet.Range["A1"]);
calculatedFieldTable.PivotFields["Product"].Orientation = PivotFieldOrientation.RowField;
calculatedFieldTable.PivotFields["Amount"].Orientation = PivotFieldOrientation.DataField;
calculatedFieldTable.DataFields["Sum of Amount"].NumberFormat = "$#,##0_);($#,##0)";

// Add calculated field.
calculatedFieldTable.CalculatedFields.Add("Tax", "=IF(Amount > 1000, 3% * Amount, 0)");

// Set calculated field as data field.
calculatedFieldTable.PivotFields["Tax"].Orientation = PivotFieldOrientation.DataField;
calculatedFieldTable.DataFields["Sum of Tax"].NumberFormat = "$#,##0_);($#,##0)";
```

Calculated Items

Calculated items are pivot table items that use custom formulas containing constants or refer to other items in the pivot table. These items can be added to the

row or column field area of the pivot table but do not exist in the source data.

In DsExcel, **ICalculatedItems** interface represents the collection of calculated items in a particular pivot table. You can fetch this collection of pivot items by using **IPivotField.CalculatedItems** method. To add calculated items to a pivot table, the **ICalculatedItems** interface provides **Add** method which accepts name and formula of the item as parameters. You can also use **IPivotItem.Formula** for setting the formula of a calculated item. To remove a calculated item from the ICalculatedItems collection, you can use **Remove** method which accepts name of the target field as parameter. Pivot cache manages all the calculated items, hence changing a calculated item affects all pivot tables using same cache in the current workbook. Also, any kind of addition, deletion or change in calculated item triggers the pivot table update.

> **Note**: An exception is thrown on:
>
> - Adding the same field to the data field section when a calculated item exists in the pivot table.
> - Adding calculated items when data field is having two or more same fields.
> - Adding a calculated item with a used name. **Name** parameter of calculated item is case-insensitive. Hence, pivot table considers the name “Formula” and “formula” as same.

Refer to the following code to create calculated items in the pivot table:

C#

```
// Get the calculated item for the specified field
ICalculatedItems countryCalcItems = calculatedItemTable.PivotFields["Country"].CalculatedItems();
ICalculatedItems productCalcItems = calculatedItemTable.PivotFields["Product"].CalculatedItems();

// Add some calculated items using formulas
countryCalcItems.Add("Oceania", "=Country[Australia]+Country[NewZealand]");
countryCalcItems.Add("America", "=Country[Canada]");
IPivotItem myPivotItem = countryCalcItems.Add("Europe", "=Country[France]");

// Change the formula of the calculated item
myPivotItem.Formula = "=Country[France]+Country[Germany]";

// Add calculated item using constant value
productCalcItems.Add("IPhone 13", "=2500");

// Get the CalculatedItema count
Console.Write("Calculated Items count : " + countryCalcItems.Count);

// Remove a calculated item
countryCalcItems.Remove("America");
```

Show Value As

While analyzing spreadsheet data, instead of comparing exact values, you may want to compare the values in terms of calculations. For instance, there are many ways to evaluate performance of a sales employee. You can compare his sales with target, sales as a percentage of total sales or sales in comparison to previous month's sale etc. To achieve these calculations easily, DsExcel provides "Show Value As" option which allows you to perform custom calculations in a pivot table by using several predefined formulas such as “% of Parent Total” or “% of Grand Total”.

DsExcel.NET provides **Calculation** property of **PivotField** interface which accepts values from **PivotFieldCalculation** enumeration for setting the predefined calculations. You can also set the base field and base field item to perform these calculations using **BaseField** and **BaseItem** properties respectively.

Sum of Amount	Column Lab							
Row Labels	Australia	Canada	France	Germany	NewZealand	United Kingdom	United States	Grand Total
Consumer Electronics	**100.00%**	**57.49%**	**111.46%**	**172.90%**	**76.22%**	**89.36%**	**182.54%**	
Bose 785593-0050	100.00%	100.00%	1350.00%	125.93%	11.76%	700.00%	160.71%	
Canon EOS 1500D	100.00%	31.25%	13.33%	650.00%	130.77%	47.06%	312.50%	
Haier 394L 4Star	100.00%	61.54%	275.00%	54.55%	216.67%	165.38%	106.98%	
IFB 6.5 Kg FullyAuto	100.00%	89.36%	88.10%	108.11%	50.00%	115.00%	160.87%	
Mi LED 40inch	100.00%	9.52				48.39%	213.33%	
Sennheiser HD 4.40	100.00%	75.0				20.93%	500.00%	
Mobile	**100.00%**	**83.0**				**238.60%**	**58.82%**	
Iphone XR	100.00%	74.3				219.05%	43.48%	
OnePlus 7Pro	100.00%	22.22%	416.67%	36.00%	200.00%	200.00%	61.11%	
Redmi 7	100.00%	92.86%	43.59%	247.06%	2.38%	4200.00%	78.57%	
Samsung S9	100.00%	139.29%	123.08%	4.17%	850.00%	70.59%	41.67%	
Grand Total	**100.00%**	**68.98%**	**104.78%**	**126.94%**	**71.22%**	**132.32%**	**118.32%**	

Sum of Amount
Value: 89.36%
Row: Consumer Electronics - IFB 6.5 Kg FullyAuto
Column: Canada

Refer to the following example code which demonstrates the value as percent of Australia.

C#

```
IPivotTable percentOfTable = percentOfSheet.PivotTables.Add(pivotCache, percentOfSheet.Range["A1"]);
percentOfTable.PivotFields["Category"].Orientation = PivotFieldOrientation.RowField;
percentOfTable.PivotFields["Product"].Orientation = PivotFieldOrientation.RowField;
percentOfTable.PivotFields["Country"].Orientation = PivotFieldOrientation.ColumnField;
percentOfTable.PivotFields["Amount"].Orientation = PivotFieldOrientation.DataField;

// set show value as, base field, base item.
IPivotField percentOfTableDataField = percentOfTable.DataFields["Sum of Amount"];
percentOfTableDataField.Calculation = PivotFieldCalculation.PercentOf;
percentOfTableDataField.BaseField = "Country";
percentOfTableDataField.BaseItem = "Australia";

percentOfSheet.Range["A:I"].AutoFit();
```

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.

DsExcel 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

DsExcel 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

DsExcel 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.

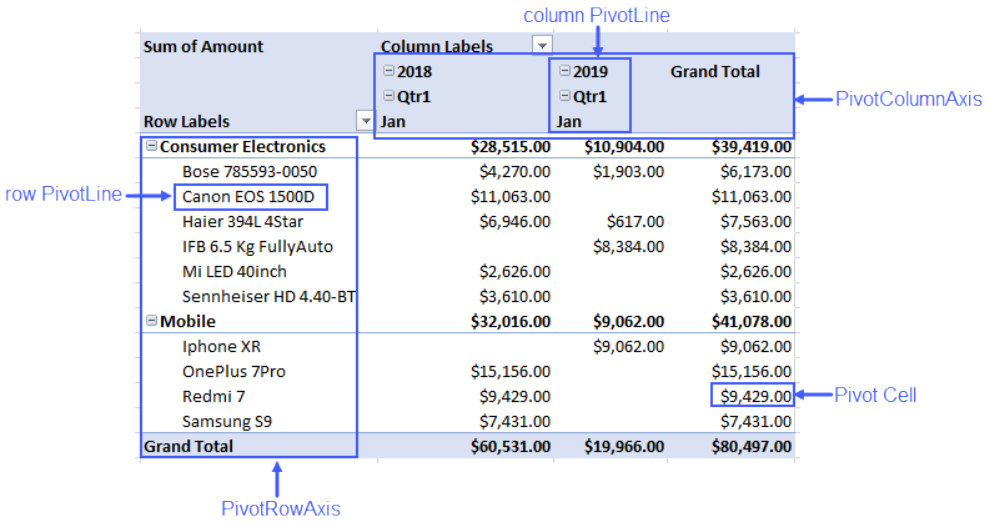
Refer to the following example code to sort 'Product' field in a pivot table.

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.



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

DsExcel 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
Consumer Electronics	
Bose 785593-0050	$6,173.00
Canon EOS 1500D	$11,063.00
Haier 394L 4Star	$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
Consumer Electronics Total	**$39,419.00**
Mobile	
Iphone XR	$9,062.00
OnePlus 7Pro	$15,156.00
Redmi 7	$9,429.00
Samsung S9	$7,431.00
Mobile Total	**$41,078.00**
Grand Total	**$80,497.00**

Note: Style applied to a pivot table is lost if the pivot table is changed in any way.

Get Pivot Table Data

DsExcel.NET provides **GETPIVOTDATA** function which queries the pivot table to fetch data as per the specified parameters. The main advantage of using this function is that it ensures that the correct data is returned, even if the pivot table layout has changed.

Syntax

=GETPIVOTDATA(data_field, pivot_table, [field1, item1, field2, item2],...)

GETPIVOTDATA function can be implemented to return a single cell value or a dynamic array depending on the parameters we are passing. To retrieve a single cell value, name of the data field and pivot table are mandatory parameters. While the third parameter which is a combination of field names and item names, is optional. However, for retrieving a dynamic array, all three parameters are required and the item name supports array like {"Canada", "US", "France") or a range reference like A1:A3. Also, you must use **IRange.Formula2{get;set;}** for GETPIVOTDATA function to return a dynamic array which is spilled across a range. For ease of use, you can also automatically generate GETPIVOTDATA function by using the **IRange.GenerateGetPivotDataFunction** method. However, the GenerateGetPivotDataFunction method returns null when the IRange object is not a single cell.

Refer to the following example code for GETPIVOTDATA function returning a single cell:

C#

```
// GenerateGetPivotDataFunction method is used to generate formula automatically for the selected cell
var worksheet2 = workbook.Worksheets.Add();
worksheet.Range["H25"].Formula = worksheet.Range["G6"].GenerateGetPivotDataFunction(worksheet2.Range["A1"]);

// Here, the GETPIVOTDATA function is used to fetch the desired result
worksheet2.Range["H24"].Formula =
@"=GETPIVOTDATA(""Amount"",Sheet1!$A$1,""Category"",""Mobile"",""Country"",""Australia"")";
```

Refer to the following example code for GETPIVOTDATA function returning a dynamic array:

C#

```
// Here, Formula2 is used along with GETPIVOTDATA to fetch the multiple values
worksheet.Range["H10"].Formula2 = @"=GETPIVOTDATA(""Amount"",$A$1,""Category"",""Consumer Electronics"",""Country"",
{""Canada"",""Germany"",""France""})";
```

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 condional 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. DsExcel 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)
	

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;
```

Expand or Collapse Pivot Table Fields

DsExcel provides **ShowDetail** property in **IPivotItem** interface which allows you to expand or collapse the outline of pivot table fields. The default value of the property is True which shows the expanded state of pivot table fields. However, it can be set to False to display the collapsed state.

Refer to the following example code to set collapsed state of two pivot table fields.

```
C#
worksheet.Range["A1:F16"].Value = sourceData;
IPivotCache pivotCache = workbook.PivotCaches.Create(worksheet.Range["A1:F16"]);
IPivotTable pivotTable = worksheet.PivotTables.Add(pivotCache, worksheet.Range["H7"], "pivottable1");

pivotTable.PivotFields["Product"].Orientation = PivotFieldOrientation.RowField;
pivotTable.PivotFields["Country"].Orientation = PivotFieldOrientation.RowField;
pivotTable.PivotFields["Category"].Orientation = PivotFieldOrientation.ColumnField;
pivotTable.PivotFields["Amount"].Orientation = PivotFieldOrientation.DataField;

//Set collapsed state
pivotTable.PivotFields["Product"].PivotItems["Banana"].ShowDetail = false;
pivotTable.PivotFields["Product"].PivotItems["Carrots"].ShowDetail = false;

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

Pivot Table Style

DsExcel .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 Row Labels	Column Labels Apple	Banana	Beans	Broccoli	Carrots	Orange	Grand Total
Fruit	**$9,848.00**	**$24,157.00**				**$3,610.00**	**$37,615.00**
Canada	$7,431.00	$8,384.00					$15,815.00
France	$2,417.00						$2,417.00
Germany		$8,250.00					$8,250.00
New Zealand		$6,906.00					$6,906.00
United States		$617.00				$3,610.00	$4,227.00
Vegetables			**$2,626.00**	**$24,313.00**	**$6,173.00**		**$33,112.00**
Australia				$9,062.00			$9,062.00
Germany			$2,626.00		$1,903.00		$4,529.00
United Kingdom				$8,239.00			$8,239.00
United States				$7,012.00	$4,270.00		$11,282.00
Grand Total	$9,848.00	$24,157.00	$2,626.00	$24,313.00	$6,173.00	$3,610.00	$70,727.00

Refer to the following example code in order to apply 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	Grand Total
Fruit	$9,848.00	$24,157.00				$3,610.00	$37,615.00
Canada	$7,431.00	$8,384.00					$15,815.00
France	$2,417.00						$2,417.00
Germany		$8,250.00					$8,250.00
New Zealand		$6,906.00					$6,906.00
United States		$617.00				$3,610.00	$4,227.00
Vegetables			$2,626.00	$24,313.00	$6,173.00		$33,112.00
Australia				$9,062.00			$9,062.00
Germany			$2,626.00		$1,903.00		$4,529.00
United Kingdom				$8,239.00			$8,239.00
United States				$7,012.00	$4,270.00		$11,282.00
Grand Total	$9,848.00	$24,157.00	$2,626.00	$24,313.00	$6,173.00	$3,610.00	$70,727.00

Refer to the following example code in order to apply custom style to the pivot table.

C#

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

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

object[,] sourceData = new object[,] {
```

```
{ "Order ID", "Product",  "Category",   "Amount", "Date", "Country" },
{ 1,  "Carrots",  "Vegetables",  4270,    new DateTime(2018, 1, 6),  "United States" },
{ 2,  "Broccoli", "Vegetables",  8239,    new DateTime(2018, 1, 7),  "United Kingdom" },
{ 3,  "Banana",   "Fruit",       617,     new DateTime(2018, 1, 8),  "United States" },
{ 4,  "Banana",   "Fruit",       8384,    new DateTime(2018, 1, 10), "Canada" },
{ 5,  "Beans",    "Vegetables",  2626,    new DateTime(2018, 1, 10), "Germany" },
{ 6,  "Orange",   "Fruit",       3610,    new DateTime(2018, 1, 11), "United States" },
{ 7,  "Broccoli", "Vegetables",  9062,    new DateTime(2018, 1, 11), "Australia" },
{ 8,  "Banana",   "Fruit",       6906,    new DateTime(2018, 1, 16), "New Zealand" },
{ 9,  "Apple",    "Fruit",       2417,    new DateTime(2018, 1, 16), "France" },
{ 10, "Apple",    "Fruit",       7431,    new DateTime(2018, 1, 16), "Canada" },
{ 11, "Banana",   "Fruit",       8250,    new DateTime(2018, 1, 16), "Germany" },
{ 12, "Broccoli", "Vegetables",  7012,    new DateTime(2018, 1, 18), "United States" },
{ 13, "Carrots",  "Vegetables",  1903,    new DateTime(2018, 1, 20), "Germany" },
{ 14, "Broccoli", "Vegetables",  2824,    new DateTime(2018, 1, 22), "Canada" },
{ 15, "Apple",    "Fruit",       6946,    new DateTime(2018, 1, 24), "France" },
};

 // Set source data
 worksheet.Range["A20:F33"].Value = sourceData;
 worksheet.Range["A:F"].ColumnWidth = 10;

 // Add pivot table
 var pivotcache = workbook.PivotCaches.Create(worksheet.Range["A20:F33"]);
 var pivottable = worksheet.PivotTables.Add(pivotcache, worksheet.Range["A1"],
"pivottable1");

 // Setting number format for a field
 worksheet.Range["D21:D35"].NumberFormat = "$#,##0.00";

 // Configure pivot table's fields
 var field_Date = pivottable.PivotFields["Date"];
 field_Date.Orientation = PivotFieldOrientation.PageField;

 var field_Category = pivottable.PivotFields["Category"];
 field_Category.Orientation = PivotFieldOrientation.RowField;

 var field_Product = pivottable.PivotFields["Product"];
 field_Product.Orientation = PivotFieldOrientation.ColumnField;

 var field_Amount = pivottable.PivotFields["Amount"];
 field_Amount.Orientation = PivotFieldOrientation.DataField;
 field_Amount.NumberFormat = "$#,##0.00";

 var field_Country = pivottable.PivotFields["Country"];
 field_Country.Orientation = PivotFieldOrientation.RowField;

 // Create pivot style with name "CustomPivotstyle"
```

```
ITableStyle pivotStyle = workbook.TableStyles.Add("CustomPivotstyle");

// Set table style as pivot table style
pivotStyle.ShowAsAvailablePivotStyle = true;

pivotStyle.TableStyleElements[TableStyleElementType.PageFieldLabels].Interior.Color =
System.Drawing.Color.LightGreen;
pivotStyle.TableStyleElements[TableStyleElementType.PageFieldValues].Interior.Color =
System.Drawing.Color.LightGreen;

pivotStyle.TableStyleElements[TableStyleElementType.GrandTotalColumn].Interior.Color =
System.Drawing.Color.PowderBlue;
pivotStyle.TableStyleElements[TableStyleElementType.GrandTotalRow].Interior.Color =
System.Drawing.Color.PowderBlue;

pivotStyle.TableStyleElements[TableStyleElementType.HeaderRow].Interior.Color =
System.Drawing.Color.MistyRose;
pivotStyle.TableStyleElements[TableStyleElementType.FirstColumn].Interior.Color =
System.Drawing.Color.LightPink;

pivotStyle.TableStyleElements[TableStyleElementType.FirstRowStripe].Interior.Color =
System.Drawing.Color.SteelBlue;
pivotStyle.TableStyleElements[TableStyleElementType.SecondRowStripe].Interior.Color =
System.Drawing.Color.NavajoWhite;

// Set ShowTableStyleRowStripes as true
pivottable.ShowTableStyleRowStripes = true;

// Set pivot table style
pivottable.Style = pivotStyle;
worksheet.Range["A1:H16"].Columns.AutoFit();
worksheet.PageSetup.TopMargin = 30;
worksheet.PageSetup.LeftMargin = 30;

// Saving workbook to PDF
workbook.Save(@"82-PivotTableCustomStyle.pdf", SaveFileFormat.Pdf);
```

Note: While applying custom styles to the pivot table, it is important to note that if users apply a TableStyle whose **ShowAsAvailablePivotStyle** property is set to false, then the InvalidOperationException is thrown.

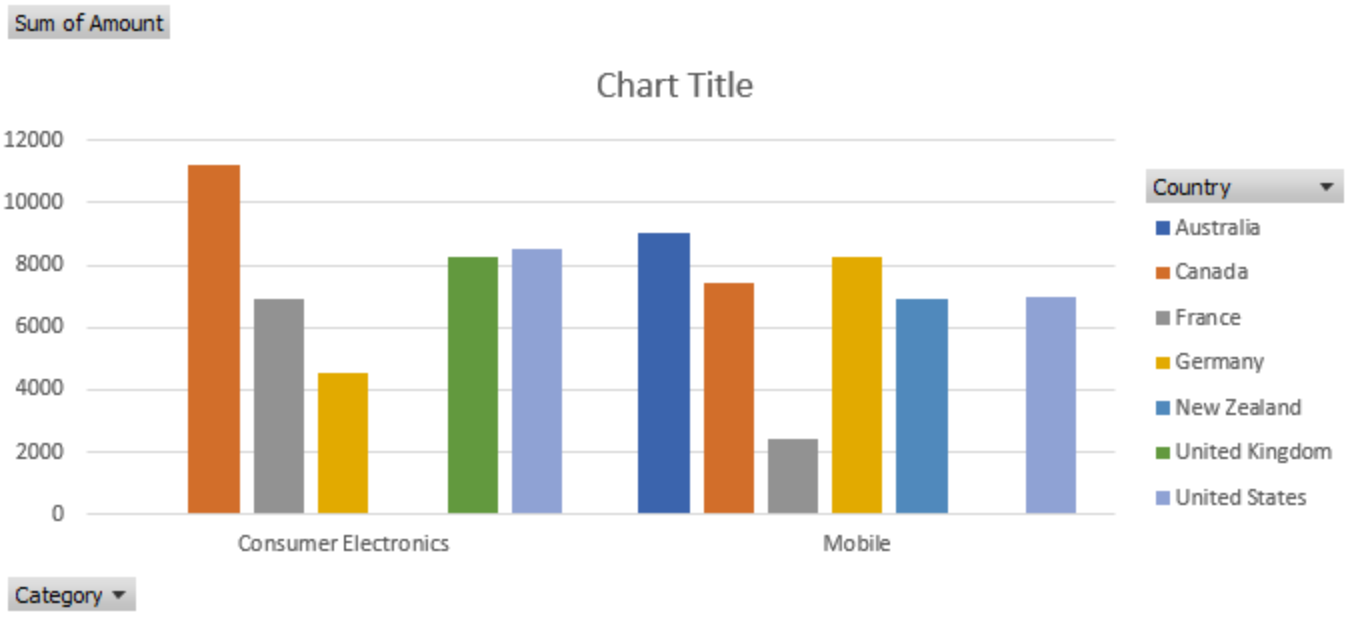
Pivot Chart

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

Sparklines can be understood as small, lightweight charts that are drawn inside cells to quickly visualize data for improved analysis. These tiny charts 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. These are particularly useful for analytical dashboards, presentations, business reports etc.

A 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.

Country	Profit% (Q1)	Profit% (Q2)	Profit% (Q3)	Profit% (Q4)	Line Sparkline	Column Sparkline	ColumnStacked100 Sparkline
Washington	56	-35	133	-78			
Wales	66	-66	21	-84			
New York	69	77	-33	83			
London	156	-35	133	78			
Thanes	68	-87	96	24			

DsExcel.NET allows you to highlight specific information and see how it varies over time using line, column, columnstacked100, and cascade sparklines. You can use Add method of the **SparklineGroups** class to add line, column, or columnstacked100 sparklines using **SparkType** enumeration. However, cascade sparkline is added using **CASCADESPARKLINE** formula. For more information about cascade sparkline, see SpreadJS Sparklines.

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

DsExcel .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
- Slicer Style
- Auto-Filter Table with Slicer
- Configure Slicer Layout
- Cut or Copy Slicer
- Duplicate Slicer
- Use Do Filter Operation
- Export Slicers

Add Slicer in Table

In DsExcel .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 DsExcel .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 InterfaceIPivotItem 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);
```

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 DsExcel .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"], "cate1", "Category",
200, 200, 100, 200);

//set slicer style to build in style.
slicer1.Style = workbook.TableStyles["SlicerStyleLight1"];
```

Modify Slicer with Custom Style

In DsExcel .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"], "cate1", "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"], "cate1", "Category",
200, 200, 100, 200);

slicer1.NumberOfColumns = 2;
//slicer1.ColumnWidth = 10;
slicer1.RowHeight = 50;
slicer1.DisplayHeader = false;
```

Auto-Filter Table with Slicer

In DsExcel, you can automatically filter tables with slicers via using the methods and properties of the **ISlicerCaches** interface. This helps in creating a new slicer cache for your table.

In order to auto-filter table with slicer, refer to the following example code.

C#

```
// Defining source data
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"},
};

// Initialize the workbook and fetch the default worksheet
var workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];

// Adding data to the table
worksheet.Range["A:F"].ColumnWidth = 15;
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 slicer for table
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1","Category",
20, 550, 100, 200);

// Apply table filter
// This synchronizes automatically to the slicer. The slicer1's selected item is
"Fruit".
worksheet.Range["A1:F16"].AutoFilter(2, "Fruit");
```

Configure Slicer Layout

DsExcel enables users to configure slicer layout using the properties of the **ISlicerCaches** interface that creates a new slicer cache for a table.

In order to configure slicer layout for a table, refer to the following example code.

Java

```
// Defining source data
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" },
};

// Initialize the workbook and fetch the default worksheet
var workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A:F"].ColumnWidth = 15;
```

```
// Adding data to the table
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, "Product", "productCache");

// Add slicer for table
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "product1",
"Product", 30, 550, 100, 200);

// Configure slicer layout.
slicer1.NumberOfColumns = 2;
slicer1.RowHeight = 25;
slicer1.DisplayHeader = false;
```

Cut or Copy Slicer

You can cut or copy slicer in a table using the properties and methods of the **ISlicerCaches** interface.

In order to copy slicer in table, refer to the following example code.

C#

```
// Defining source data
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" },
};

// Initialize the workbook and fetch the default worksheet
var workbook = new Workbook();
```

```
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A:F"].ColumnWidth = 15;

// Adding data to the table
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 slicer, slicer's range is Range["H3:J16"]
ISlicer slicer = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "Category",
30, 550, 100, 200);

// Range["H3:J16"] must contain slicer's range, copy a new shape to Range["K3:M16"]
worksheet.Range["H3:J16"].Copy(worksheet.Range["K3"]);
```

In order to cut slicer in table, refer to the following example code.

C#

```
// Defining source data
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" },
};

// Initialize the workbook and fetch the default worksheet
var workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A:F"].ColumnWidth = 15;

// Adding data to the table
```

```
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 slicer, slicer's range is Range["H3:J16"]
ISlicer slicer = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "Category",
30, 550, 100, 200);

// Range["H3:J16"] must contain slicer's range, cut a new shape to Range["K3:M16"]
worksheet.Range["H3:J16"].Cut(worksheet.Range["K3"]);
```

Duplicate Slicer

You can add duplicate slicer using DsExcel with the help of the **Add** method of the **ISlicerCaches** interface.

In order to add duplicate slicer in the worksheet, refer to the following example code.

C#

```
// Defining source data
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" },
};

// Initialize the workbook and fetch the default worksheet
var workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A:F"].ColumnWidth = 15;

// Adding data to the table
```

```
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 slicer for the table
ISlicer slicer = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "Category",
30, 550, 100, 200);

// Duplicate slicer
IShape newShape = slicer.Shape.Duplicate();
```

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();
```

Print

DsExcel.NET allows you to output a workbook or worksheet to a printer using the **PrintOut** method provided by **IWorkbook** and **IWorksheet** interface respectively. This method accepts object of **PrintOutOptions** class as a parameter. In the background, PrintOut method uses DsPdf to print workbook to a printer. Hence, you need to reference **DS.Documents.Imaging.Windows** to implement the feature.

> **Note**: The PrintOut method works only on the Windows environment. To print in other environments, you can export the document to PDF and then take a print of the converted document. For information on exporting a spreadsheet to PDF, see Export to PDF.

Print Options

DsExcel.NET provides **PrintOutOptions** class that contains various properties and events to set printing options such as printer name, number of copies, double-sided printing, first and last page to be printed etc. To print a document with specified printing options, you need to create an instance of the PrintOutOptions class, define the options for that instance and pass that instance while calling the PrintOut method.

Refer to the following example code to generate multiple copies of double-sided print for a workbook:

C#

```
// Create a workbook instance.
var workbook = new Workbook();

// Add data for table.
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}
};
// Create a worksheet instance.
IWorksheet worksheet = workbook.Worksheets[0];
```

```
// Bind data with table.
worksheet.Range["A1:F7"].Value = data;
worksheet.Range["A:F"].ColumnWidth = 12;

// Add table.
worksheet.Tables.Add(worksheet.Range["A1:F7"], true);

// Create an instance of PrintOutOptions class.
PrintOutOptions options = new PrintOutOptions();

// Set name of the target printer.
options.ActivePrinter = "[Real printer name]";

// Print 3 copies.
options.Copies = 3;

// Print on double sides vertically.
options.Duplex = Duplex.Vertical;

// Print this workbook.
workbook.PrintOut(options);

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

Note: The double-sided printing option only works if the printer has double-sided printing functionality.

Page Setup

DsExcel .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 DsExcel .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 DsExcel .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;
```

You can also use **IsAutoFirstPageNumber** property of the IPageSetup interface to set the first page number to “Auto”.

C#

```
// Set first page number to auto.
worksheet2.PageSetup.IsAutoFirstPageNumber = true;
```

Configure Page Breaks

DsExcel .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 DsExcel .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 to 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.

DsExcel .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 DsExcel .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

DsExcel 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.

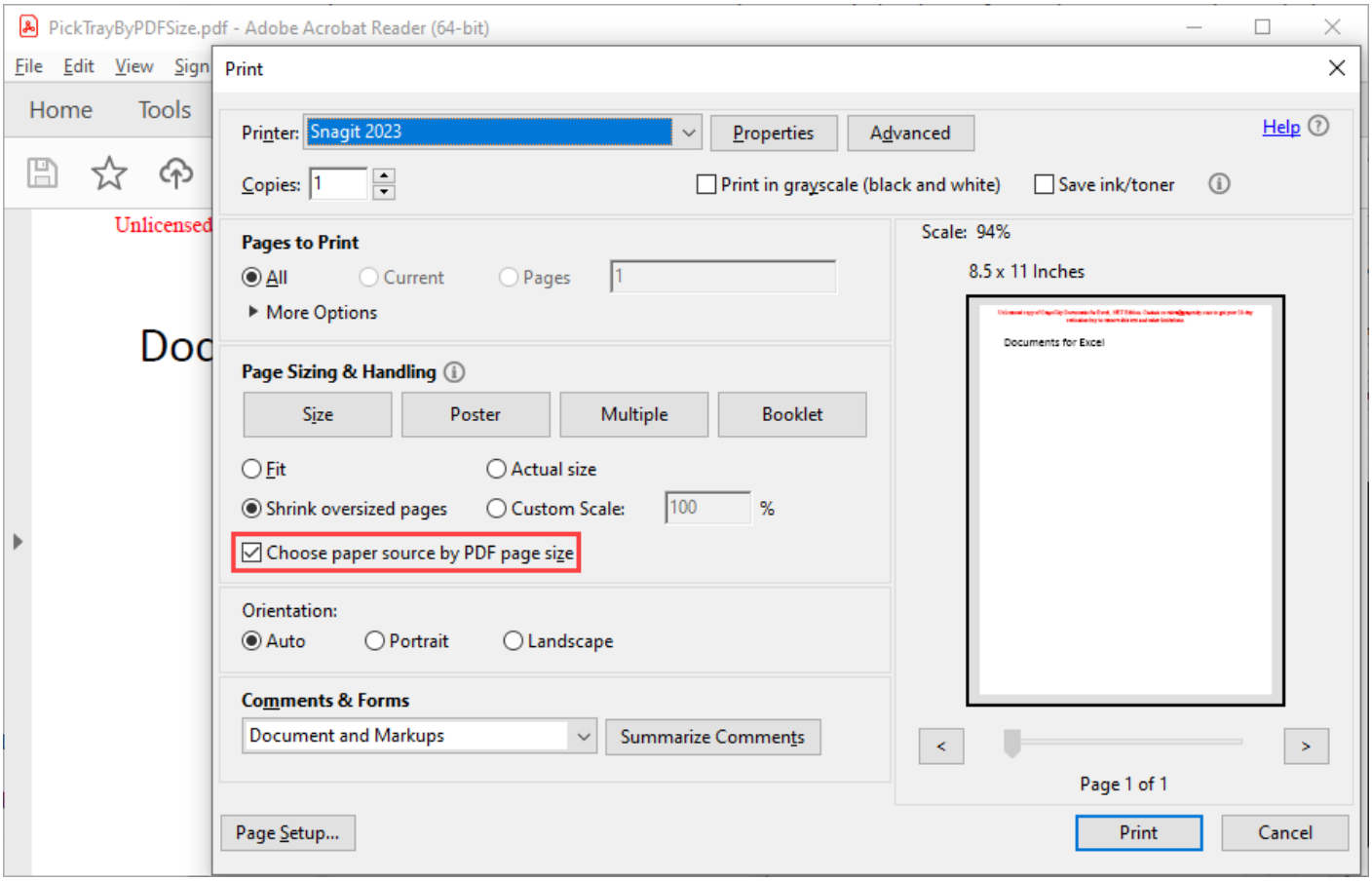
Refer to the following example code to configure paper source while printing a PDF document.

C#

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

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

```
worksheet.Range["A1"].Value = "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

DsExcel 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, DsExcel 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 DsExcel. 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;
}
```

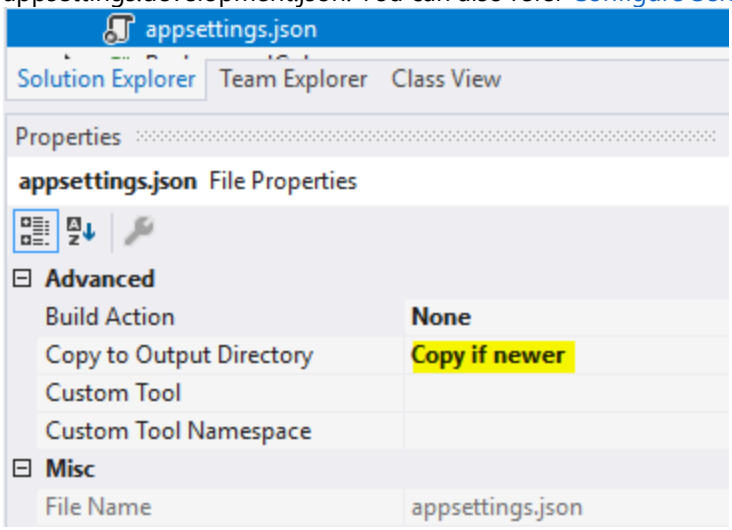
3. Attach the logger factory to workbook as shown below:

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

DsExcel 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 DsExcel log level to "Information".

JSON

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning",
      "GrapeCity.Documents.Excel": "Information"
```

```
    }
  }
}
```

Similarly, the following setting in appsettings.json sets the DsExcel 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.RollingFil
   - 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();
    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.
```

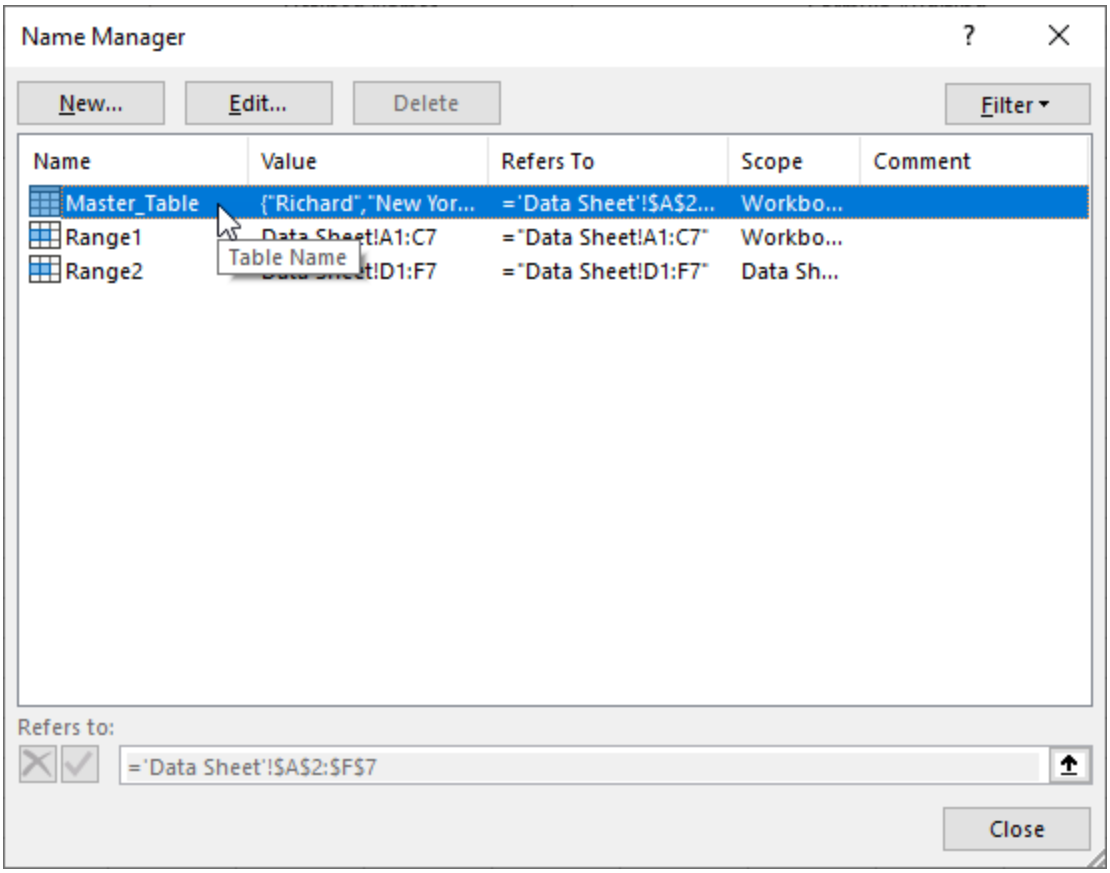
Defined Names

Defined names refer to names given to constants, tables, cell ranges, or formulas so that you can refer to them in a formula without making it too complex to understand. The defined names are especially useful in complex calculations, such as calculating taxes for a whole financial year, where you will have difficulty finding and understanding the cells having different investments, taxable incomes, etc.

DsExcel supports defined names with the help of **Names** property in **IWorkbook** and **IWorksheet** interfaces and **Name** property in **ITable** interface.

Name a Table

Name a table by using Name property of ITable interface. The scope of the table name is workbook by default, as tables are created in workbook scope only. This name appears in Excel's Name Manager, as shown below.



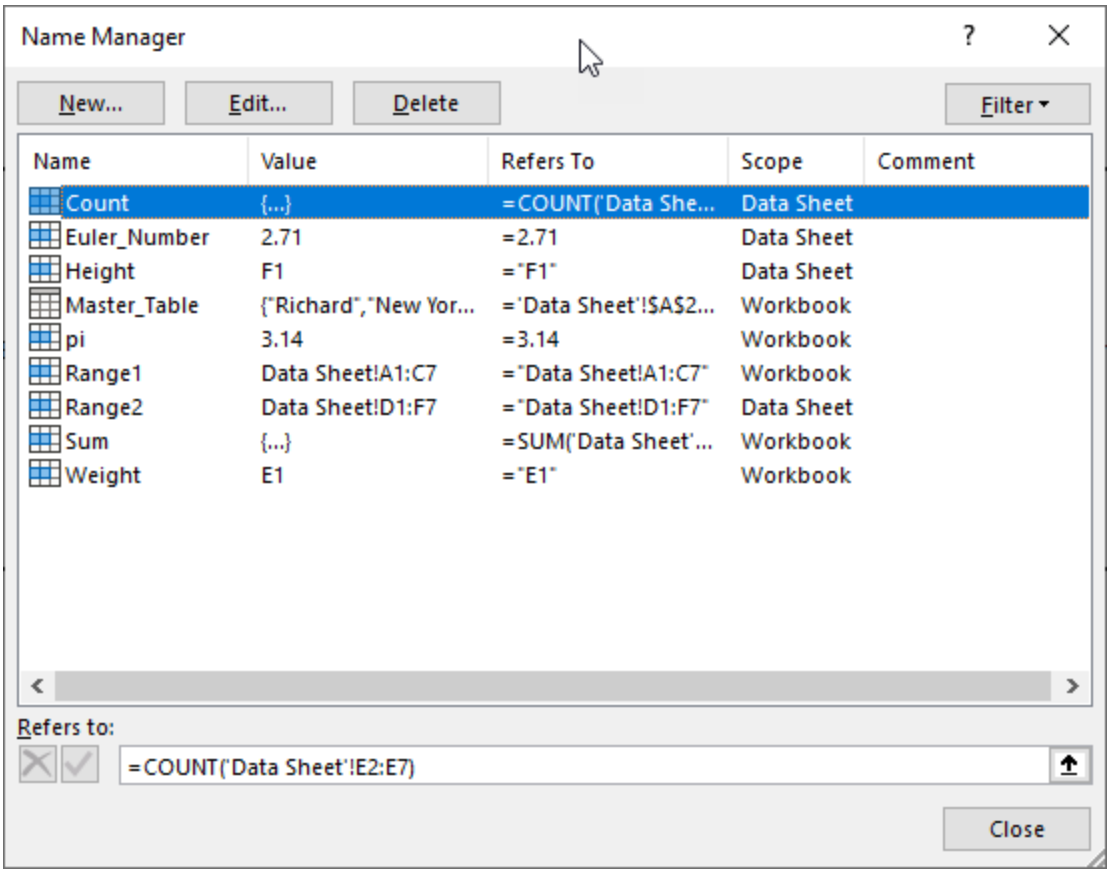
Refer to the following example code to name the table.

C#

```
// Name a table.
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F7"], true);
table.Name = "Master Table";
```

Name a Cell Range, Formula, and Constant

Name a cell range, formula, and constant using the Names property with Workbook and Worksheet objects. This property adds an IName object storing the name and referenced cell, formula, or constant. The name added to a workbook object is stored in the workbook scope, while the name added to a worksheet object is saved in the worksheet scope. It appears in Excel's Name Manager, as shown below.



Refer to the following example code to name a cell range, a formula, and a constant in workbook scope.

C#

```
// Name a range in wokrbook scope.
workbook.Names.Add("Range1", "Data Sheet!A1:C7");

// Name formula in workbook scope.
workbook.Names.Add("Sum", "=SUM(F2:F7)");

// Name a constant in workbook scope.
workbook.Names.Add("pi", "3.14");

// Name a cell in workbook scope.
workbook.Names.Add("Weight", "E1");
```

Refer to the following example code to name a cell range, a formula, and a constant in worksheet scope.

C#

```
// Name a range in worksheet scope.
workbook.Worksheets[worksheet.Index].Names.Add("Range2", "Data Sheet!D1:F7");

// Name formula in worksheet scope.
```

```
workbook.Worksheets[worksheet.Index].Names.Add("Count", "=COUNT(E2:E7)");

// Name a constant in worksheet scope.
workbook.Worksheets[worksheet.Index].Names.Add("Euler_Number", "2.71");

// Name a cell in worksheet scope.
workbook.Worksheets[worksheet.Index].Names.Add("Height", "F1");
```

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. DsExcel provides templates to create highly effective and well-designed Excel reports.

Some of the powerful features provided by DsExcel templates are as follows:

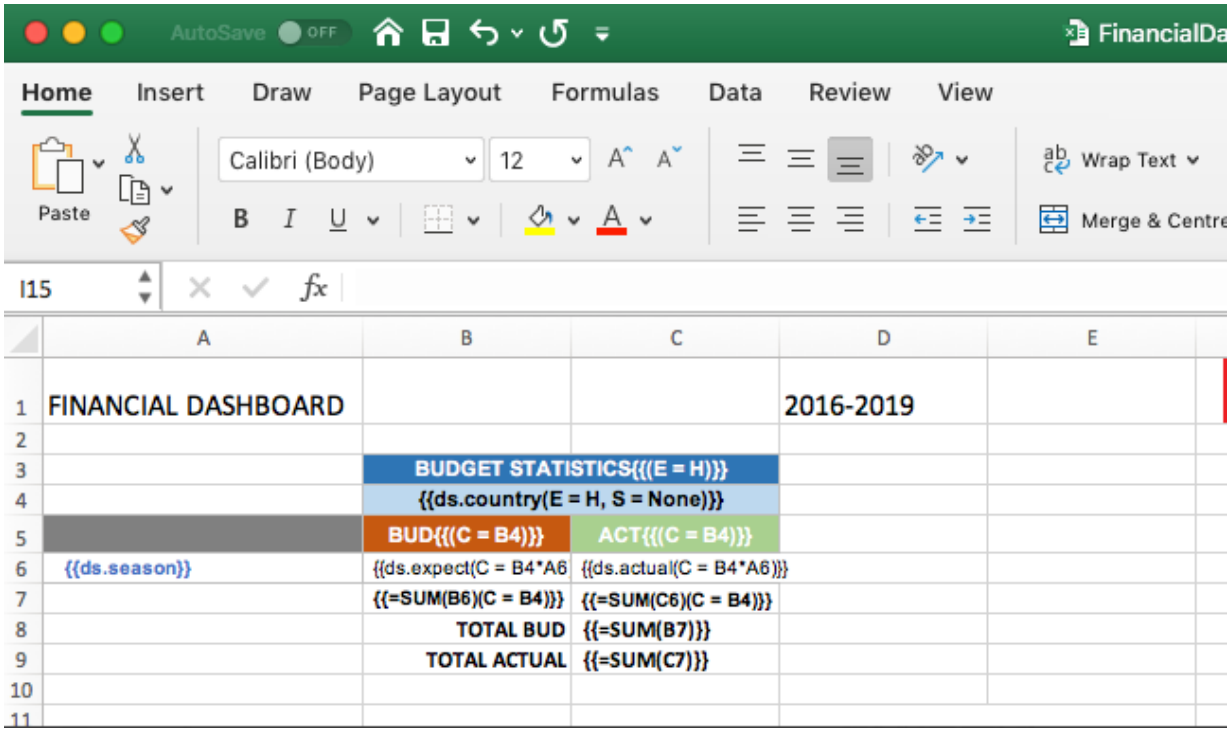
- **Flexible**: DsExcel templates have a highly flexible template syntax and API to bind Excel documents to data. It follows easy data population rules in fields.
- **Efficient**: DsExcel templates provide extended reusability. This means the templates can be used with minor modifications, or as it is time and again, saving both time and effort.
- **Multi-platform**: DsExcel templates are supported on Windows, Linux and macOS.
- **Multi-domain**: DsExcel templates cater to complex use-cases to create Excel reports for any scenario.

The DsExcel 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 DsExcel 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.



- 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 fileds '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
	{{ds.dpt.name(E=H, R=C2:D6, G=list)}}	
	MANAGER	{{ds.dpt.mgr}}
	BUDGET	{{ds.dpt.bud}}
	Employee	**Salary**
	{{ds.dpt.emp.name(G=list)}}	{{ds.dpt.emp.salary}}

The Excel report generated from the Department & Budget template is given below:

C	D	E	F
Marketing		**Sales**	
MANAGER	Carl Sommerset	MANAGER	Kelly Johnson
BUDGET	$ 3,54,586	BUDGET	$ 2,37,721
Employee	**Salary**	**Employee**	**Salary**
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		ACME INC.
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 ACME INC.

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 DsExcel.

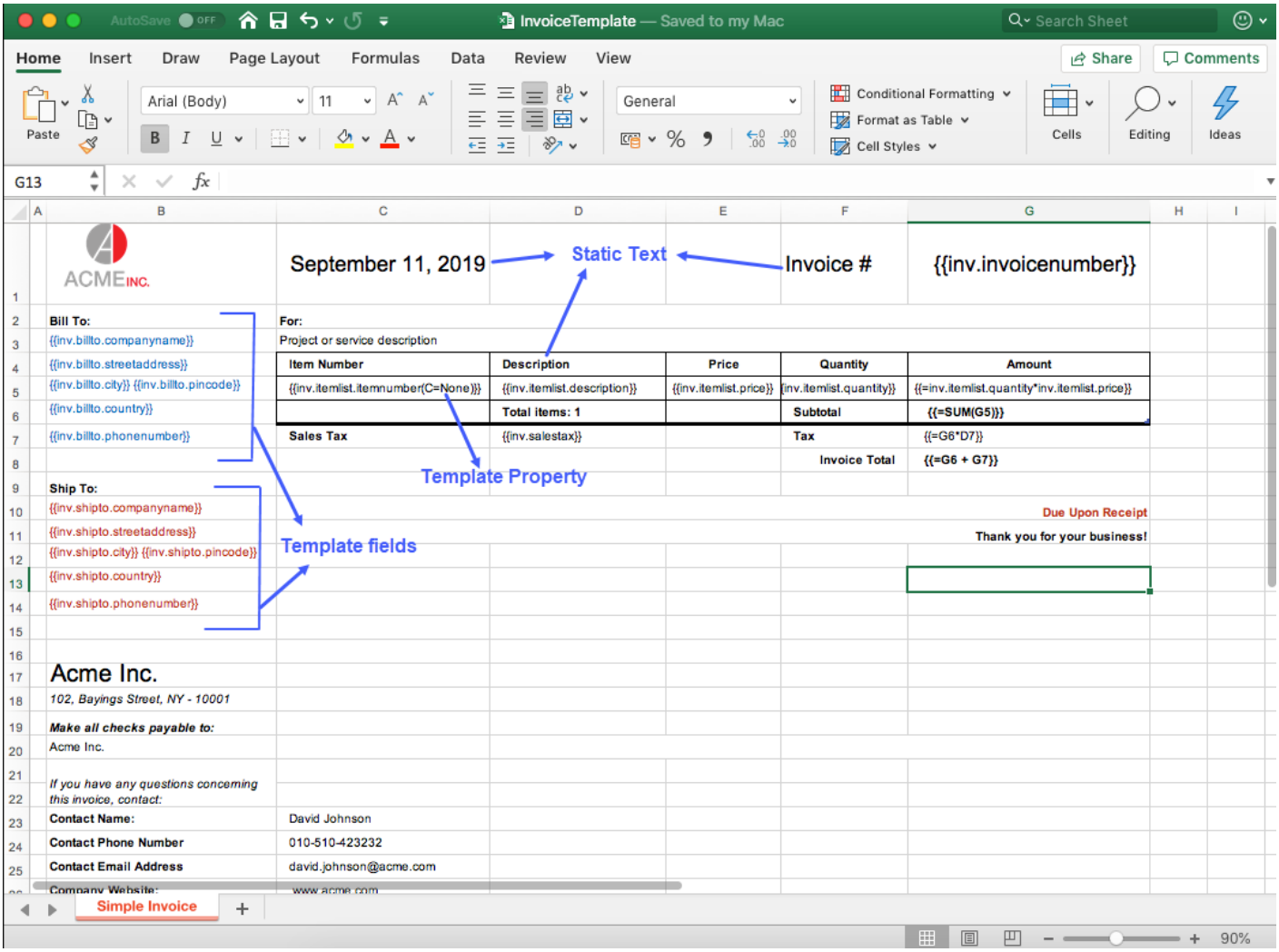
The first step to configure a template is to create a template layout in Excel, which is a pre requisite to generate Excel reports. This layout defines the outline of how the final report will look like and can include static text, data bound fields and other template properties.

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.

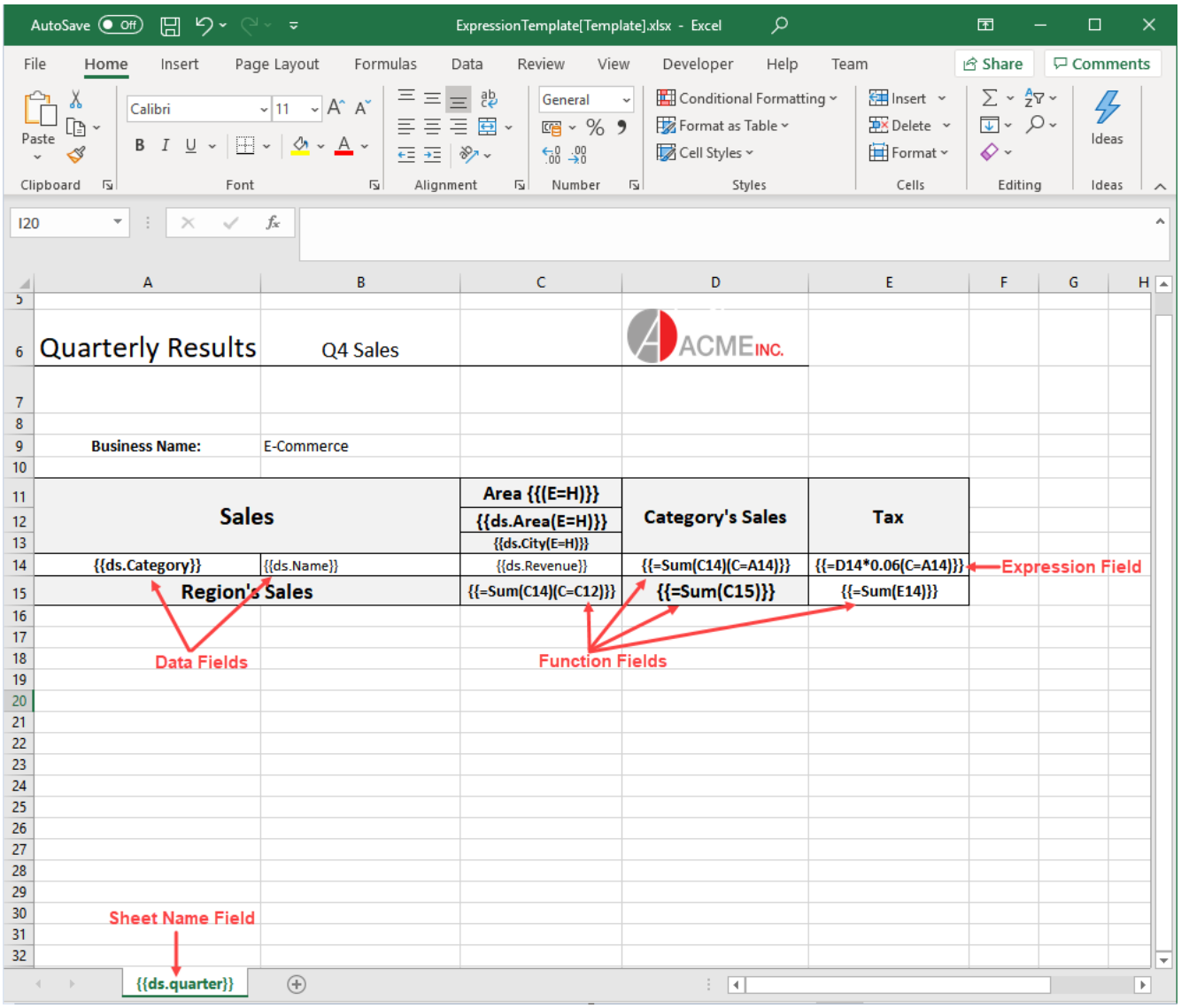
Templates, in DsExcel, also supports:

- Conditional Formatting

- Global Settings
- Fixed Layout
- Default Values in Template Cells
- PDF Form Builder
- Charts
- Tables
- Sparklines
- Paginated Templates

Template Fields

A template layout can contain various fields bound to the data source. The below image shows different template fields:



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 '.'

Student Report	ACME INC.
Student Name	{{report.student.name(R=A3:B9)}}
Father's Name	{{report.student.family.father.name(S=None,E=H)}}
Occupation	{{report.student.family.father.occupation(E=H)}}
Mother's Name	{{report.student.family.mother.name(E=H)}}
Occupation	{{report.student.family.mother.occupation(E=H)}}

- **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: {{emp.name}}	{{emp.id{R=A4:F4, S=None}}}	Belongs to {{emp.department}} of our company	{{department.name.hodname}}	{{emp.department.id}}	{{emp.department.loc}} 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		ACME INC.
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 ACME INC.

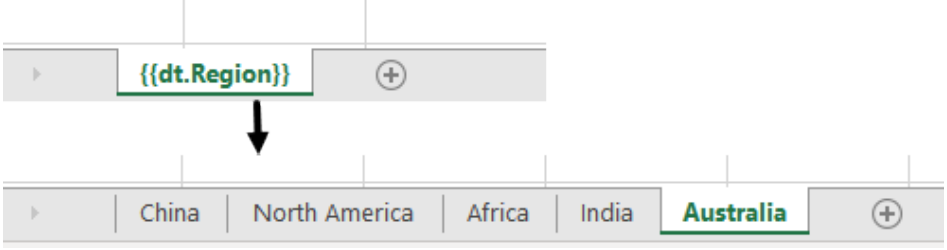
Business Name: E-Commerce

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**

Sheet Name

DsExcel 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.

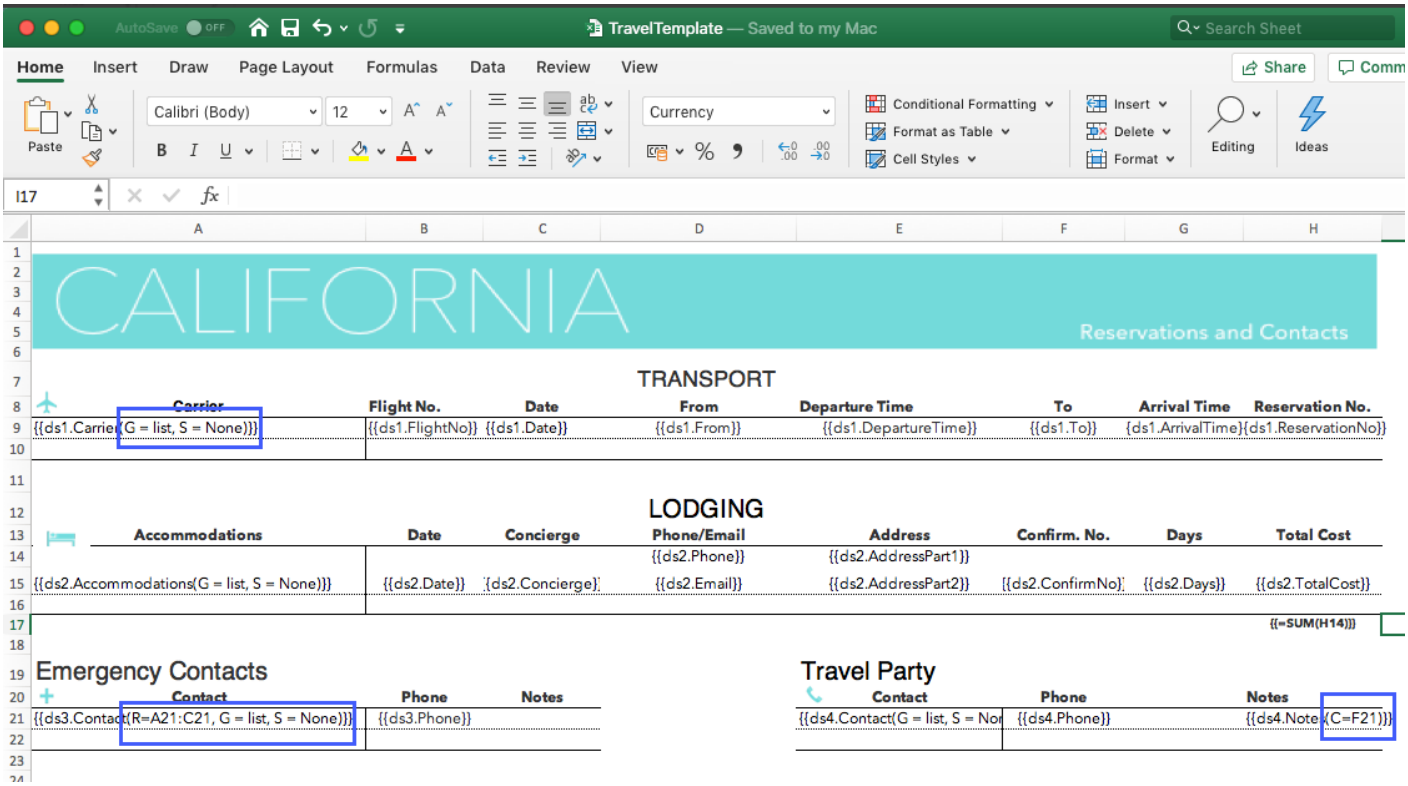
For example, if we specify {{dt.Region}} as the sheet name and the data source contains data for 5 regions, the final report will consist of 5 worksheets and the individual sheet will contain data for a specific region.



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:



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)	
Team	Name
{{ds.Team}}	{{ds.Name}}

Repeat group	
Team	Name
{{ds.Team(G=R)}}	{{ds.Name}}

Normal group	
Team	Name
{{ds.Team(G=N)}}	{{ds.Name}}

List group	
Team	Name
{{ds.Team(G=L)}}	{{ds.Name}}

Merge group(default)	
Team	Name
New York	Hellen
	Hunter
	Jim
	Phillip
Seattle	Bob
	Jaguar
	Tommy

Repeat group	
Team	Name
New York	Hellen
New York	Hunter
New York	Jim
New York	Phillip
Seattle	Bob
Seattle	Jaguar
Seattle	Tommy

Normal group	
Team	Name
New York	Hellen
	Hunter
	Jim
	Phillip
Seattle	Bob
	Jaguar
	Tommy

List group	
Team	Name
Seattle	Bob
Seattle	Tommy
Seattle	Jaguar
New York	Phillip
New York	Hunter
New York	Hellen
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.

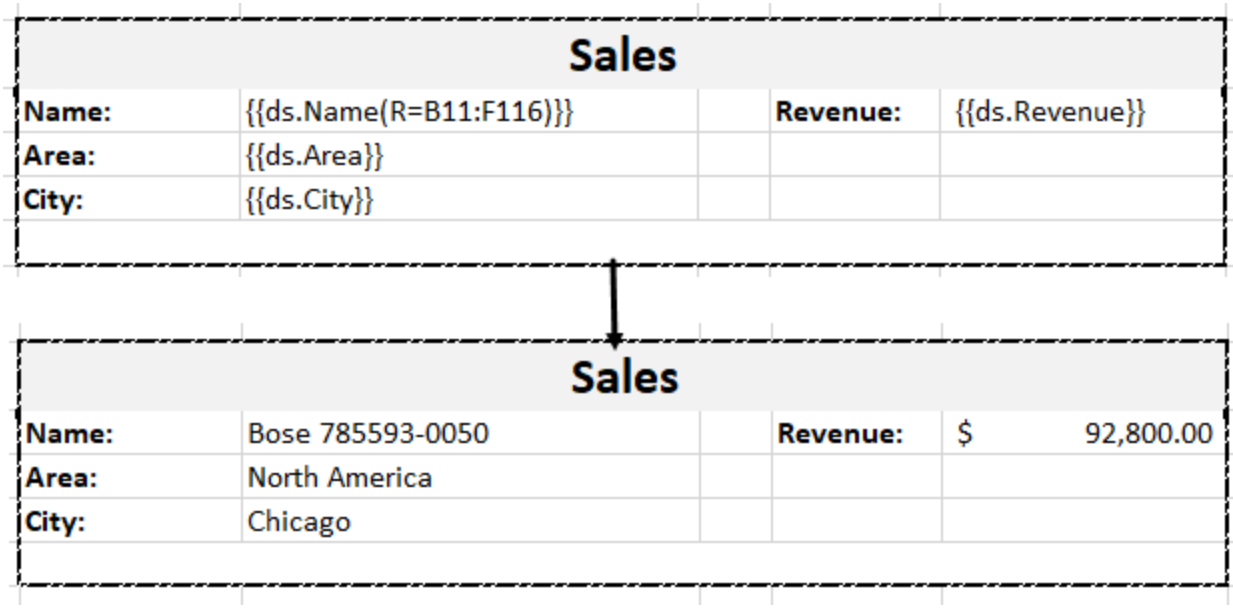
Value: Cell range

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.

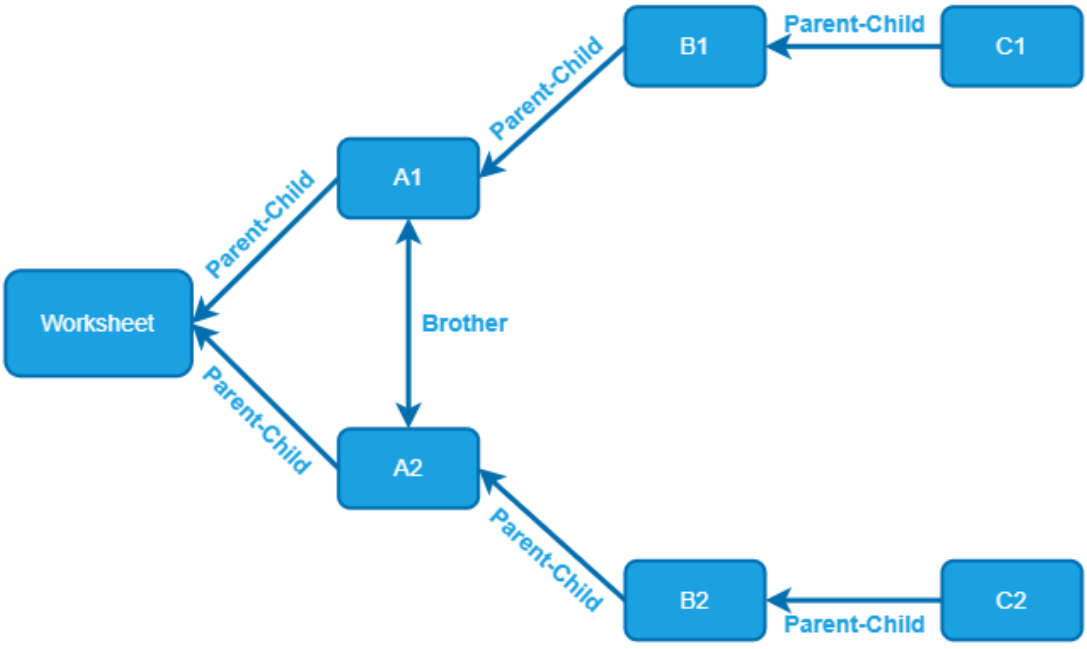


Sort

The Sort property ('S') defines the sorting type within the template. This property applies to a single or multiple columns, determined by the respective cell values. The sorting functionality extends beyond sorting in ascending and descending order; it allows the application of custom sorting rules to single or multiple columns.

DsExcel implements sorting by observing the following basic rules:

1. DsExcel processes the sort function of a template by distinguishing the cells into template cells and instance cells. Template cells are the cells in the template file, while instance cells are the cells that are created after the template has been processed.
2. DsExcel treats the relationship between the template cells as a parent-child relationship. A similar relationship applies between the instance cells.
3. DsExcel sorts the cells between brother instance cells but cannot change the parent-child relationship. Hence, the instance cells to be sorted must fulfill the following two conditions:
    a. They come from the same template cell.
    b. They have a common parent cell.



For example, let's consider the following data source:

	A	B	C	D
1	**Area**	**City**	**Product**	**Revenue**
2	North America	Chicago	Bose 785593-0050	92800
3	North America	New York	Bose 785593-0050	92800
4	South America	Santiago	Bose 785593-0050	19550
5	North America	Chicago	Canon EOS 1500D	98650
6	North America	Minnesota	Canon EOS 1500D	89110
7	South America	Santiago	Canon EOS 1500D	459000
8	North America	Chicago	Haier 394L 4Star	367050
9	South America	Quito	Haier 394L 4Star	729100
10	South America	Santiago	Haier 394L 4Star	578900
11	North America	Fremont	IFB 6.5 Kg FullyAuto	904930
12	South America	Buenos Aires	IFB 6.5 Kg FullyAuto	673800
13	South America	Medillin	IFB 6.5 Kg FullyAuto	82910
14	North America	Chicago	Mi LED 40inch	550010
15	North America	Minnesota	Mi LED 40inch	1784702
16	South America	Santiago	Mi LED 40inch	102905
17	North America	Chicago	Sennheiser HD 4.40-BT	178100
18	South America	Quito	Sennheiser HD 4.40-BT	234459
19	North America	Minnesota	Iphone XR	1734621
20	South America	Santiago	Iphone XR	109300
21	North America	Chicago	OnePlus 7Pro	499100
22	South America	Quito	OnePlus 7Pro	215000
23	North America	Minnesota	Redmi 7	81650
24	South America	Quito	Redmi 7	276390
25	North America	Minnesota	Samsung S9	896250
26	South America	Buenos Aires	Samsung S9	896250
27	South America	Quito	Samsung S9	716520

If you use the following template, the cells in column D cannot be sorted because the Sort property is restrained by context cell A1 (i.e., no common parent cell).

	A	B	C	D
1	{{ds.Area(G=L)}}	{{ds.City}}	{{ds.Product}}	{{ds.Revenue(C=A1,S=desc)}}
2				

Hence, you can only add the Sort property to cell A1. Then the template will be as follows:

	A	B	C	D
1	{{ds.Area(G=L,S=(D1 desc))}}	{{ds.City}}	{{ds.Product}}	{{ds.Revenue}}
2				

Result:

	A	B	C	D
1	North America	Minnesota	Mi LED 40inch	1784702
2	North America	Minnesota	Iphone XR	1734621
3	North America	Fremont	IFB 6.5 Kg FullyAuto	904930
4	North America	Minnesota	Samsung S9	896250
5	South America	Buenos Aires	Samsung S9	896250
6	South America	Quito	Haier 394L 4Star	729100
7	South America	Quito	Samsung S9	716520
8	South America	Buenos Aires	IFB 6.5 Kg FullyAuto	673800
9	South America	Santiago	Haier 394L 4Star	578900
10	North America	Chicago	Mi LED 40inch	550010
11	North America	Chicago	OnePlus 7Pro	499100
12	South America	Santiago	Canon EOS 1500D	459000
13	North America	Chicago	Haier 394L 4Star	367050
14	South America	Quito	Redmi 7	276390
15	South America	Quito	Sennheiser HD 4.40-BT	234459
16	South America	Quito	OnePlus 7Pro	215000
17	North America	Chicago	Sennheiser HD 4.40-BT	178100
18	South America	Santiago	Iphone XR	109300
19	South America	Santiago	Mi LED 40inch	102905
20	North America	Chicago	Canon EOS 1500D	98650
21	North America	New York	Bose 785593-0050	92800
22	North America	Chicago	Bose 785593-0050	92800
23	North America	Minnesota	Canon EOS 1500D	89110
24	South America	Medillin	IFB 6.5 Kg FullyAuto	82910
25	North America	Minnesota	Redmi 7	81650
26	South America	Santiago	Bose 785593-0050	19550

You can only sort cells that have the Sort property; cells without the Sort property will not be sorted. This means that in the above data source, sorting the value of A1 based on the value of D1 will only sort A1, not D1.

Sorting can be performed in three ways:

- Enum
- Array
- Expression

Value: *Enum*

S=Asc (default value): Ascending

S=Desc : Descending

S=None: None

Value: Array

S={"X", "Y","Z"}

Value: Expression

S=(Cell A Asc {"X", "Y","Z"}, Cell B Desc)

Example 1: Sorting Single Column

{{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

Example 2: Sorting Multiple Columns

{{ds.OrderID(S=(C12,D12 desc),G=List)}}

The below image shows how the Order ID column is sorted based on C12 and D12 cells. You can also download the **Excel template layout** used in below example.

Business Name:	E-Commerce		
Order ID	**Customer**	**Date**	**Sales**
{{ds.OrderID(S=(C12,D12 desc),G=List)}}	{{ds.Customer}}	{{ds.Date}}	{{ds.Sales}}
Region's Sales			**{{=sum(D12)}}**

Business Name:	E-Commerce		
Order ID	**Customer**	**Date**	**Sales**
10002	William Smith	20-01-2024	$5,620.25
10003	Sophia Miller	20-01-2024	$3,354.50
10001	Noah Taylor	20-01-2024	$1,620.25
10004	Noah Taylor	23-01-2024	$3,563.00
10005	Ava Johnson	23-01-2024	$1,500.00
10007	Emma Brown	24-01-2024	$8,865.50
10008	Isabella Taylor	24-01-2024	$4,332.05
10006	Ava Davis	24-01-2024	$3,500.00
10009	Noah Anderson	25-01-2024	$5,568.54
10011	Liam Taylor	26-01-2024	$6,659.56
10012	Olivia Smith	26-01-2024	$4,321.05
10010	William Smith	26-01-2024	$2,659.56
10013	Emma Brown	27-01-2024	$6,521.52
10015	Liam Johnson	28-01-2024	$5,542.02
10014	Ava Jones	28-01-2024	$4,321.05
Region's Sales			**$67,948.85**

Example 3: Sorting using Custom Rule

{{ds.City(S=(A12 desc {"New York", "Chicago", "Minnesota", "Santiago", "Fremont", "Quito", "Medillin", "Buenos Aires"}))}}

The below image shows how the City column is sorted based on custom sorting rule. You can also download the **Excel template layout** used in below example.

Business Name:	E-Commerce		

City	Name	Category	Sales
{{ds.City(S=(A12 desc {"New York", "Chicago", "Minnesota", "Santiago", "Fremont", "Quito", "Medillin", "Buenos Aires"})}}	{{ds.Name}}	{{ds.Category}}	{{ds.Revenue}}
City's Sales			**{{=sum(D12)}}**

Business Name:	E-Commerce		

City	Name	Category	Sales
Buenos Aires	IFB 6.5 Kg FullyAuto	Consumer Electronics	$6,73,800.00
	Samsung S9	Mobile	$8,96,250.00
Medillin	IFB 6.5 Kg FullyAuto	Consumer Electronics	$82,910.00
Quito	Haier 394L 4Star	Consumer Electronics	$7,29,100.00
	OnePlus 7Pro	Mobile	$2,15,000.00
	Redmi 7	Mobile	$2,76,390.00
	Samsung S9	Mobile	$7,16,520.00
	Sennheiser HD 4.40-BT	Consumer Electronics	$2,34,459.00
Fremont	IFB 6.5 Kg FullyAuto	Consumer Electronics	$9,04,930.00
Santiago	Bose 785593-0050	Consumer Electronics	$19,550.00
	Canon EOS 1500D	Consumer Electronics	$4,59,000.00
	Haier 394L 4Star	Consumer Electronics	$5,78,900.00
	Iphone XR	Mobile	$1,09,300.00
	Mi LED 40inch	Consumer Electronics	$1,02,905.00
Minnesota	Canon EOS 1500D	Consumer Electronics	$89,110.00
	Iphone XR	Mobile	$17,34,621.00
	Mi LED 40inch	Consumer Electronics	$17,84,702.00
	Redmi 7	Mobile	$81,650.00
	Samsung S9	Mobile	$8,96,250.00
Chicago	Bose 785593-0050	Consumer Electronics	$92,800.00
	Canon EOS 1500D	Consumer Electronics	$98,650.00
	Haier 394L 4Star	Consumer Electronics	$3,67,050.00
	Mi LED 40inch	Consumer Electronics	$5,50,010.00
	OnePlus 7Pro	Mobile	$4,99,100.00
	Sennheiser HD 4.40-BT	Consumer Electronics	$1,78,100.00
New York	Bose 785593-0050	Consumer Electronics	$92,800.00
City's Sales			**$1,24,63,857.00**

Page Break

The page break property specifies whether to add a new page after a field or not. It is determined by the location of the template cell in the generated report.

- If the template cell in the generated report is located in the first row and the first column then no page break is added.
- If the template cell in the generated report is located in the first column, horizontal page break is added.
- If the template cell in the generated report is located in the first row, vertical page break is added
- If the template cell in the generated report is located in any other location than the first row and first column, both horizontal and vertical page breaks area added

Value: Boolean

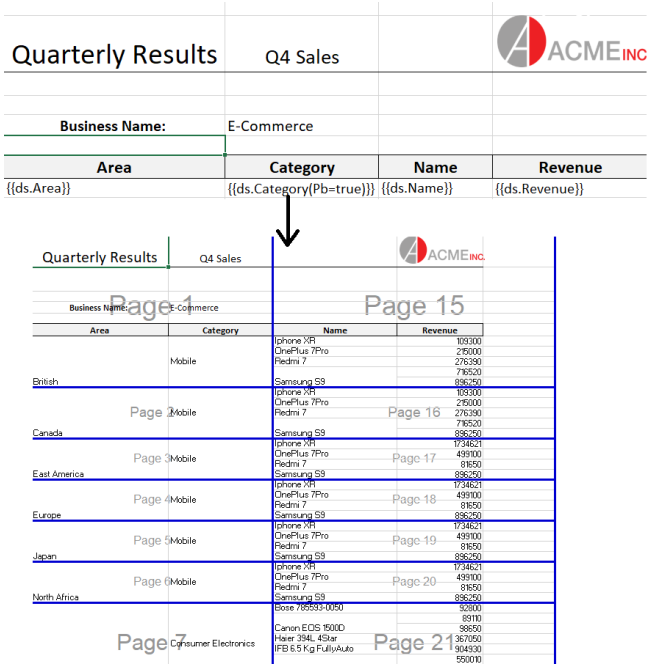
Pagebreak=True

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.



Note: In pagination mode, the **Pagebreak** template property is ignored.

Image

The image property specifies whether or not to add an image. If the value is true, you can also specify the image width and height or maintain the aspect ratio.

The width and height specify the custom dimensions of an image in a cell, whereas keepaspect fits the image size to the cell size and maintains the aspect ratio as much as possible. When specifying width and height, you also need to specify the unit of the dimension, either pt or px.

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.

Image.keepaspect= True

Image.keepaspect= False (Default value)

You can also use the following abbreviations: img, w, h, and ka for image, width, height, and keepaspect, respectively, to create the syntax.

> **Note:** Keepaspect takes precedence over width and height settings when you specify both properties of the image.

Example

{{ds.icon(Image=true)}}

{{ds.icon(Image=true, Image.width=150px)}}

{{ds.icon(Image=true, Image.height=150px)}}

{{ds.icon(Image=true, Image.keepaspect=true)}}

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.

TREADSTONE

BIKES

{{ds.BikeType(R=A5:A6,S=None)}}

{{ds.BikeSeries.Name(R=A7:N9)}}

Origin: s.CountryImage(image=true,image.keepaspect=true)}}

{{ds.BikeSeries.Description}}

{{ds.BikeSeries.BikeImage(image=true,image.w=173pt,image.h=96pt)}}

Product No	Product	Color	Size	Weight	Dealer	List Price
{{ds.BikeSeries.Items.ProductNo(C=A	{{ds.BikeSeries.Items.P	Series.Item	Series.Iter	eries.Items	ms.Dealer}}	eries.Items.ListPrice}}

TREADSTONE

BIKES

Mountain Bikes

Mountain-100

Origin:

Top-of-the-line competition mountain bike. Performance-enhancing options include the innovative HL Frame, super-smooth front suspension, and traction for all terrain.

Product No	Product	Color	Size	Weight	Dealer	List Price
BK-M82S-38	Mountain-100 Silver, 38	Silver	38	20.35	$1,912.15	$3,399.99
BK-M82B-38	Mountain-100 Black, 38	Black	38	20.35	$1,898.09	$3,374.99

For information regarding template properties related to pagination, see Pagination Properties and Functions.

> **Note:** All the above-mentioned template properties are case-insensitive, which means DsExcel ignores cases and matches values regardless of their lower or upper case letters.

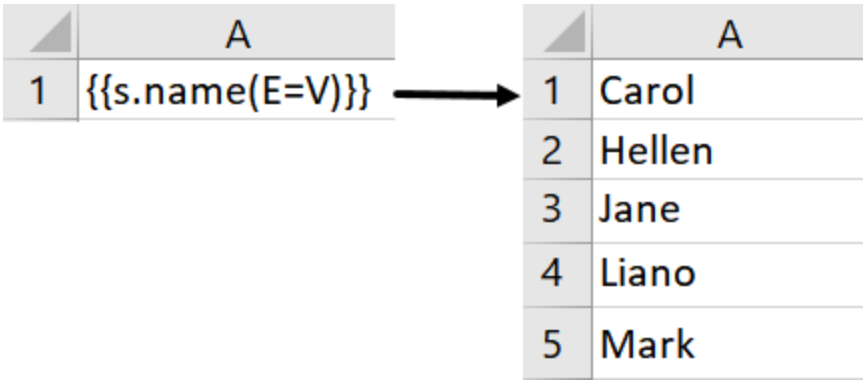
Cell Expansion

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:

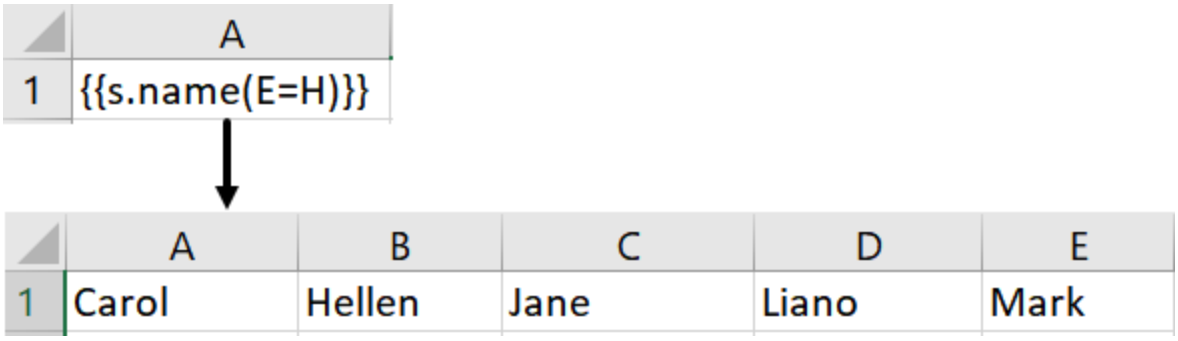
Vertical Expansion

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.



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:



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:

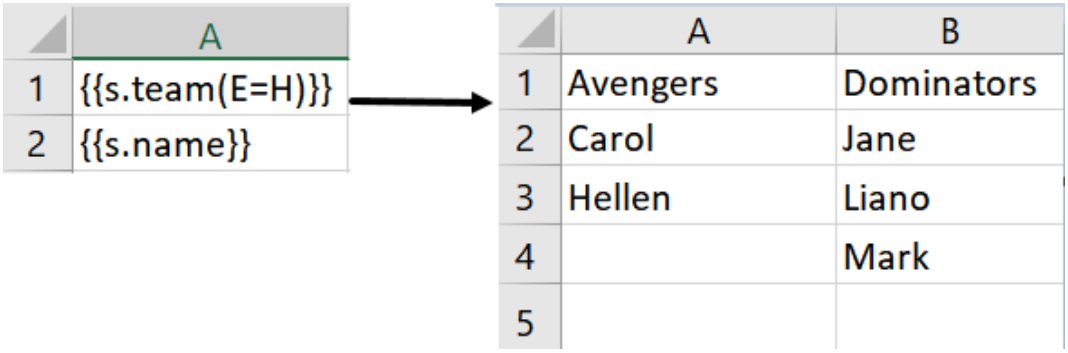
- **Filtering Relationship**: The data in the cell is filtered using data of the context cell as the filter condition. For example, in the below image, the data in the 'name' cell is filtered corresponding to the data in its context cell:

	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:



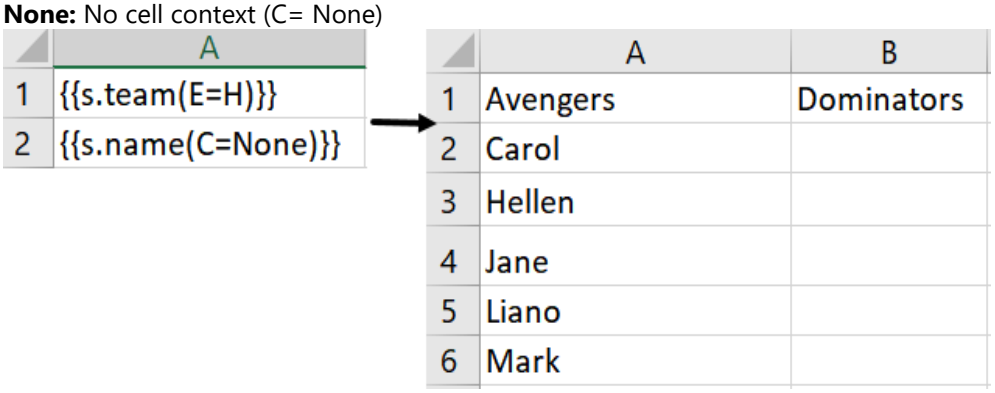
	A
1	{{s.team(E=H)}}
2	{{s.name}}

	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:

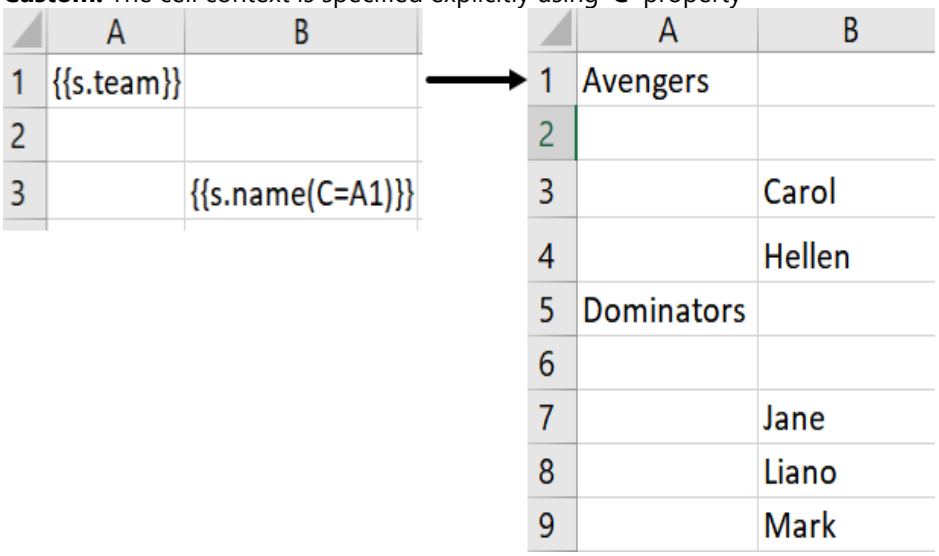
- **None:** No cell context (C= None)



	A
1	{{s.team(E=H)}}
2	{{s.name(C=None)}}

	A	B
1	Avengers	Dominators
2	Carol	
3	Hellen	
4	Jane	
5	Liano	
6	Mark	

- **Custom:** The cell context is specified explicitly using '**C'** property



	A	B
1	{{s.team}}	
2		
3		{{s.name(C=A1)}}

	A	B
1	Avengers	
2		
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.

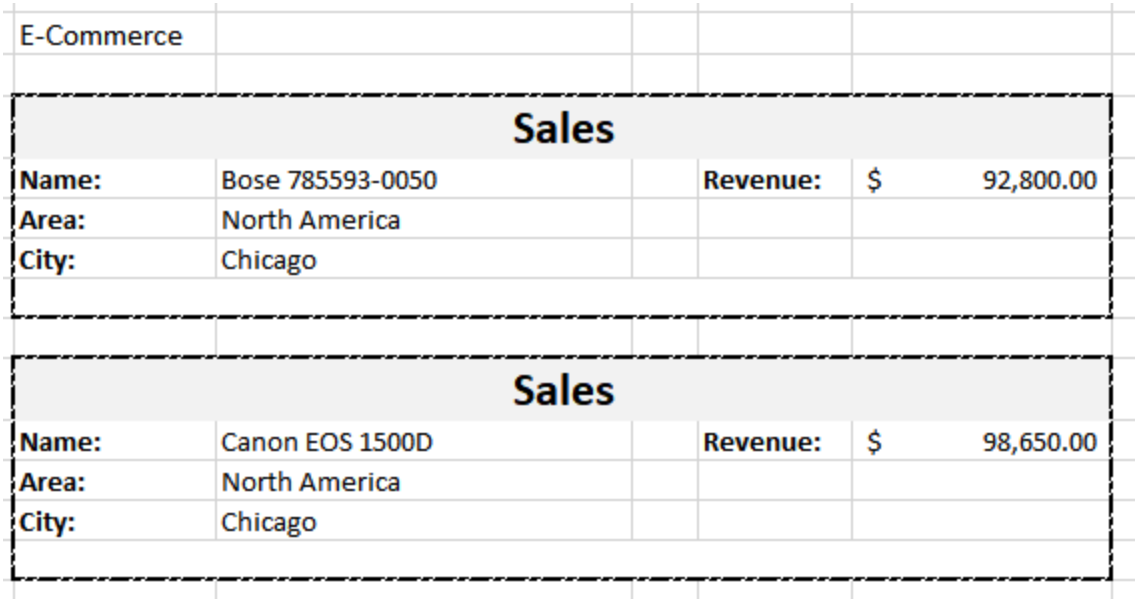
The **Fallback context** can be defined in a cell using the Range property, in case no default or explicit context is defined. The cell that defines the range is followed as a context for other cells to expand.

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 repect 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

Sales				
Name:	{{ds.Name(R=B11:F16)}}		**Revenue:**	{{ds.Revenue}}
Area:	{{ds.Area}}			
City:	{{ds.City}}			

The above template layout will generate the following Excel report:



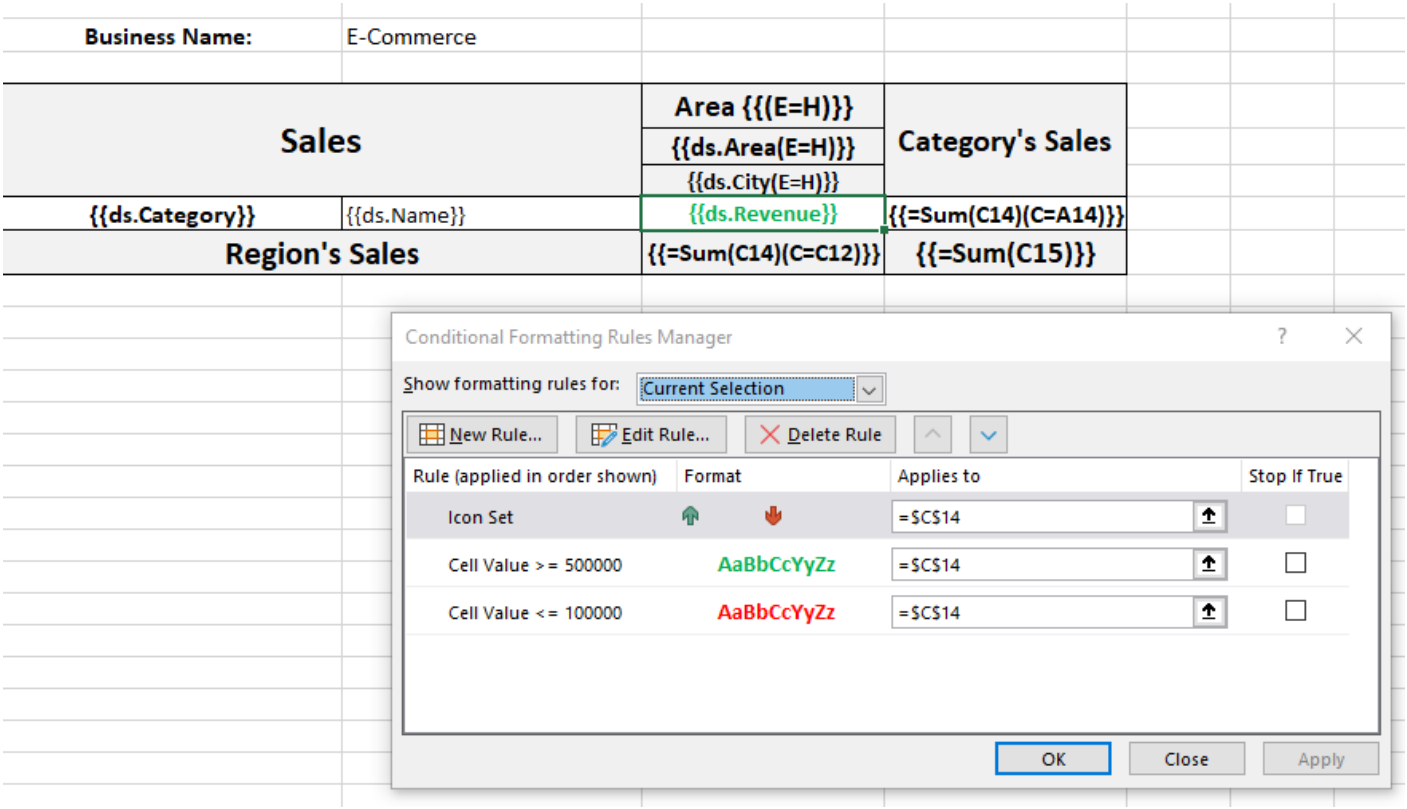
E-Commerce				
Sales				
Name:	Bose 785593-0050		**Revenue:**	$ 92,800.00
Area:	North America			
City:	Chicago			
Sales				
Name:	Canon EOS 1500D		**Revenue:**	$ 98,650.00
Area:	North America			
City:	Chicago			

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.



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			
	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

Limitation

If the formula reference in a conditional formatting rule refers to a template cell, it is not handled correctly by DsExcel 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 DsExcel 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 DsExcel template are explained below:

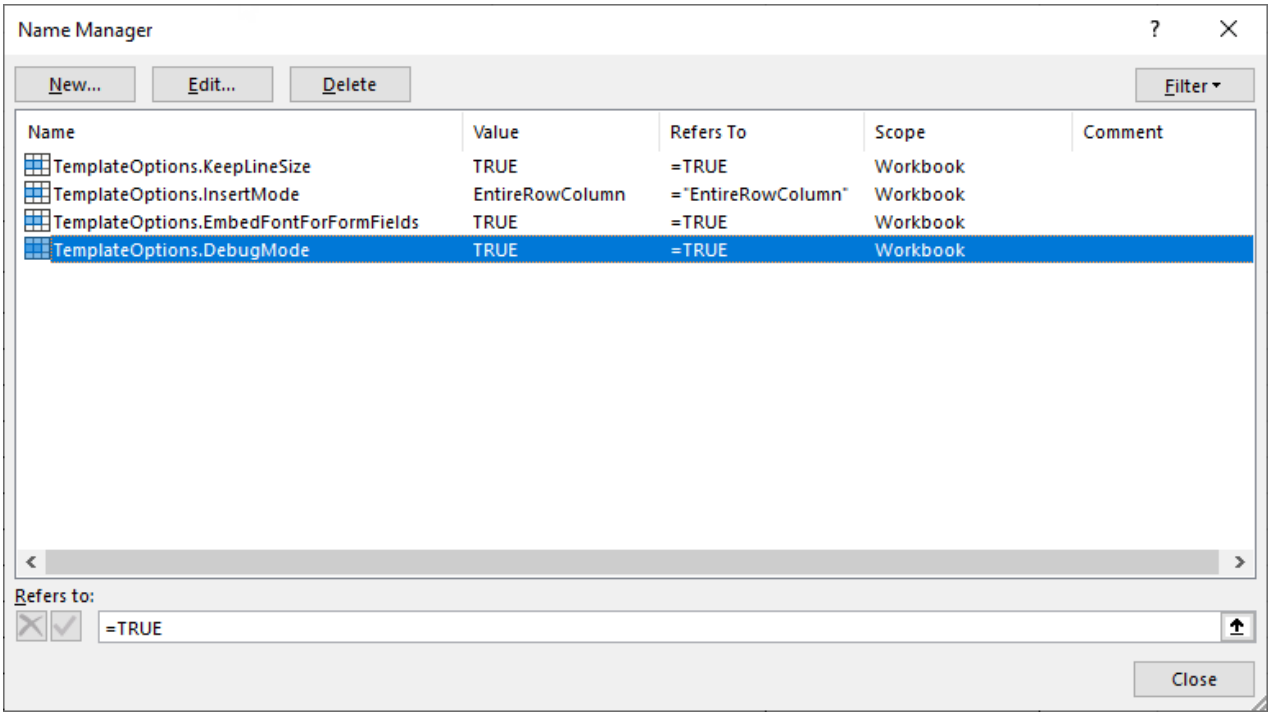
Global Settings	Description	Value
TemplateOptions.KeepLineSize	It specifies whether the row height and column width should be kept the same throughout the template	**Type:** Boolean **Value:** True False (Default Value)
TemplateOptions.InsertMode	It specifies whether to insert extra cells or entire rows and columns when extra space is needed while expanding the template	**Type:** String **Value:** Cells (Default Value) EntireRowColumn
TemplateOptions.EmbedFontForFormFields	It specifies whether the fonts used by form fields should be embedded in exported PDF file. **Note:** This setting is only applicable for PDF form fields	**Type:** Boolean **Value:** True (Default Value) False
TemplateOptions.DebugMode	It specifies whether to retain the original template data after the template has been expanded by keeping the template and the report in the same workbook.	**Type:** Boolean **Value:** True False (Default Value)
TemplateOptions.PaginationMode	It limits the number of instances of a template cell generated on a page. When the number of instances exceeds the CountPerPage property value, a new page is automatically created with the same layout as the template.	**Type:** Boolean **Value:** True False (Default Value)

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 DsExcel 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 DsExcel 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");

//Init DebugMode setting
workbook.Names.Add("TemplateOptions.DebugMode", "true");

//Init template global settings
workbook.Names.Add("TemplateOptions.PaginationMode", "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**.

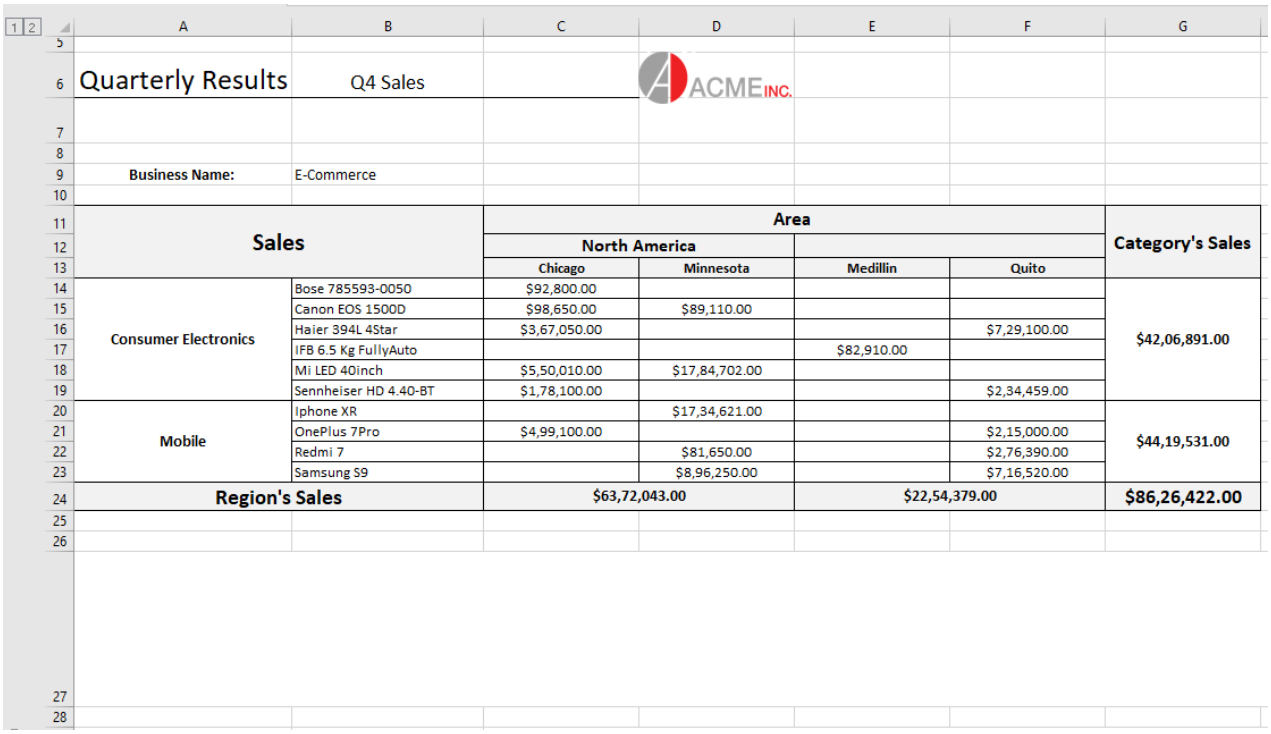
KeepLineSize

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		ACME INC.			
7							
8							
9	**Business Name:**	E-Commerce					
10							
11	**Sales**		**Area**				**Category's Sales**
12			**North America**				
13			**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.



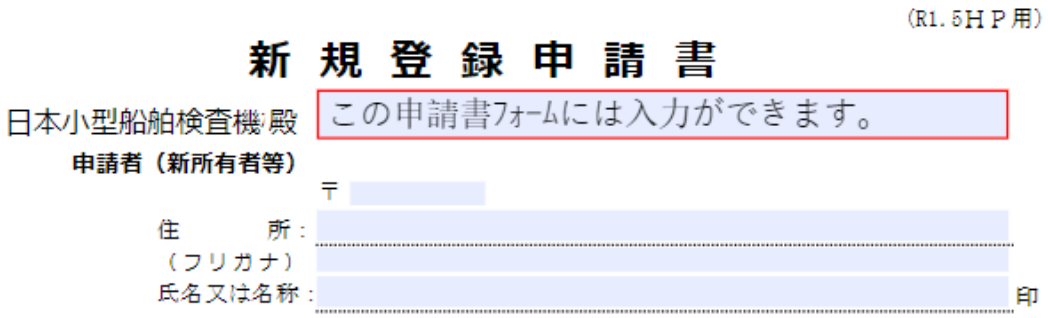
Quarterly Results	Q4 Sales		ACME INC.			
Business Name:	E-Commerce					
Sales		**Area**				**Category's Sales**
		North America				
		Chicago	**Minnesota**	**Medillin**	**Quito**	
Consumer Electronics	Bose 785593-0050	$92,800.00				**$42,06,891.00**
	Canon EOS 1500D	$98,650.00	$89,110.00			
	Haier 394L 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			
	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**

EmbedFontForFormFields

This setting allows you to embed font files used by form fields in the PDF document generated by DsExcel.

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 DsExcel when 'TemplateOptions.EmbedFontForFormFields' is set to True. You can also download the **Excel template layout** used in below example.



DebugMode

By setting **TemplateOptions.DebugMode** to true, you can compare the generated report to its corresponding template within the same workbook.

Below image shows the generated Excel report when TemplateOptions.DebugMode option is set to **true**.

Template Sheet

	A	B	C	D
5				
6	Quarterly Results	Q4 Sales	TREADSTONE	
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				

Sales_Template | Sales

Report Sheet

	A	B	C	D	E	F	G
5							
6	Quarterly Results	Q4 Sales	TREADSTONE				
7							
8							
9	**Business Name:**	E-Commerce					
10							
11	**Sales**					**Area**	
12						**North America**	
13			**Chicago**	**Fremont**	**Minnesota**	**New York**	**Buenos Aires**
14	**Consumer Electronics**	Bose 785593-0050	$92,800.00			$92,800.00	
15		Canon EOS 1500D	$98,650.00		$89,110.00		
16		Haier 394L 4Star	$367,050.00				
17		IFB 6.5 Kg FullyAuto		$904,930.00			$673,800.00
18		Mi LED 40inch	$550,010.00		$1,784,702.00		
19		Sennheiser HD 4.40-BT	$178,100.00				
20	**Mobile**	Iphone XR			$1,734,621.00		
21		OnePlus 7Pro	$499,100.00				
22		Redmi 7			$81,650.00		
23		Samsung S9			$896,250.00		$896,250.00
24	**Region's Sales**					**$7,369,773.00**	
25							
26							

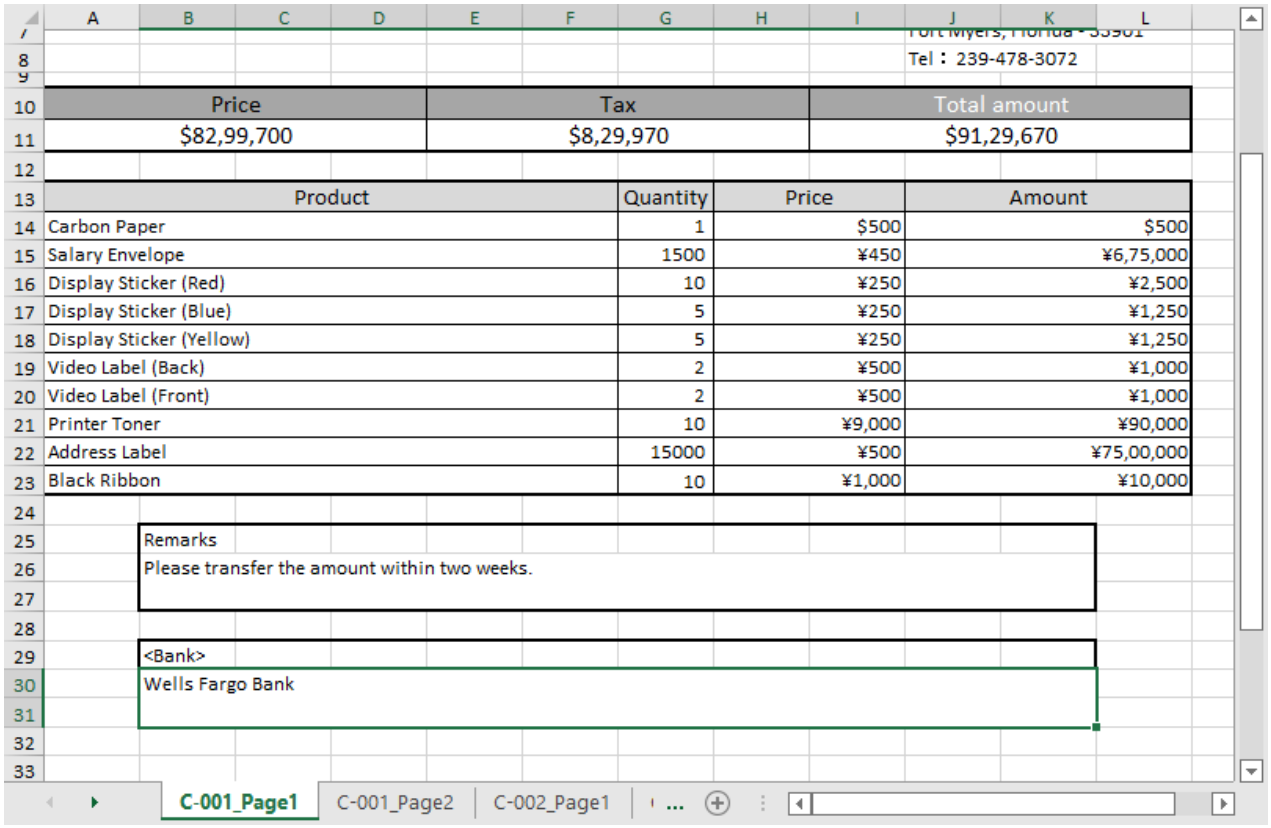
Sales_Template | Sales

PaginationMode

This setting ensures that **CountPerPage** template property is considered by the template engine. When **TemplateOptions.PaginationMode** is true, the worksheet paginates on basis of value set in CountPerPage template property. In such case, value of **TemplateOptions.KeepLineSize** is always considered as **true**. Also, value of **TemplateOptions.InsertMode** is considered as **EntireRowColumn** in the pagination mode.

Below images show Page1 and Page2 of the generated Excel report when TemplateOptions.PaginatedMode is set to true along with CountPerPage template property.

Page 1



Tel：239-478-3072

Price	Tax	Total amount
$82,99,700	$8,29,970	$91,29,670

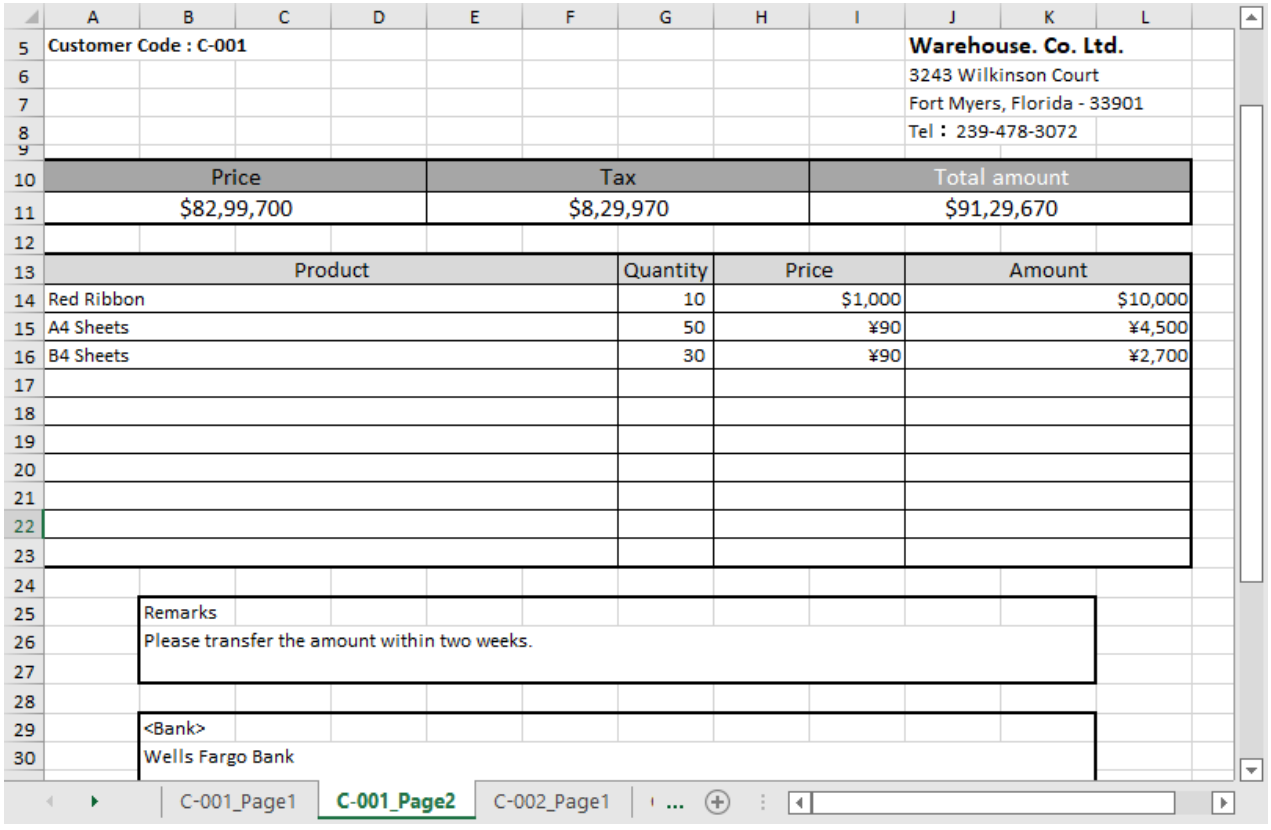
Product	Quantity	Price	Amount
Carbon Paper	1	$500	$500
Salary Envelope	1500	¥450	¥6,75,000
Display Sticker (Red)	10	¥250	¥2,500
Display Sticker (Blue)	5	¥250	¥1,250
Display Sticker (Yellow)	5	¥250	¥1,250
Video Label (Back)	2	¥500	¥1,000
Video Label (Front)	2	¥500	¥1,000
Printer Toner	10	¥9,000	¥90,000
Address Label	15000	¥500	¥75,00,000
Black Ribbon	10	¥1,000	¥10,000

Remarks
Please transfer the amount within two weeks.

<Bank>
Wells Fargo Bank

C-001_Page1 | C-001_Page2 | C-002_Page1

Page 2



Customer Code : C-001

Warehouse. Co. Ltd.
3243 Wilkinson Court
Fort Myers, Florida - 33901
Tel：239-478-3072

Price	Tax	Total amount
$82,99,700	$8,29,970	$91,29,670

Product	Quantity	Price	Amount
Red Ribbon	10	$1,000	$10,000
A4 Sheets	50	¥90	¥4,500
B4 Sheets	30	¥90	¥2,700

Remarks
Please transfer the amount within two weeks.

<Bank>
Wells Fargo Bank

C-001_Page1 | C-001_Page2 | C-002_Page1

Fixed Layout

When DsExcel 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, DsExcel 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 either Insert, Overwrite or OverwriteWithFormat

   - **Insert**: (Default Value) Inserts a blank line before setting the data and style in the cells.
   - **Overwrite(O)**: Directly set data in the cells without inserting blank rows however it retains style of the template layout.
   - **OverwriteWithFormat(OF)**: The data is set in the new cell that copies the style and merges from the template cell.

   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.

Business Name:	E-Commerce		
Area	**City**	**Name**	**Sales**
{{ds.Area(G=list, FM=overwrite)}}	{{ds.City}}	{{ds.Name}}	{{ds.Revenue}}
Region's Sales			**$0.00**

   The data in data source contains 8 rows. After DsExcel 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. To see the output of **fillMode** set to **OverwriteWithFormat**, see the Sample Browser.

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	**Business Name:**	E-Commerce		
10				
11	**Area**	**City**	**Name**	**Sales**
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	**Region's Sales**			**$0.00**

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 DsExcel 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 DsExcel 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, DsExcel 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.

Business Name:	E-Commerce		
Sales		**Area {{(E=H)}}** **{{ds.Area(E=H)}}** **{{ds.City(E=H)}}**	**Category's Sales**
{{ds.Category}}	{{ds.Name}}	{{ds.Revenue(defaultValue=0)}}	**{{=Sum(C14)(C=A14)}}**
Region's Sales		**{{=Sum(C14)(C=C12)}}**	**{{=Sum(C15)}}**

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

DsExcel 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 DsExcel 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"	{{ds.Work(form={"typ	{{ds.Cell(form={"type	{{ds.Home(form={"ty	{{ds.Email(form={"type": "te	{{ds.Birthday(form={"	{{ds.Address(form={"type": "t

After DsExcel 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 DsExcel 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:

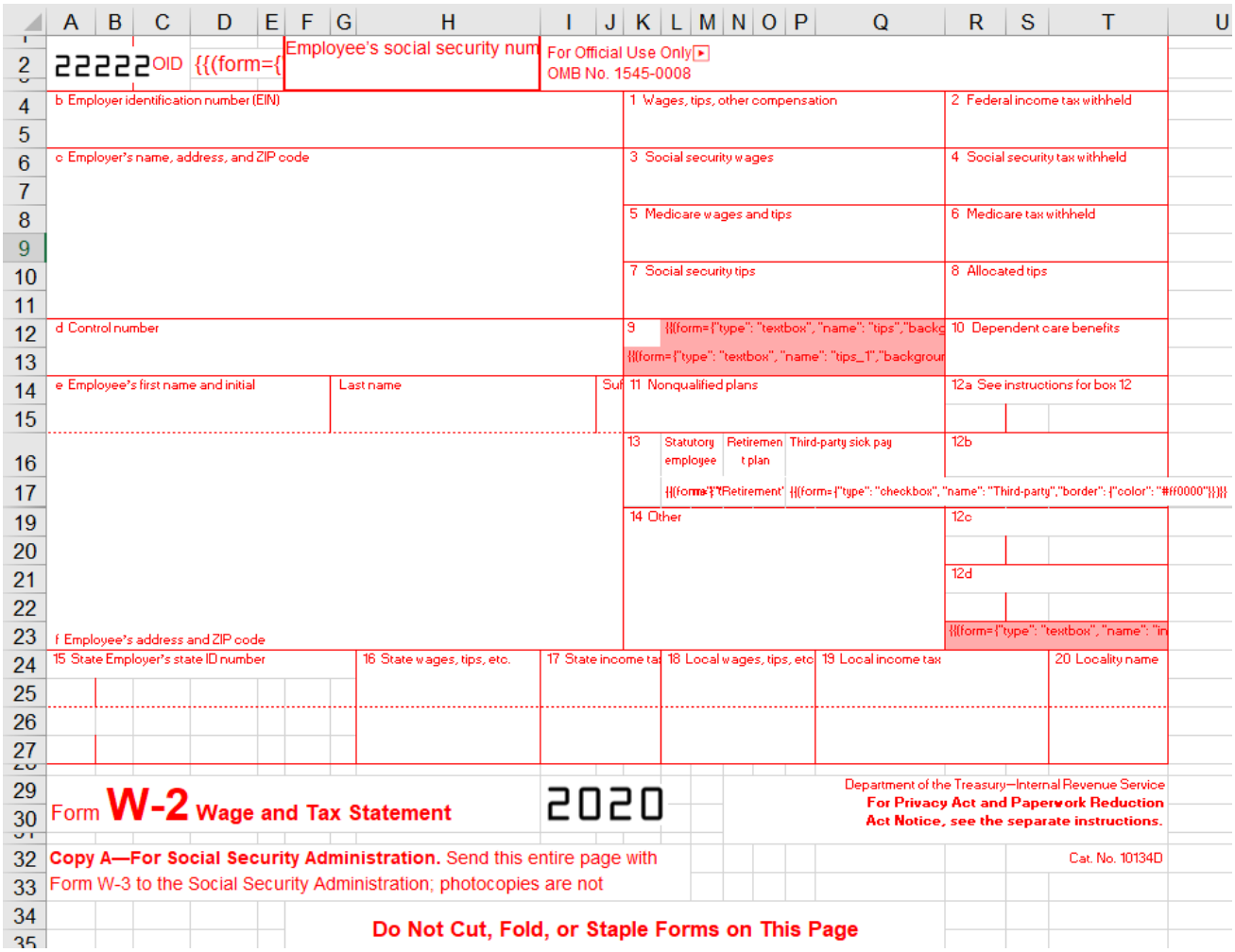
{{(form={"type": "textbox", "name": "tips","backgroundcolor": "#ffabab"})}}

Checkbox fields:

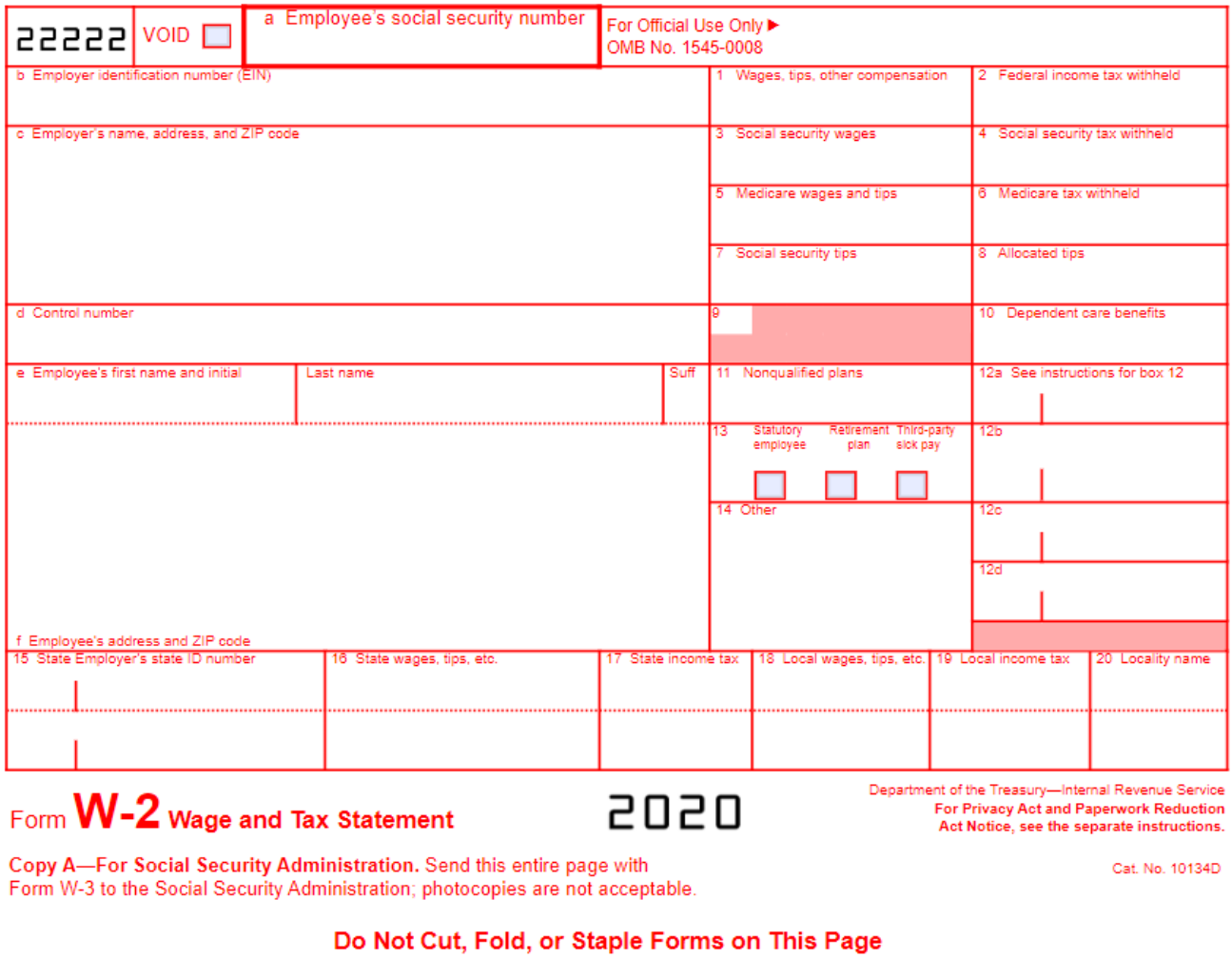
{{(form={"type": "checkbox", "name": "Retirement","border": {"color": "#ff0000"})}}}

{{(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.



After DsExcel processes the template and exports it to a PDF document, the PDF form will look like below:



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
type	Enum string Standard Form Fields • checkbox • textbox • listbox • combobox • radiobutton • pushbutton • signature Custom Form	{"type": "listbox"}	Indicates the type of form field. (Mandatory field)

		Fields • text • date • time • tel • email • url • password • month • week • number • search		
alternateName		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.
backgroundcolor		String	{"backgroundcolor": "#ffff00"} {"backgroundcolor": "rgb(255, 178, 0)"} {"backgroundcolor": "rgba(188, 100, 0, 255)"}	Indicates background color of the form field.
border	**width**	Yes	{"border":{"width": 120}}	Indicates width, color and style settings of the border for the form field.
	color	String	{"border":{"color": "#ffff00"} {"border": {"color": "rgb(255, 178, 0)"}} {"border": {"color": "rgba(188, 100, 0, 255)"}}	
	style	Enum string: ◦ none ◦ solid ◦ dashed ◦ beveled ◦ inset ◦ underline ◦ unknown	{"border": {"style": "dashed"}}	
font	**size**	Yes	{"font": {"size": 18}}	Indicates various font settings which can be used in the form field.
	color	String	{"font": {"color": "#ffff00"}} {"font": {"color": "rgb(255, 178, 0)"}} {"font": {"color": "rgba(188, 100, 0, 255)"}}	
	name	String	{"font": {"name": "sans-serif"}}	
	bold	Boolean	{"font": {"bold>": true}}	
	italic	Boolean	{"font": {"italic": true}}	
locked		Boolean	{"locked": true}	Indicates whether

			the user can change the properties of field or not.
name	String	{"name": "The field name"}	Indicates the unique name of field.
readOnly	Boolean	{"readOnly": true}	Indicates whether the user can change the value of field or not
required	Boolean	{"required": true}	Indicates whether the field must have a value.
printed	Boolean	{"printed":false}	Indicates whether to print the field when page is printed.
hidden	Boolean	{"hidden":true}	Indicates whether to display the field or not.
mouseUp	JsonObject	{"mouseUp":{"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 mouse button is released in the active area of the field.
mouseDown	JsonObject	{"mouseDown":{"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 mouse button is pressed in the active area of the field.
mouseEnter	JsonObject	{"mouseEnter":{"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 mouse button enters the field's active area.
mouseExit	JsonObject	{"mouseExit":{"script":"fBox1 = this.getField(\"checkbox\")	Indicates the actions to be

		;\r\nfBox1.display = display.hidden", "submit":"http://localhost:80//myscript#FDF" ,"reset":{"fieldNames": ["checkbox","textbox"]}}}	performed in sequence when the mouse button exitsthe field's active area.
onFocus	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.
onBlur	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.
format	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.
validate	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.
calculate	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.
keystroke	String	{"keystroke":"if (!event.willCommit)	Indicates a

			{\r\n"+"var f =this.getField (\"myPictures\");\r\n"+"var i =this.getIcon(event.change) ;\r\n"+"f.buttonSetIcon(i);\r\n"+"};"}	JavaScript action to be performedwhen 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.
autofocus		Boolean	{"type": "password" , "autofocus": true}	Indicates whether a date field should automatically get focus when the page loads.
disabled		Boolean	{"type": "password" , "disabled": true}	Indicates whether a field is disabled or not.
autocomplete	Enum String • on • off • name • honorific-prefix • given-name • additional-name • family-name • honorific-suffix • nickname • email • username • new-password • current-password • one-time-code • organization-title • organization	• cc-number • cc-exp • cc-exp-month • cc-exp-year • cc-csc • cc-type • transaction-currency • transaction-amount • bday-day • language • bday • bday-day • bday-month • bday-year • sex • tel • tel-country-code • tel-national	{"type": "date" , "autocomplete": "bday"}	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. The behavior of this property depends on the browser implementation.

	○ street-address ○ address-line1 ○ address-line2 ○ address-line3 ○ address-level4 ○ country ○ country-name ○ postal-code ○ cc-name ○ cc-given-name ○ cc-additional-name ○ cc-family-name	○ tel-area-code ○ tel-local ○ tel-local-prefix ○ tel-local-suffix ○ tel-extension ○ impp ○ url ○ photo		

2. **JsonObject Settings**

Name		Value Type	Example	Description
script		String	{"script":"fBox1 = this.getField(\"checkbox\");\r\nfBox1.display = display.hidden"}	Indicates an action which causes a script to be compiled and executed by the JavaScript interpreter.
submit		String	{"submit":"http://localhost:80//myscript#FDF"}	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.
reset	**fieldNames**	Yes	{"fieldNames": ["checkbox", "textbox"]}	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.
	exclude	Boolean	{"exclude":true}	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
checkStyle	Enum string: • check • circle • cross • diamond • square • Star	{"checkStyle": "circle"}	Indicates the style of check mark.
value	Boolean	{"value": true}	Indicates the value of Checkbox. (If the value is missing, DsExcel automatically tries to convert the cell's value to Boolean, and then, set it to the property after processing the template.)
defaultValue	Boolean	{"defaultValue": false}	Indicates the default value of Checkbox.

4. **Textbox Form Field Settings**

Name	Value Type	Example	Description
value	String	{"value": "Hunter"}	Indicates the value of Textbox.
defaultValue	String	{"defaultValue": "Input your name!"}	Indicates the default value of Textbox.
combo	Boolean	{"combo":true}	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
password	Boolean	{"password":true}	Indicates whether the field is intended for entering a secure password that should not be visible on the screen.
spellcheck	Boolean	{"spellcheck":false}	Indicates whether the text entered in the field is spell-checked.
scrollable	Boolean	{"scrollable":false}	Indicates whether the field is scrollable to accommodate more text than it fits within its annotation rectangle.
maxlength	Integer	{"maxlength":10}	Indicates the maximum length of the field's text, in characters.
multiline	Boolean	{"multiline":true}	Indicates whether the field can contain multiple lines of text.
justification	**Enum string:** • left • center • right	{"justification": "center"}	Indicates the justification to be used while displaying the field's text.

Note: DsExcel 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
value	String Array	{"value": ["US", "UK"]}	Indicates the value of Listbox.
defaultValue	String Array	{"defaultValue": ["US", "UK"]}	Indicates the default value of Listbox.
commitOnSelChange	Boolean	{"commitOnSelChange": true}	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
selectedIndex	Integer	{"selectedIndex": 0}	Indicates the indexes of selected item.
sort	Boolean	{"sort": true}	Indicates whether the field's option items should be sorted alphabetically.
spellCheck	Boolean	{"spellCheck": true}	Indicates whether the text entered in the field is spell-checked.
selectedIndexes	Integer Array	{"selectedIndexes": [0, 2, 5]}	Indicates the indexes of selected items.
multiSelect	Boolean	{"multiSelect": true}	Indicates whether more than one of the field's option items may be selected simultaneously.
exportValue	String	{"exportValue": "TheResult"}	Indicates the export value of Listbox field.

6. **Combobox Form Field Settings**

Name	Value Type	Example	Description
value	String Array	{"value": ["US", "UK"]}	Indicates the value of Combobox.
defaultValue	String Array	{"defaultValue": ["US", "UK"]}	Indicates the default value of Combobox.
commitOnSelChange	Boolean	{"commitOnSelChange": true}	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
selectedIndex	Integer	{"selectedIndex": 0}	Indicates the indexes of selected item.
sort	Boolean	{"sort": true}	Indicates whether the field's option items should be sorted alphabetically.
spellCheck	Boolean	{"spellCheck": true}	Indicates whether the text entered in the field is spell-checked.
editable	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
checkStyle	Enum string: • check • circle • cross • diamond • square • Star	{"checkStyle": "circle"}	Indicates the style of check mark.
groupName	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, DsExcel automatically adds radio buttons expanded from the same template cell to the same group after processing the template)
radiosInUnison	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).
checkedChoice	String	{"checkedChoice": "Team5"}	Indicates the value of checked option.
defaultCheckedChoice	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
highlighting	**Enum string:** • none • invert • outline • push	{"highlighting": "outline"}	Indicates the annotation's highlighting mode.
caption	String	{"caption": "Push"}	Indicates the button's caption.
image	Base64 String	{"image": "The base64 image data."}	Indicates the button's image.
captionImageRelation	**Enum string:** • captionOnly	{"captionImageRelation": "captionBelowIcon"}	Indicates the positioning of button's caption relative to image.

		• imageOnly • captionBelowIcon • captionAboveIcon • captionAtRight • captionAtLeft • captionOverlaid		
downCaption		String	{"downCaption": "Push Down"}	Indicates the button's caption when user presses the button.
downImage		Base64 String	{"downImage": "The base64 image data."}	Indicates the button's image when user presses the button.
rolloverCaption		String	{"rolloverCaption": "Rollover"}	Indicates the button's caption when the user rolls the cursor into its active area without pressing the mouse button.
rolloverImage		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.
imageScale	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. 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. This value is used only if the image is scaled proportionally.
	y	Float	{"imageScale": {"proportional": true, "y": 0.8}}	
	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
lockType	**Enum string:** • all	{"lockType": "specifiedOnly"}	Indicates the type of locked fields.

	• specifiedOnly • allButSpecified		
fieldNames	String Array	{"fieldNames": ["signerName", "time"]}	Indicates the list of field names which should be included or excluded from processing depending on lockType property.
LockedFields	Boolean	{"LockedFields": true}	Indicates whether to lock the fields when SignatureFormField is signed or not.

Note: DsExcel Template generates only digital signature fields in PDF documents. If you want to add signatures on signature fields, you need to use DsPdf or PDFBox to process.

Apart from the above mentioned standard PDF form fields, DsExcel 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 DsExcel 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 Document Solutions 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
pattern	String	{"type": "tel" , "pattern":"[0-9]{3}-[0-9]{2}-[0-9]{3}"}	Pattern the value must match to be valid
placeholder	String	{"type": "password" , "placeholder": "4 to 8 characters"}	Sets or returns the value of the placeholder attribute of an tel field
maxlength	Integer	{"type": "password" , "minlength": 4, "maxlength": 8}	Maximum length (number of characters) of value
minlength	Integer	{"type": "password" , "minlength": 4, "maxlength": 8}	Minimum length (number of characters) of value

Email Custom Form Input Type Settings

Property	Value	Example	Description
multiple	Boolean	{"type": "email" , "multiple": true}	Sets or returns whether a user is allowed to enter more than one email address in the email field.
pattern	String	{"type": "email" , "pattern": "\\S+@\\S+\\.\\S+"}	Sets or returns the value of the pattern attribute of an email field.
placeholder	String	{"type": "email" , "placeholder": "example@xxx.com"}	Sets or returns the value of the placeholder attribute of an email field.

Text Custom Form Input Type Settings

Property	Value	Example	Description
maxlength	Integer	{"type": "text" , "minlength": 4, "maxlength": 8}	Maximum length (number of characters) of value
minlength	Integer	{"type": "text" , "minlength": 4, "maxlength": 8}	Minimum length (number of characters) of value

pattern	String	{"type": "text" , "pattern": "\\S+@\\S+\\.\\S+"}	Sets or returns the value of the pattern attribute of field
placeholder	String	{"type": "text" , "placeholder": "Input your name!"}	Sets or returns the value of the placeholder attribute of field
spellcheck	Boolean	{"type": "text" , "spellcheck": true}	Whether the element may be checked for spelling errors

Search Custom Form Input Type Settings

Property	Value	Example	Description
maxlength	Integer	{"type": "search" , "minlength": 4, "maxlength": 8}	Maximum length (number of characters) of value
minlength	Integer	{"type": "search" , "minlength": 4, "maxlength": 8}	Minimum length (number of characters) of value
placeholder	String	{"type": "search" , "placeholder": "Search..."}	Sets or returns the value of the placeholder attribute of field
spellchecker	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
validateonmessage	String	Localized validation message
validateoninput	Boolean	Indicates whether validation should be performed immediately during user input
maxlength	Number	Maximum number of characters to be accepted
minlength	Number	Minimum number of characters which can be considered valid
required	Boolean	Indicates whether the form filling is required or not
pattern	String	Regular expression that must be matched by the entered value to pass constraint validation
max	Number	Maximum value to accept for this input
min	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
autocomplete	All	Input type.

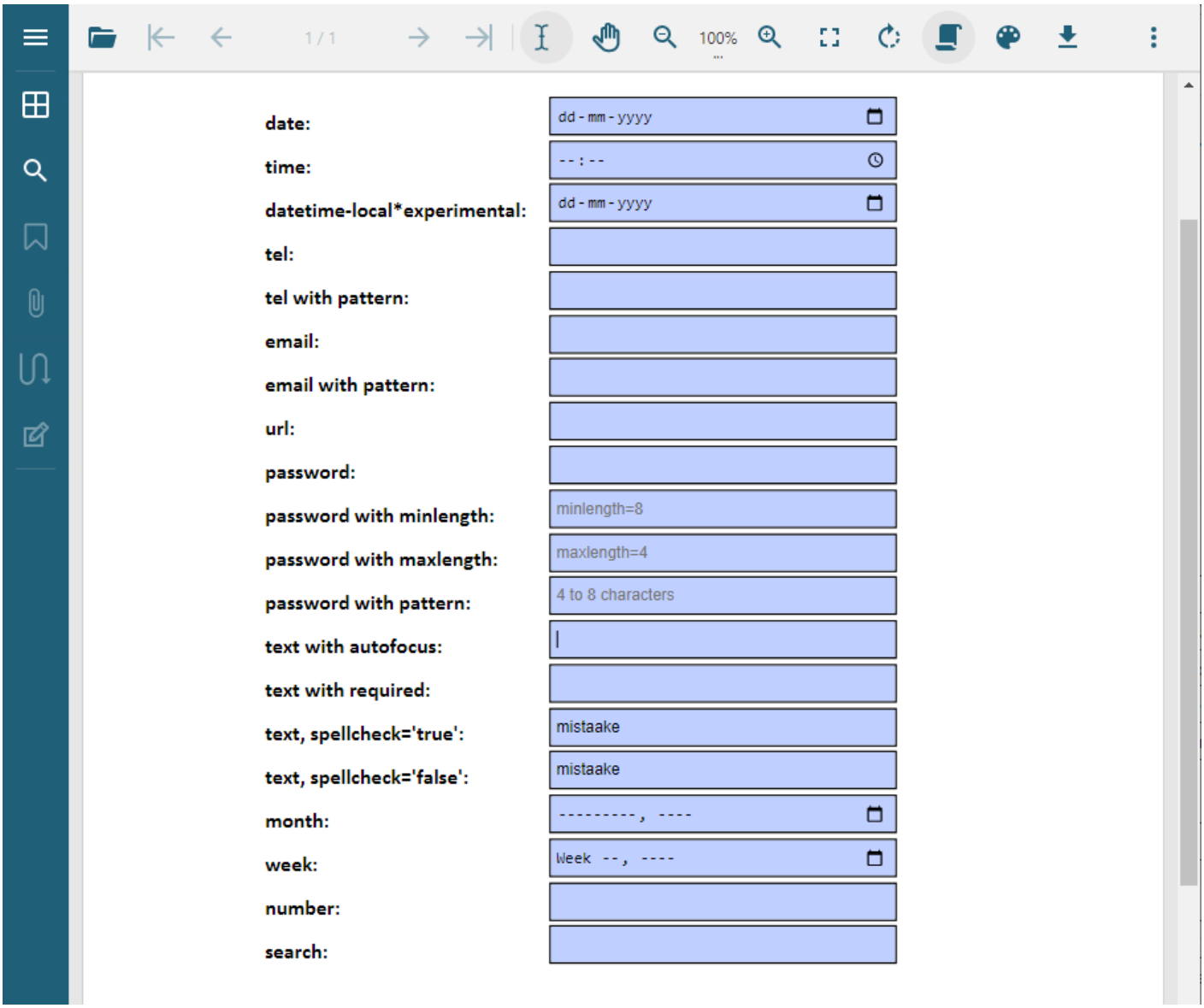
autofocus	All	Automatically focus the form control when the page is loaded.
defaultvalue	All	The default value.
disabled	All	Whether the form control is disabled.
displayname	All	Text label for the input control. Applicable only if the field appears in the Form Filler dialog box.
min	number and date	Minimum value to accept for the input.
max	number and date	Maximum value to accept for the input.
maxlength	password, search, tel, text and url	Maximum length (number of characters) of value.
minlength	password, search, tel, text and url	Minimum length (number of characters) of value
multiline	text	Set this property to true if you want to use the textarea as a user input element.
multiple	email	Boolean. Whether to allow multiple values or not.
pattern	password, text and tel	Pattern value must match to be valid.
placeholder	password, search, tel, text and url	Text that appears in the form control when it has no value set.
readonly	All	Boolean. The value is not editable.
required	All	Boolean. A value is required or must be check for the form to be submittable.
spellcheck	search and text	Whether the element may be checked for spelling errors.
type	All	Type of form control.
validateonmessage	All	Localized validation message.
validateoninput	All	Indicates whether validation should be performed immediately during user input.

The following Excel template shows various input types and settings supported with DsExcel 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 Document Solutions 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
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 entere
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'
password with minle	{{{form={"type":"password", "autocomplete":"new-password", "validateoninput": true, "placeholder": "minlength=8", "minlength": 8, "backgrour
password with maxle	{{{form={"type":"password", "autocomplete":"off", "validateoninput": true, "placeholder": "maxlength=4", "maxlength": 4, "backgroundcolor": "r
password with patte	{{{form={"type":"password", "autocomplete":"off", "validateoninput": true, "placeholder": "4 to 8 characters", "pattern": "^(?=.*\\d).{4,8}$", "vali
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='fals	{{{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 DsExcel processes the above template and exports it to a PDF document, the PDF form will look like below in Document Solutions PDF Viewer:



Note: The PDF form with custom input fields can only be filled in Document Solutions PDF Viewer. You can also customize the custom form input type settings by using Document Solutions 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.

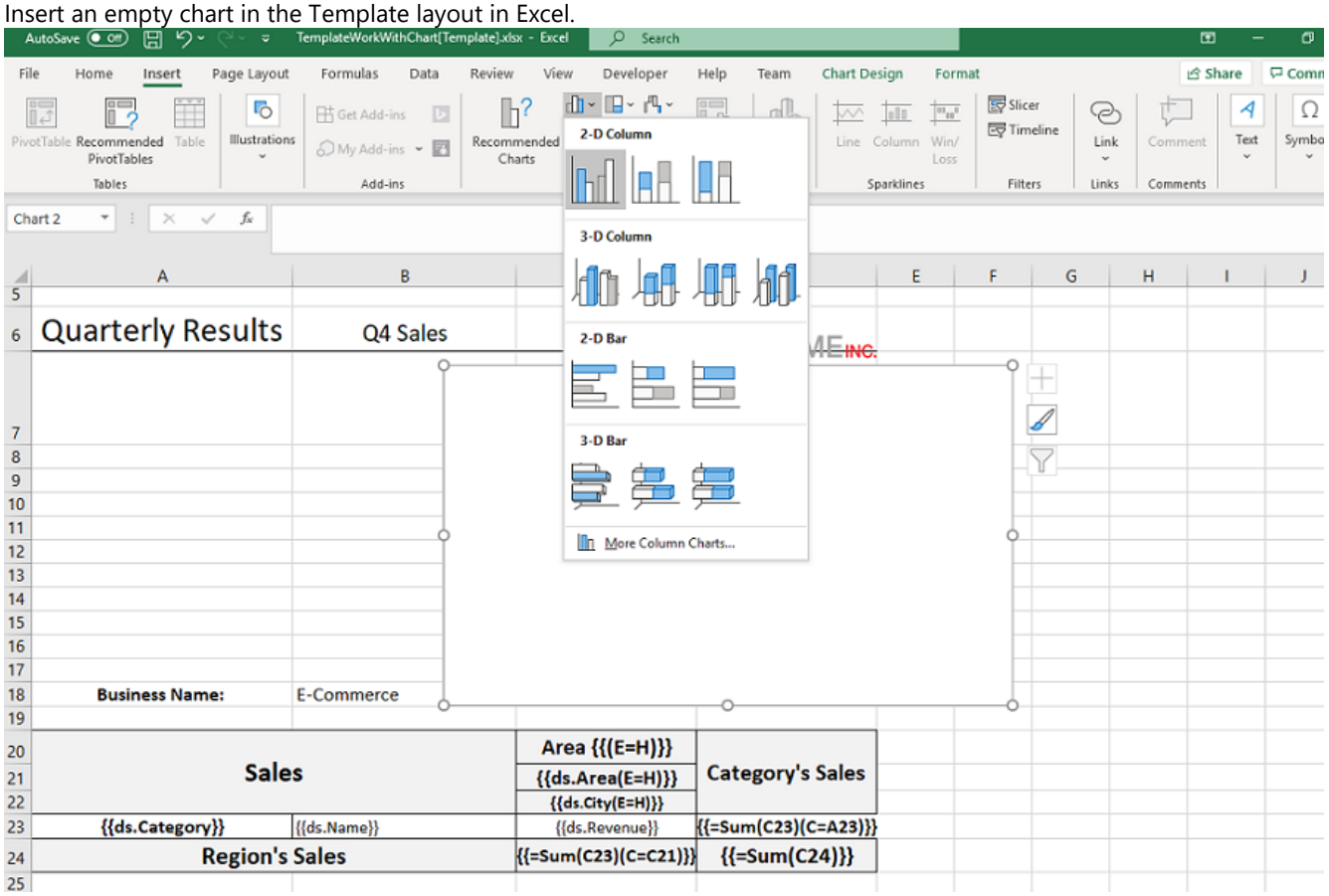
The Excel charts are bound with template cells by specifying the series name, series value and axis labels in the template layout. While processing the template layout, the chart is bound to the data, and the Excel report is generated with the chart displaying final data. A chart can be placed in a worksheet 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:

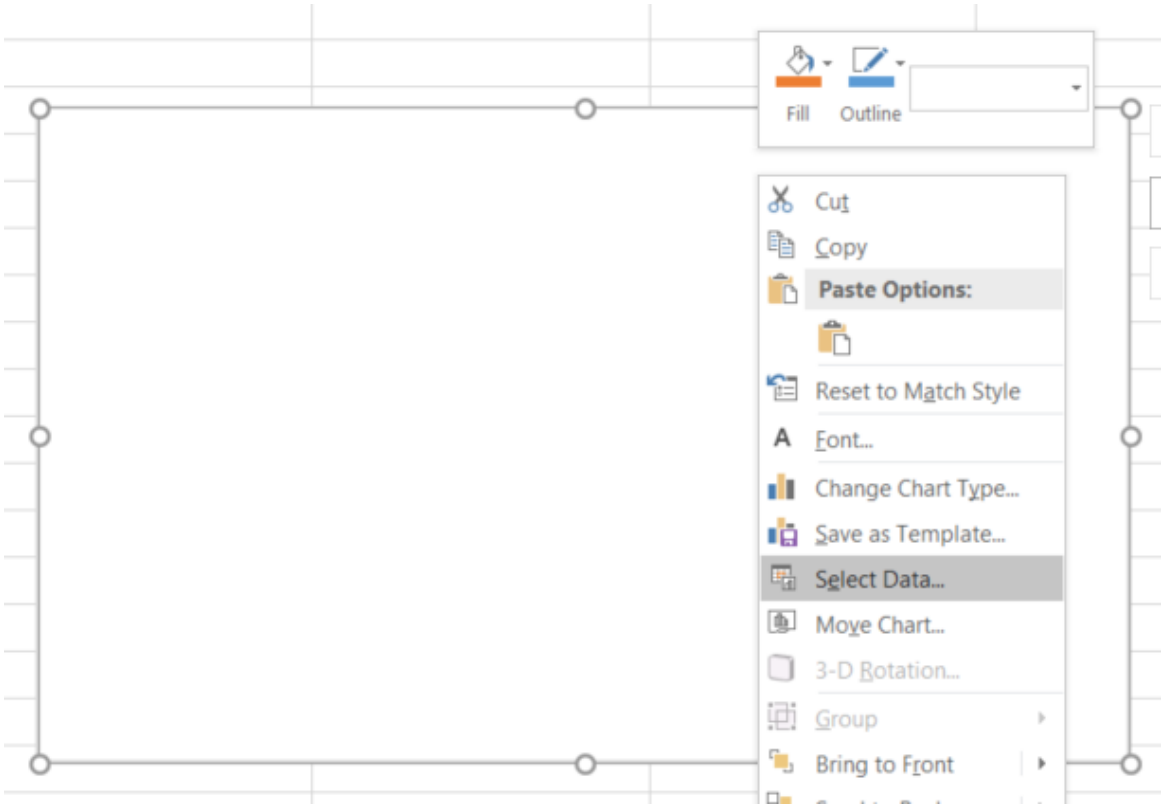
Here we are adding a chart to a 'Quarterly Sales Data' report which displays the sales of electronic goods in different areas of North and South America. The chart in the template configures the name of an electronic item as series name, revenue as series value and the city in which sales have been done as axis labels to display the sales data in a graphical manner.

You can also download the **Excel template layout** used in below example.

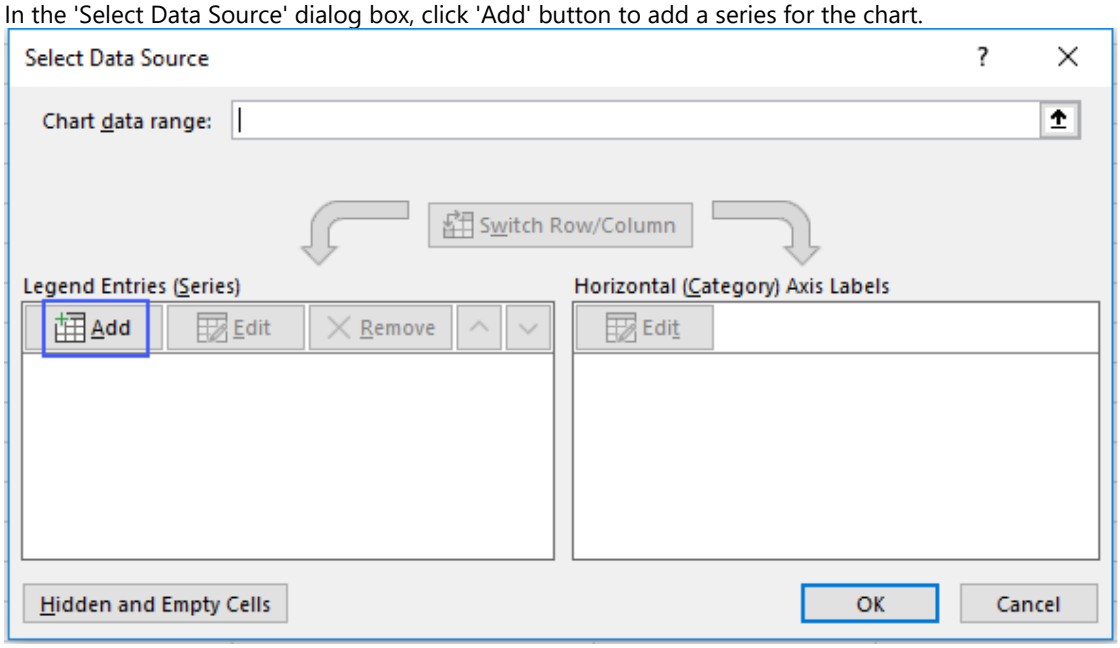
1. Insert an empty chart in the Template layout in Excel.



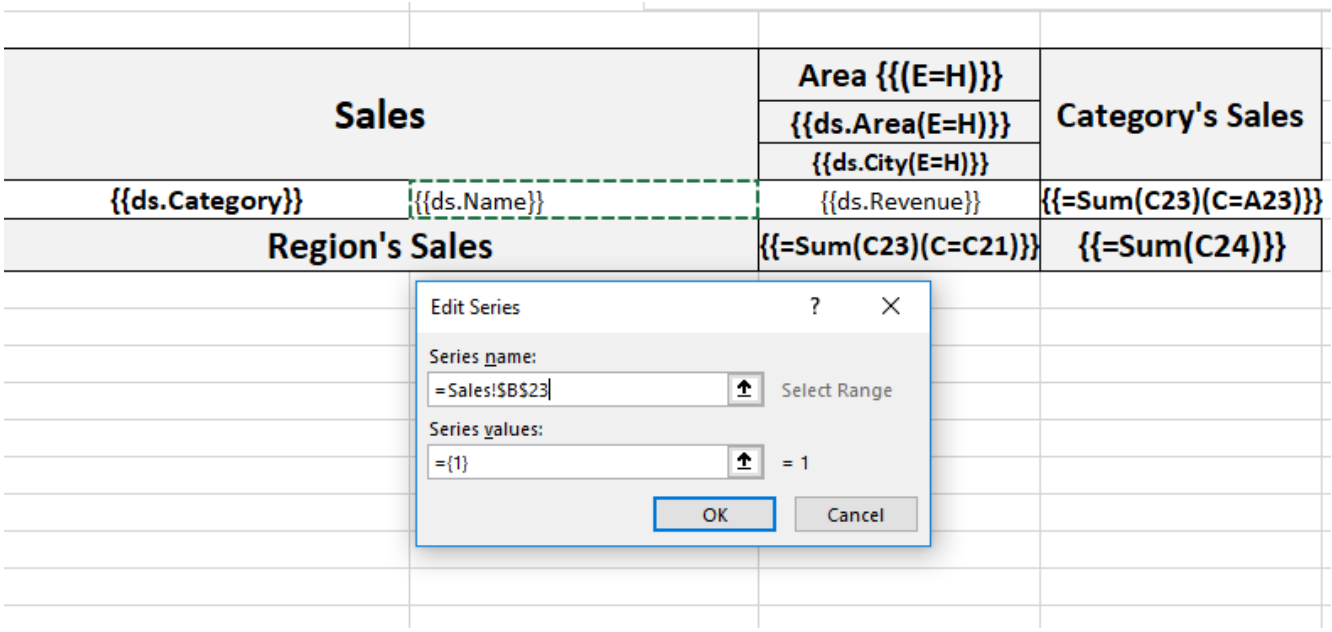
2. Right click on the chart and choose 'Select Data' from context menu



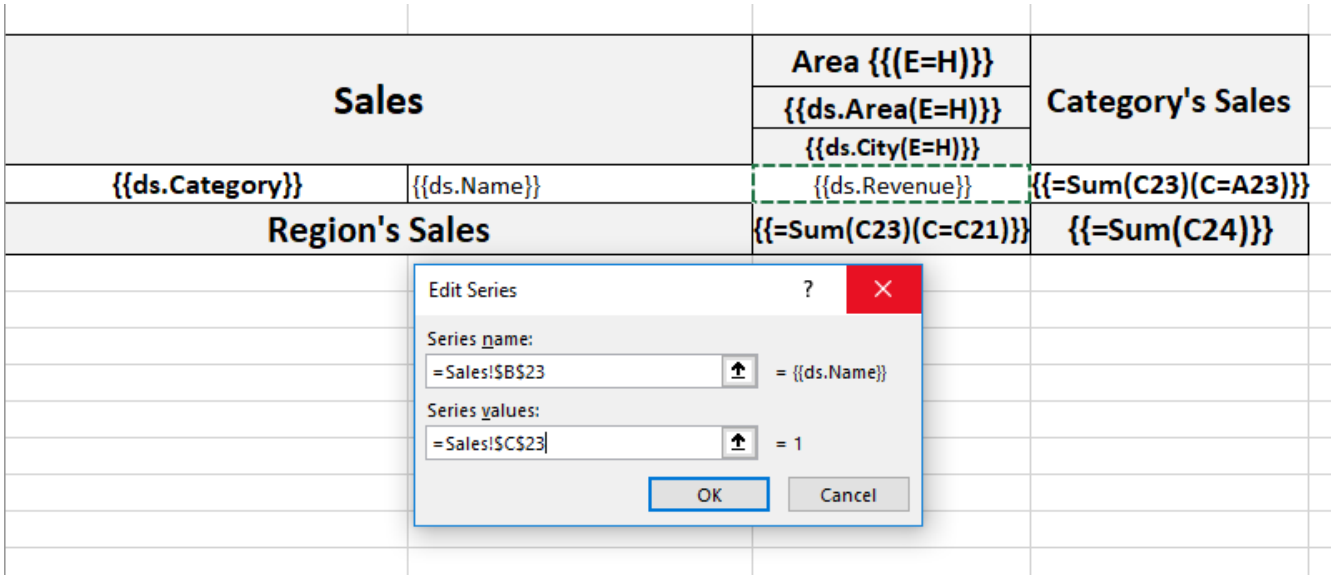
3. In the 'Select Data Source' dialog box, click 'Add' button to add a series for the chart.



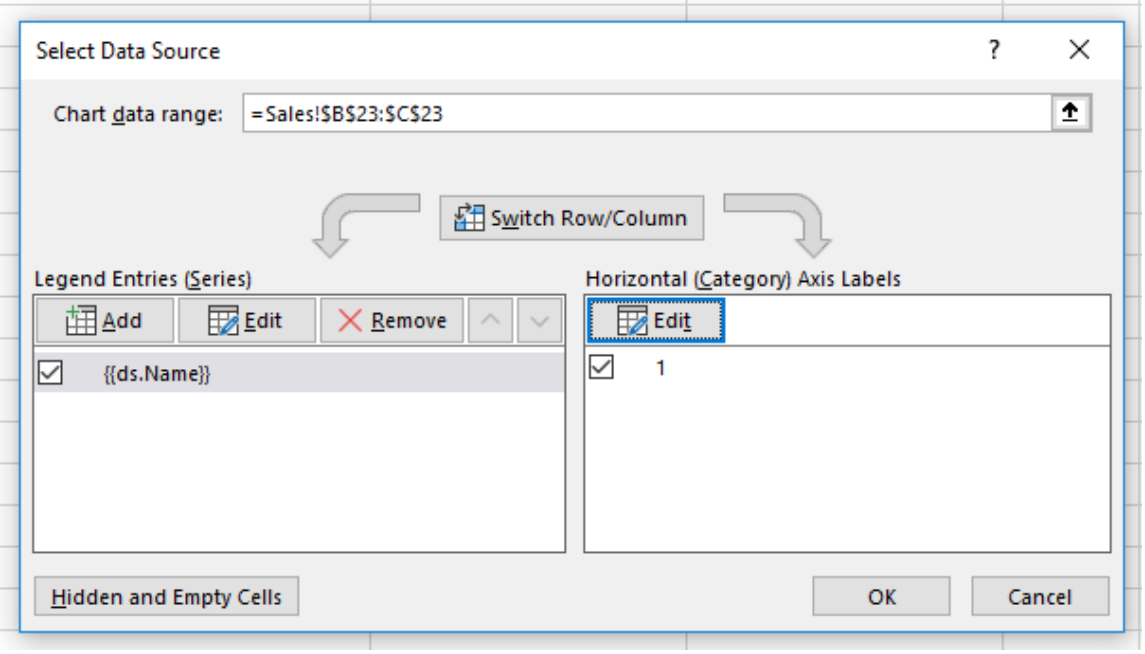
4. In the 'Edit Series' dialog box, click in 'Series Name' and then select 'ds.Salesman' field of the template layout as salesman field is being used as series for the chart.



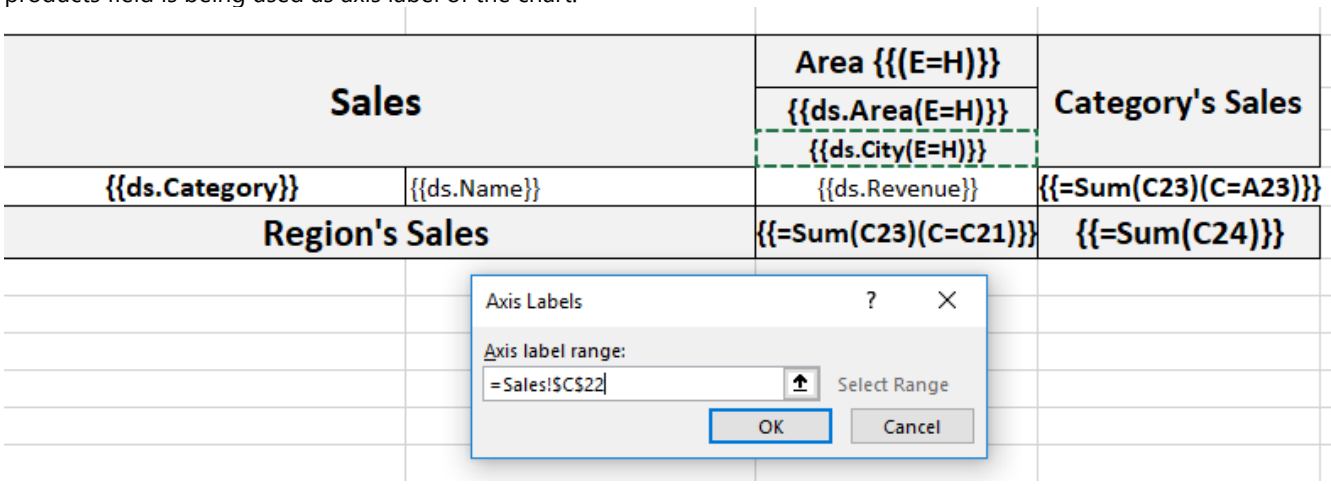
5. Next, click in 'Series Values' and then select 'ds.Sales' field of the template layout as sales field is being used as the value for the series of the chart.



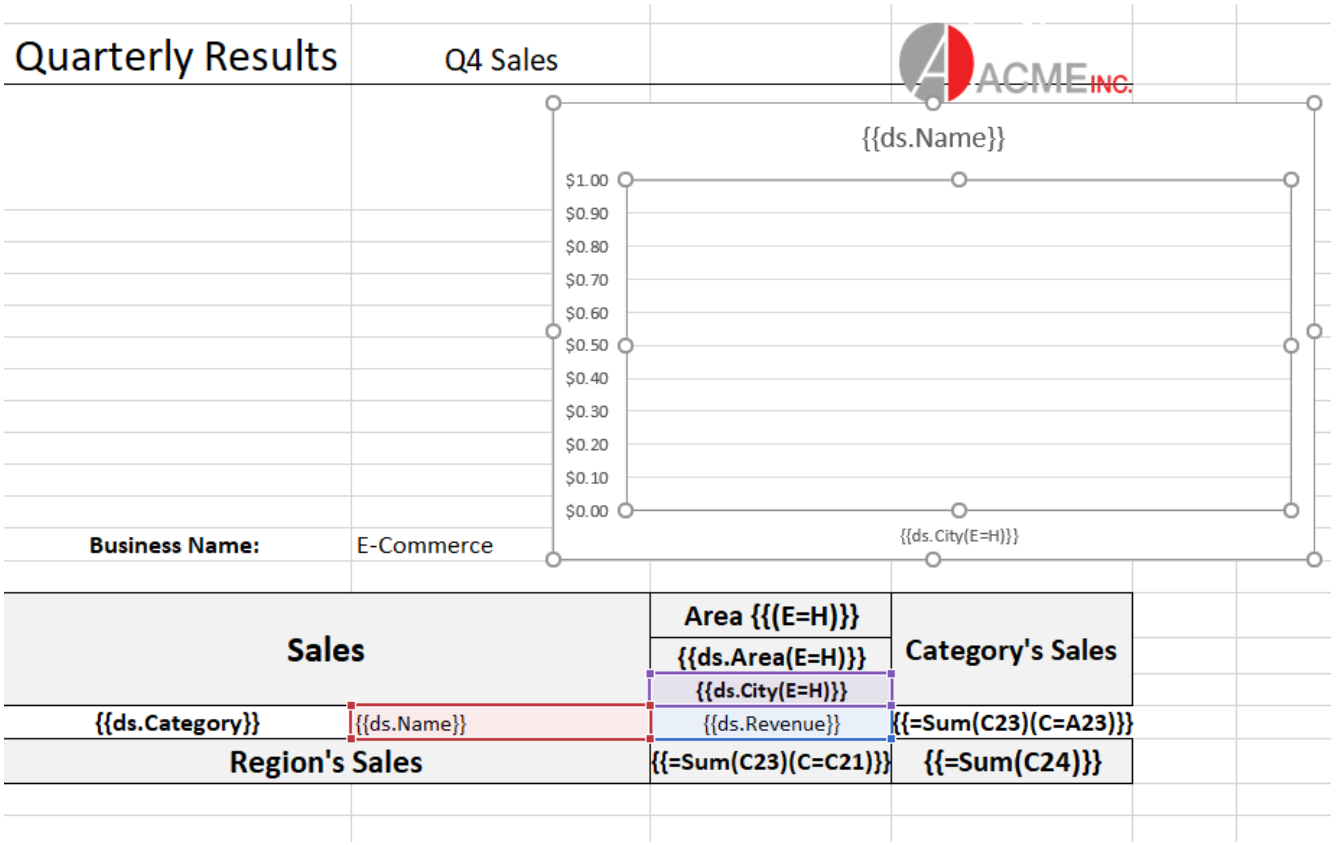
6. Click on the 'Edit' button highlighted in the below screenshot.



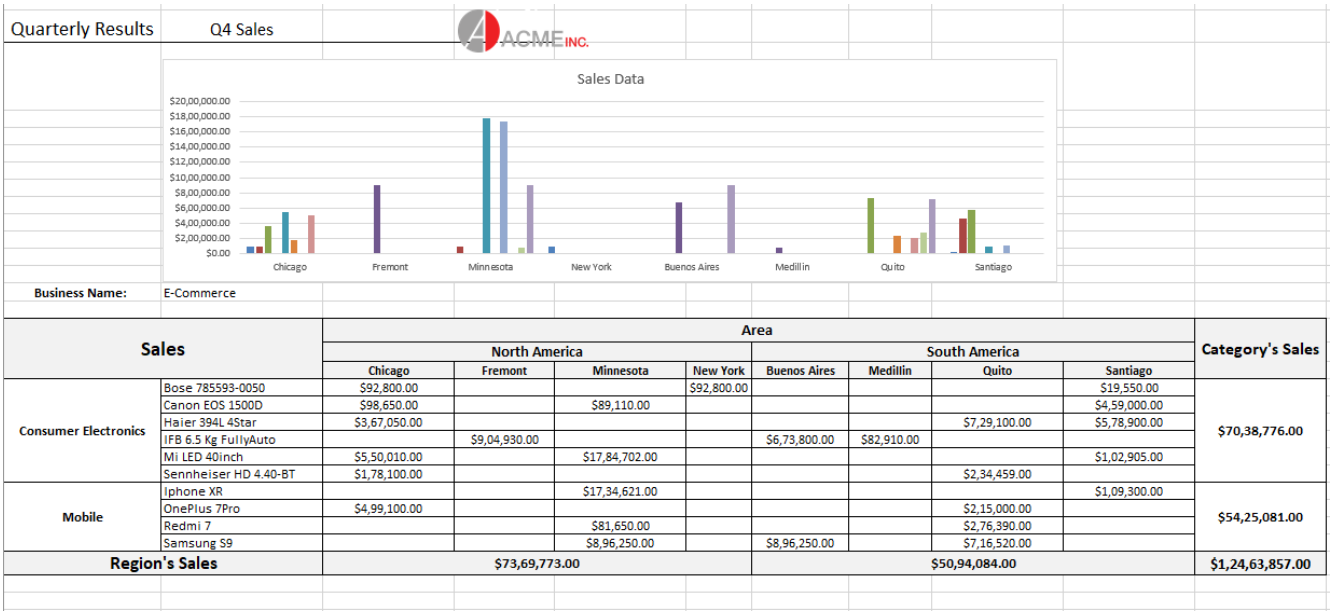
7. In the 'Axis Labels' dialog box, click in 'Axis Label Range' and then select 'ds.Product' field of the template layout as products field is being used as axis label of the chart.



8. Click Ok to submit the data configuration.



After processing the template layout, the Excel report will look like below.



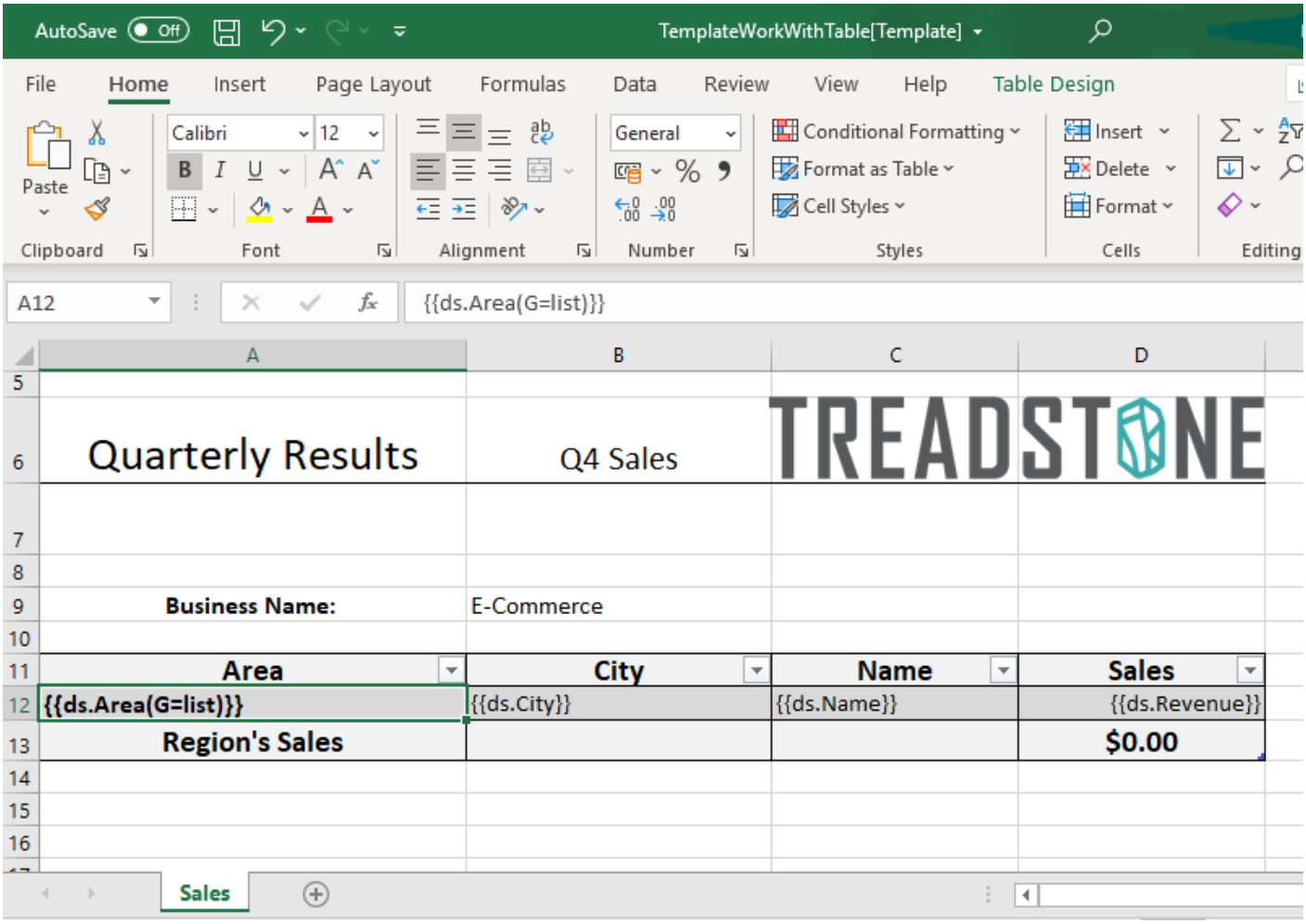
Sales		Area								Category's Sales
		North America				South America				
		Chicago	Fremont	Minnesota	New York	Buenos Aires	Medillin	Quito	Santiago	
Consumer Electronics	Bose 785593-0050	$92,800.00			$92,800.00				$19,550.00	$70,38,776.00
	Canon EOS 1500D	$98,650.00		$89,110.00					$4,59,000.00	
	Haier 394L 4Star	$3,67,050.00						$7,29,100.00	$5,78,900.00	
	IFB 6.5 Kg FullyAuto		$9,04,930.00			$6,73,800.00	$82,910.00			
	Mi LED 40inch	$5,50,010.00		$17,84,702.00					$1,02,905.00	
	Sennheiser HD 4.40-BT	$1,78,100.00						$2,34,459.00		
Mobile	Iphone XR			$17,34,621.00					$1,09,300.00	$54,25,081.00
	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		$8,96,250.00		$7,16,520.00		
Region's Sales		$73,69,773.00				$50,94,084.00				$1,24,63,857.00

Tables

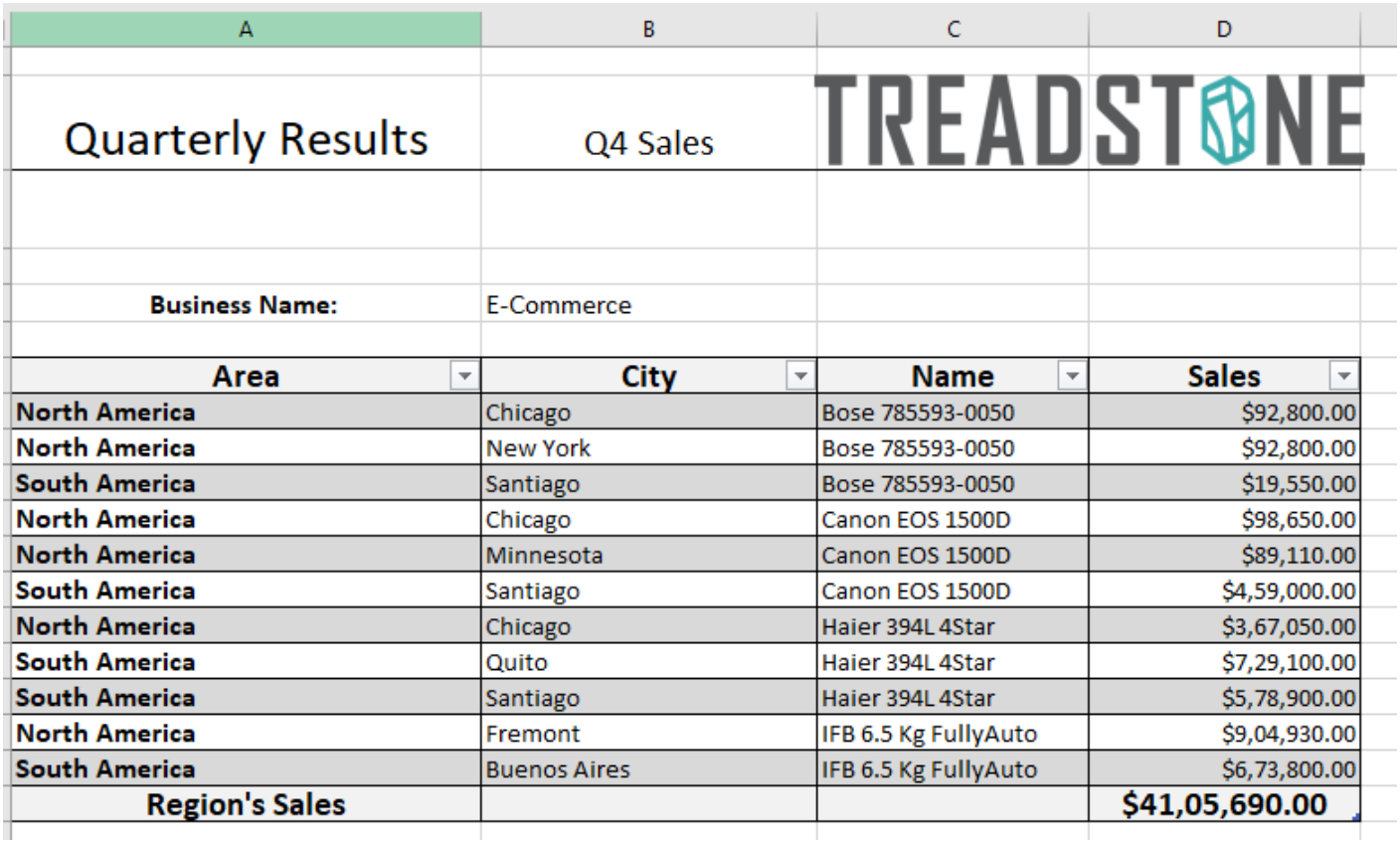
Tables are essential to depict large amounts of data in an organized way. DsExcel supports using Excel tables in template layouts where various operations can also be performed on it like filtering, sorting etc.

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 DsExcel processes the template layout, the Excel report will look like below:



	A	B	C	D
1	Quarterly Results	Q4 Sales	TREADSTONE	
2				
3				
4	**Business Name:**	E-Commerce		
5				
6	**Area**	**City**	**Name**	**Sales**
7	**North America**	Chicago	Bose 785593-0050	$92,800.00
8	**North America**	New York	Bose 785593-0050	$92,800.00
9	**South America**	Santiago	Bose 785593-0050	$19,550.00
10	**North America**	Chicago	Canon EOS 1500D	$98,650.00
11	**North America**	Minnesota	Canon EOS 1500D	$89,110.00
12	**South America**	Santiago	Canon EOS 1500D	$4,59,000.00
13	**North America**	Chicago	Haier 394L 4Star	$3,67,050.00
14	**South America**	Quito	Haier 394L 4Star	$7,29,100.00
15	**South America**	Santiago	Haier 394L 4Star	$5,78,900.00
16	**North America**	Fremont	IFB 6.5 Kg FullyAuto	$9,04,930.00
17	**South America**	Buenos Aires	IFB 6.5 Kg FullyAuto	$6,73,800.00
18	**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 DsExcel.
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 DsExcel.

> **Note:** Table formulas are also supported in template cells.

Limitations

- In DsExcel 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.

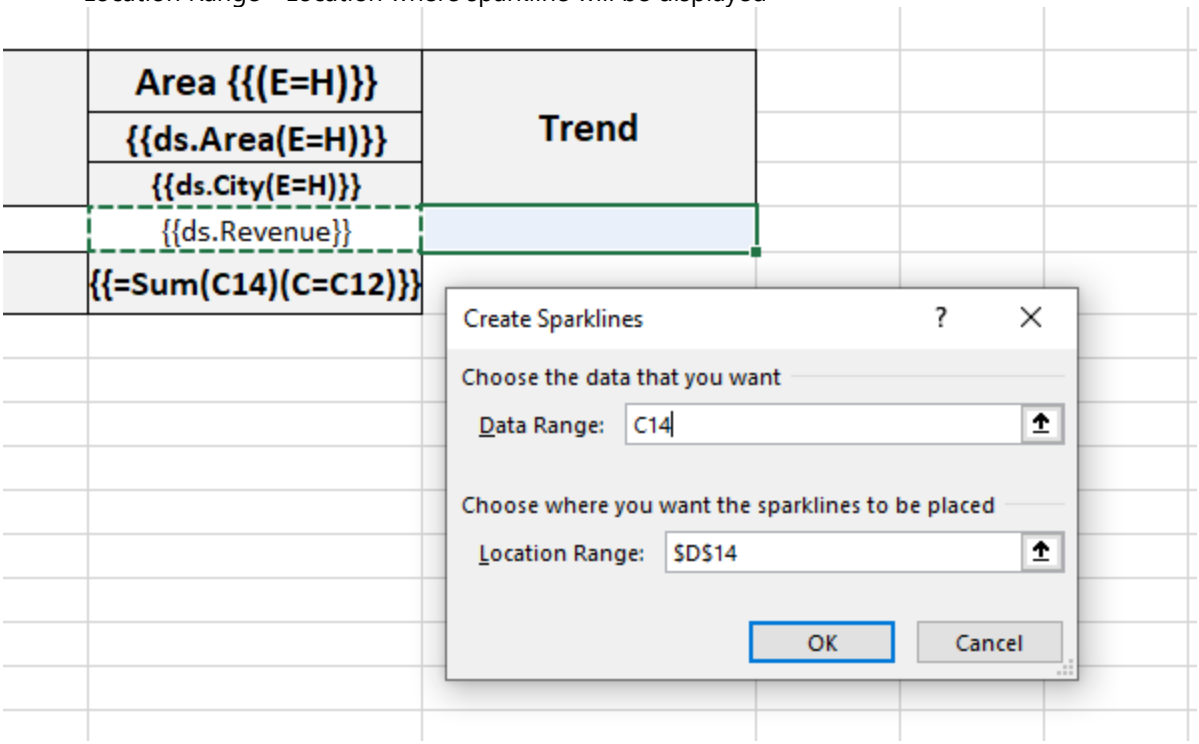
Sparklines

DsExcel 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 DsExcel 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	
	Canon EOS 1500D	$98,650.00		$89,110.00					$4,59,000.00	
	Haier 394L 4Star	$3,67,050.00						$7,29,100.00	$5,78,900.00	
	IFB 6.5 Kg FullyAuto		$9,04,930.00			$6,73,800.00	$82,910.00			
	Mi LED 40inch	$5,50,010.00		$17,84,702.00					$1,02,905.00	
	Sennheiser HD 4.40	$1,78,100.00						$2,34,459.00		
Mobile	Iphone XR			$17,34,621.00					$1,09,300.00	
	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		$8,96,250.00		$7,16,520.00		
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.

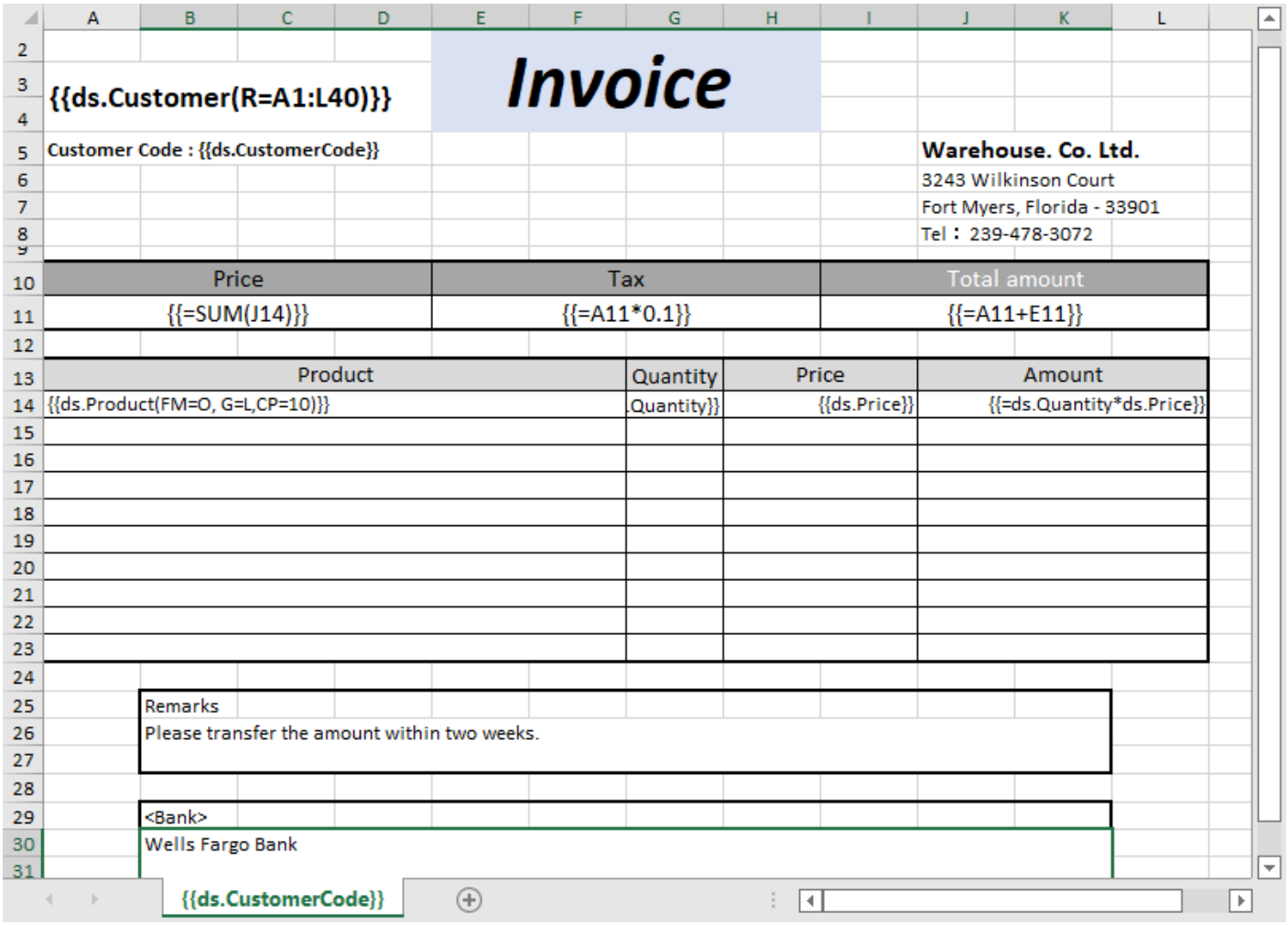
Paginated Templates

Paginated template lets you paginate worksheets in a report having the same layout such as invoice, progress report, medical test reports etc. Reports can be paginated in two ways; either by page size or by number of records per page.

The page-size pagination approach is suitable in case of list type reports such as a grocery list, product catalog etc. In this case, a new page or sheet is created when its content goes beyond the specified page size. On the other hand, count-per-page pagination is helpful in generating reports such as invoices which require to be presented in fixed format with specified number of records on a worksheet grouped by particular data field. In this case, remaining records are rendered on other worksheet, thereby paginating the worksheet based on the specified number of records.

DsExcel provides various template properties and functions to cater the two pagination approaches. All of these properties and functions work only in the pagination mode, that is when **TemplateOptions.PaginationMode** property is set to **true**. To know more about these properties and functions, see Pagination Properties and Functions.

In a paginated report, name of the page worksheet consists of the name of the template worksheet and its index. For example, if name of the template worksheet is "Sheet1" and it generates two page worksheets, then name of the first page worksheet is "Sheet1_Page1", and that of the second page is "Sheet1_Page2". For more information about pagination mode, see Global Settings.



Note: In pagination mode:

- DsExcel ignores the Pagebreak template property.
- Shapes or pictures repeat on paginated sheets depending on the pagination rule applied to the upper left corner cell of the area where they are present. If the cell appears in a repeating region of the layout, the shape or picture appears as configured.

Limitations

- DsExcel does not support formula, conditional formatting, data validation, tables, charts in the workbook generated using paginated templates.
- Although DsExcel supports formulas starting with "=" in paginated templates, their export to Excel files is not supported. Supported formulas include "sum", "count", "average", "max", "min", "product", "stddev", "stddevp", "var", and "varp".

Pagination Properties and Functions

This topic discusses various template properties and functions with respect to the two pagination approaches, page-size pagination and count-per-page pagination. The table below summarizes the properties and functions required in each

case.

Page-size Pagination	Count-per-page Pagination
Pagination Properties	**Pagination Properties**
RepeatOutput	**CountPerPage**
KeepTogether	**RepeatType**
AttachTo	**RepeatWithGroup**
RepeatWithGroup	**NoRepeatAction**
	Pagination Functions
	PageCount
	PageNumber

Page-size Pagination

In page-size pagination approach, a new page is generated when data rows exceed page size specified for the template document. In this mode, headers and footers are not repeated on every page and appear only once in the beginning and end of the document. Hence, in some cases, they might appear alone on a page depending on the spread of records. Similarly, there are cases of merged cell data spanning across the pages in which the cell data appears only once on the first occurrence of a template cell. To handle these scenarios, DsExcel provides following properties:

RepeatOutput

The RepeatOutput property specifies whether the value of merged cells appears only on the first page or on each page of the report. The property is implemented in a paginated template based on page size.

Value: Boolean

Example: {{ds.Client(RepeatOutput=true)}}

The below image shows how to let the template cell have value on each page if the cell expands to a merged cell across pages. You can download the **Excel template layout** used in the example below.

Template

REPORT		
{{ds.Company(R=B4:D8)}}		
Client{{(RepeatWithGroup=B6)}}	Slip Number	Amount
{{ds.Client(RepeatOutput=true)}}	{{ds.SlipNumber}}	{{ds.Amount}}
Subtotal		{{=SUM(D6)}}
TOTAL		{{=SUM(D7)}}

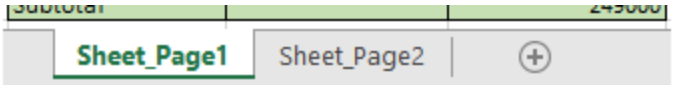
Page1

REPORT		
CompanyA		
Client	Slip Number	Amount
	Aa00001	10000
	Aa00002	12000
	Aa00003	10000
	Aa00004	12000
	Aa00005	10000
ClientA	Aa00006	12000
	Ab00001	10000
	Ab00002	12000
	Ab00003	10000
ClientB	Ab00004	12000
	Ac00001	10000
ClientC	Ac00002	12000
	Ad00001	10000
	Ad00002	12000
	Ad00003	10000
	Ad00004	12000
ClientD	Ad00005	10000
	Aa00002	12000
	Ae00003	10000
	Ae00004	12000
ClientE	Aa00001	10000
Subtotal		230000
CompanyB		
Client	Slip Number	Amount
	Ba00001	20000
ClientA	Ba00002	21000
	Bb00001	20000
	Bb00002	21000
	Bb00003	20000
ClientB	Bb00004	21000
	Bc00005	20000
	Bc00006	22000
	Bc00007	20000
ClientC	Bc00008	22000
Subtotal		249000

Page2

Client	Slip Number	Amount
	Bc00009	20000
ClientC	Bc00010	22000
Subtotal		249000
CompanyC		
Client	Slip Number	Amount
	Ca00005	30000
ClientA	Ca00006	35000
ClientB	Cb00005	20000
	Cc00005	33000
	Cc00006	30000
	Cc00007	20000
ClientC	Cc00008	21000
Subtotal		189000
TOTAL		668000

Sheet_Page1 Sheet_Page2



KeepTogether

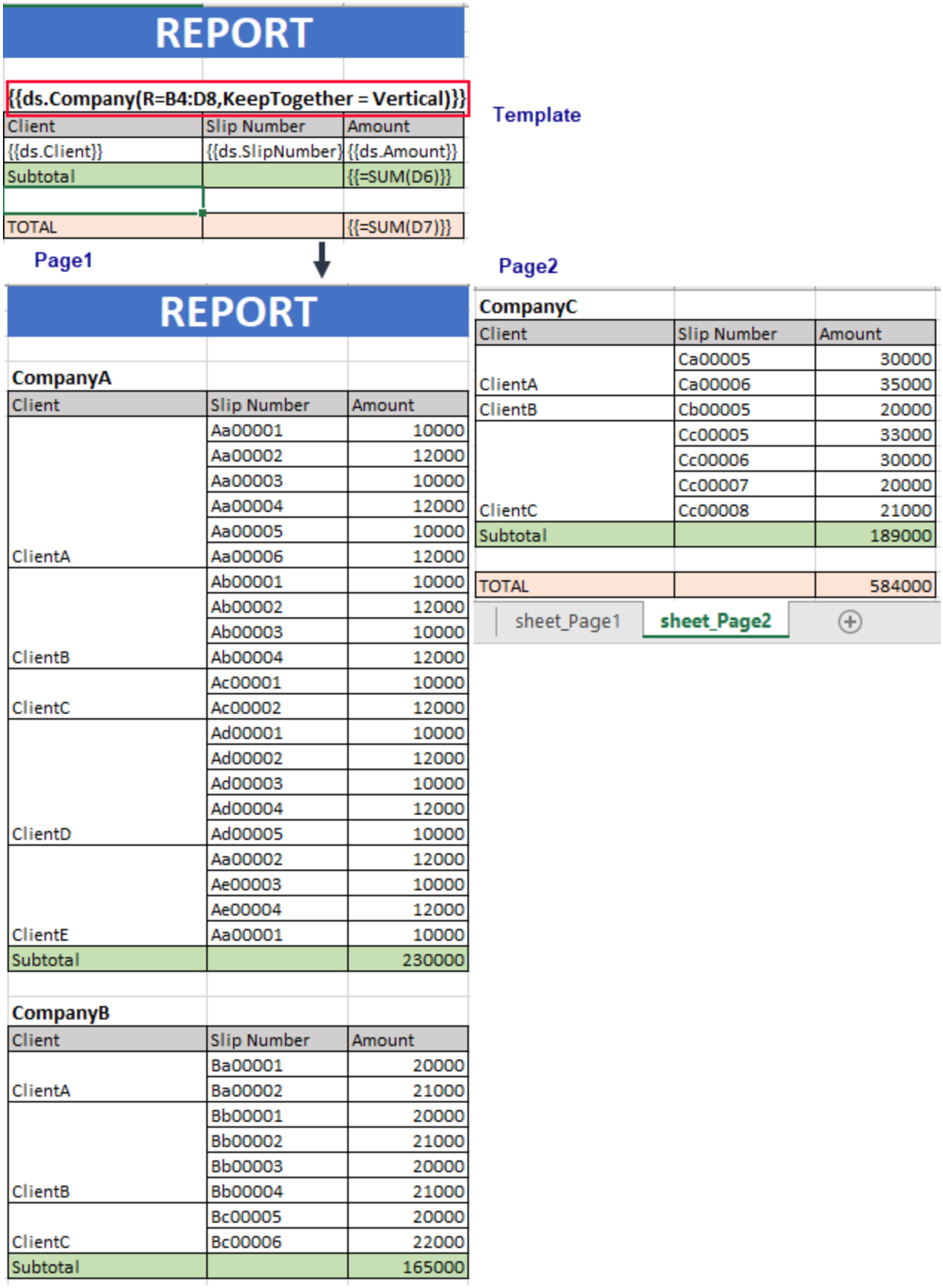
The KeepTogether property ensures the cell, and its descendants appear on the same page. The property allows you to choose if you want to keep the cells together with horizontal pagination or vertical pagination.

Value: Enum

- **None**: (Default value) Places the cell and its descendants according to the availability of space on a page during pagination.
- **Horizontal**: Keeps the cell and its descendants on the same page as much as possible while paginating horizontally.
- **Vertical**: Keeps the cell and its descendants on the same page as much as possible while paginating vertically.
- **Both**: Keeps the cell and its descendants on the same page as much as possible while paginating horizontally or vertically.

Example: {{ds.Company(KeepTogether = Vertical)}}

The below image shows how to keep the grouped data together as much as possible by using the KeepTogether property. You can download the **Excel template layout** used in the example below.



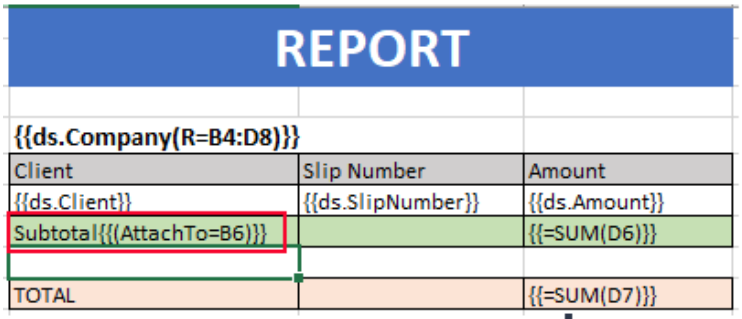
AttachTo

The AttachTo property enables you to bind a cell template with another cell so that the referred cell does not appear alone after pagination. The property takes reference of the cell to be attached to a specific cell.

Value: Cell location

Example: Subtotal{{(AttachTo=B6)}}

The below image shows the usage of AttachTo property wherein it depicts how the cell appears together with the specific cell. You can download the **Excel template layout** used in below example.



REPORT		
{{ds.Company(R=B4:D8)}}		
Client	Slip Number	Amount
{{ds.Client}}	{{ds.SlipNumber}}	{{ds.Amount}}
Subtotal{{(AttachTo=B6)}}		{{=SUM(D6)}}
TOTAL		{{=SUM(D7)}}

Template

Page1

REPORT		
CompanyA		
Client	Slip Number	Amount
ClientA	Aa00001	10000
	Aa00002	12000
	Aa00003	10000
	Aa00004	12000
	Aa00005	10000
	Aa00006	12000
ClientB	Ab00001	10000
	Ab00002	12000
	Ab00003	10000
	Ab00004	12000
ClientC	Ac00001	10000
	Ac00002	12000
ClientD	Ad00001	10000
	Ad00002	12000
	Ad00003	10000
	Ad00004	12000
	Ad00005	10000
ClientE	Aa00002	12000
	Ae00003	10000
	Ae00004	12000
	Aa00001	10000
Subtotal		230000
CompanyB		
Client	Slip Number	Amount
ClientA	Ba00001	20000
	Ba00002	21000
ClientB	Bb00001	20000
	Bb00002	21000
	Bb00003	20000
	Bb00004	21000
ClientC	Bc00005	20000
	Bc00006	22000
Subtotal		165000
CompanyC		
Client	Slip Number	Amount
ClientA	Ca00005	30000

sheet_Page1 | sheet_Page2

Page2

	Ca00006	35000
ClientB	Cb00005	20000
	Cc00005	33000
	Cc00006	30000
	Cc00007	20000
ClientC	Cc00008	21000
Subtotal		189000
TOTAL		584000

sheet_Page1 | sheet_Page2

RepeatWithGroup

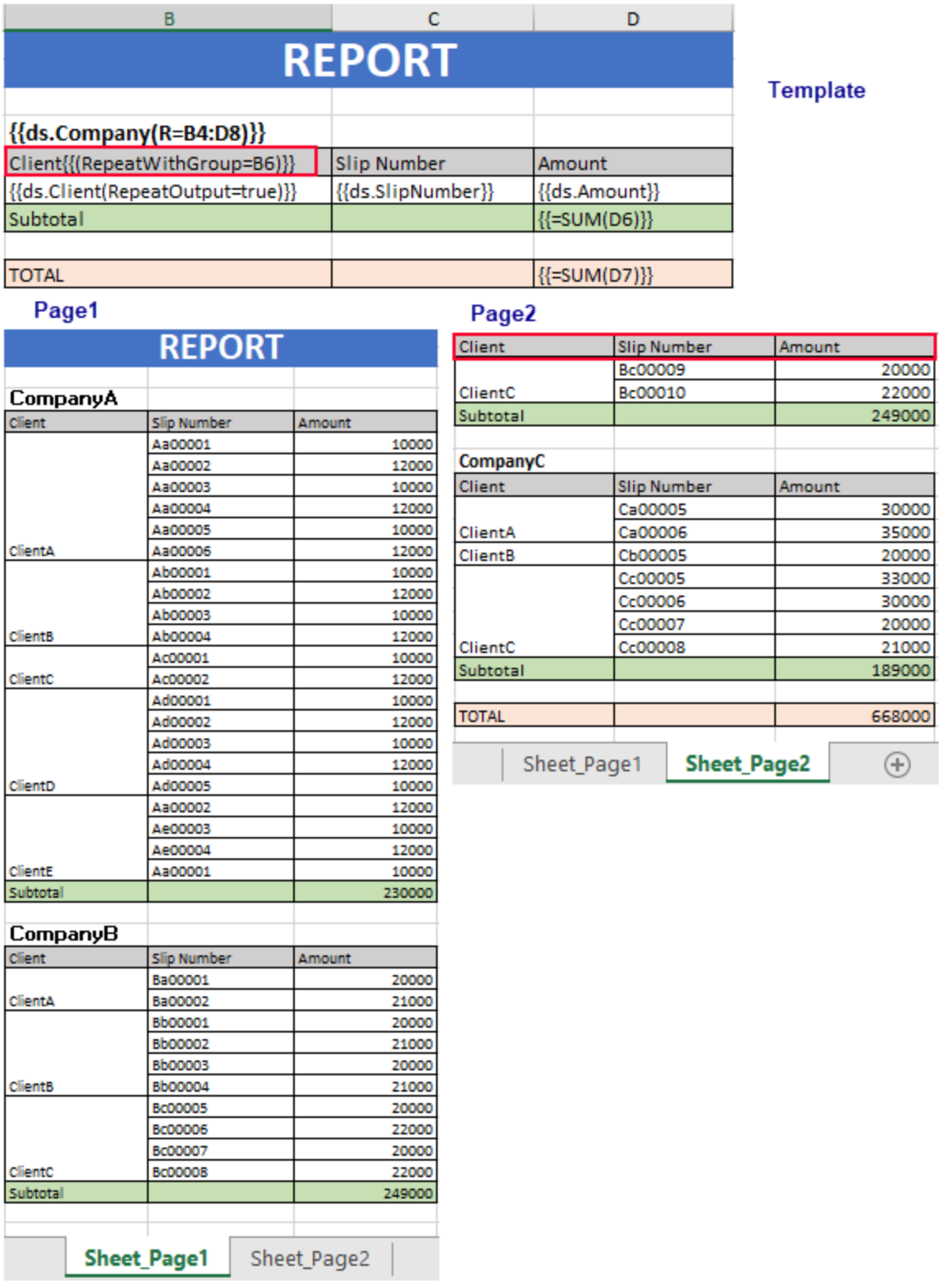
In case of Page-size pagination, the RepeatWithGroup property specifies the cell reference in the template that repeats with a cell in the generated report. For example, you can specify to repeat a group header with the details row on all pages.

Value: Cell location

Example

Client{{(RepeatWithGroup =B6)}}

The below image shows the implementation of the **RepeatWithGroup** property along with the **RepeatOutput**. You can download the **Excel template layout** used in the example below.



B	C	D
REPORT		
{{ds.Company(R=B4:D8)}}		
Client{{(RepeatWithGroup=B6)}}	Slip Number	Amount
{{ds.Client(RepeatOutput=true)}}	{{ds.SlipNumber}}	{{ds.Amount}}
Subtotal		{{=SUM(D6)}}
TOTAL		{{=SUM(D7)}}

Template

Page1

REPORT

CompanyA

Client	Slip Number	Amount
	Aa00001	10000
	Aa00002	12000
	Aa00003	10000
	Aa00004	12000
	Aa00005	10000
ClientA	Aa00006	12000
	Ab00001	10000
	Ab00002	12000
	Ab00003	10000
ClientB	Ab00004	12000
	Ac00001	10000
ClientC	Ac00002	12000
	Ad00001	10000
	Ad00002	12000
	Ad00003	10000
	Ad00004	12000
ClientD	Ad00005	10000
	Aa00002	12000
	Ae00003	10000
	Ae00004	12000
ClientE	Aa00001	10000
Subtotal		230000

CompanyB

Client	Slip Number	Amount
	Ba00001	20000
ClientA	Ba00002	21000
	Bb00001	20000
	Bb00002	21000
	Bb00003	20000
ClientB	Bb00004	21000
	Bc00005	20000
	Bc00006	22000
	Bc00007	20000
ClientC	Bc00008	22000
Subtotal		249000

Sheet_Page1 Sheet_Page2

Page2

Client	Slip Number	Amount
	Bc00009	20000
ClientC	Bc00010	22000
Subtotal		249000

CompanyC

Client	Slip Number	Amount
	Ca00005	30000
ClientA	Ca00006	35000
ClientB	Cb00005	20000
	Cc00005	33000
	Cc00006	30000
	Cc00007	20000
ClientC	Cc00008	21000
Subtotal		189000
TOTAL		668000

Sheet_Page1 Sheet_Page2

Count-per-page pagination

In Count per Page pagination, a new page (or sheet) is generated when data rows exceed a specific count for grouped

data. In this case, every new page has the same layout as that specified by the template.

CountPerPage

DsExcel provides paginated templates using **CountPerPage** template property which specifies the maximum number of template cell instances generated on a page on template expansion. When CountPerPage value is set to "*", there is no limit on number of the tempate cell instances that are generated as long as there is space available on the paper.

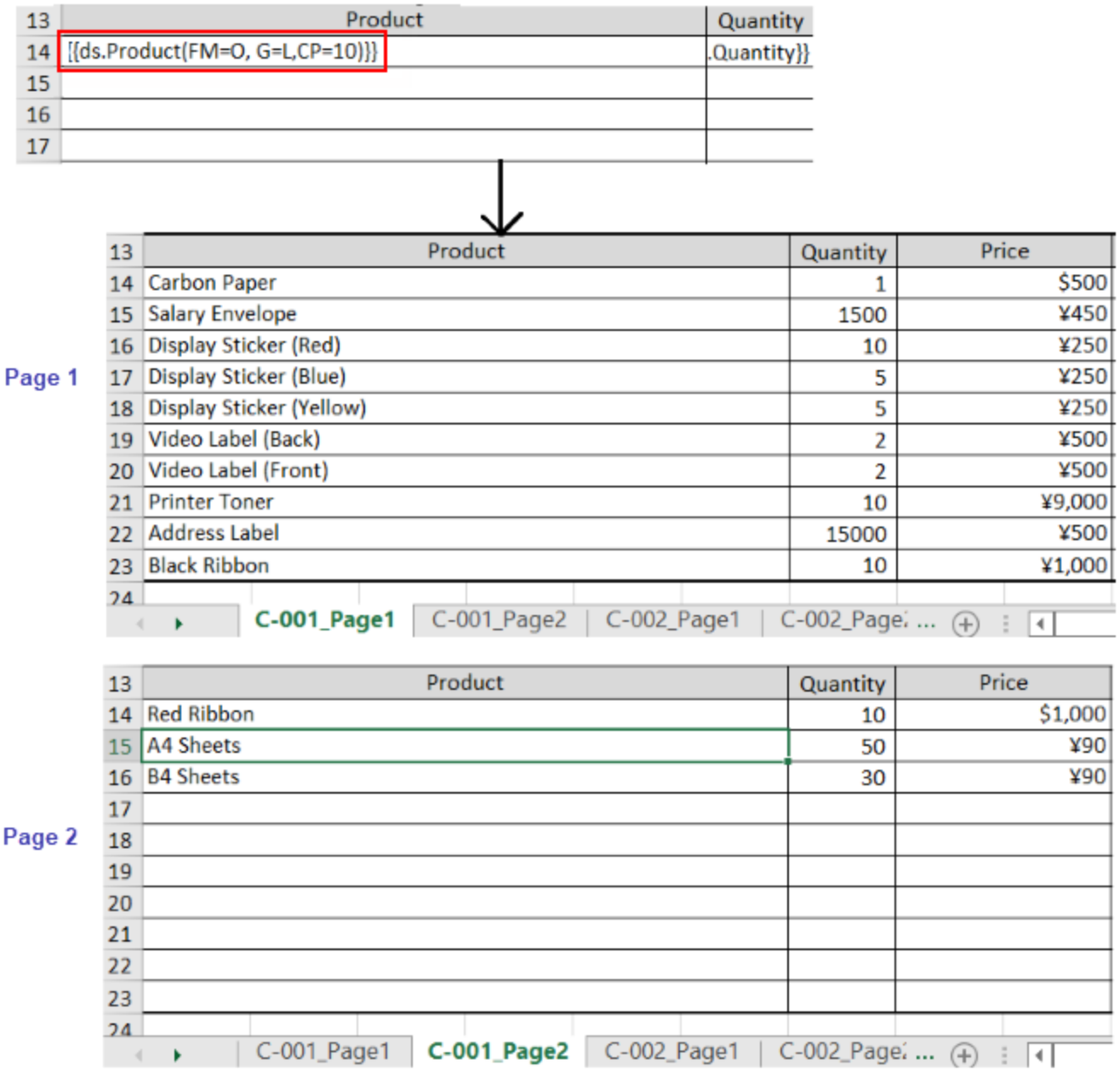
The property automatically creates a new page or worksheet with the same template layout including headers and footers, when the generated record or group count in a report exceeds value of the property. The term 'count' in CountPerPage property refers to the number of records, only when **Group** property of the template is set to **List**. When this property is set to **Normal**, **Merge** or **Repeat**, CountPerPage property considers the count as number of groups. Note that the CountPerPage property is only supported when **TemplateOptions.PaginationMode** is set to **true**.

Value: Integer or '*'

Example

{{ds.Product(FM=O, G=L, CP=10)}}

The image below shows how to apply CountPerPage template property when grouped on **List** basis. You can download the **Excel template layout** used in the example below.



Note:

- **CountPerPage** property can only be set for one template cell in the template worksheet.
- When CountPerPage is implemented, pagination by paper size is ignored.
- When value for CountPerPage(CP) is set to '*', then FillMode(FM) can only be set to **Insert**.
- When **FillMode** property of a cell is set to **Overwrite** and **CountPerPage** property has a value, the **FillRange** property can be omitted as it is automatically calculated by the engine based on value of the CountPerPage property.

RepeatType

The RepeatType property determines how to repeat a cell value within a group when the **RepeatWithGroup** property is

set.

Value: Enum

- **PerPage**: (Default Value) The template cell is visible on every page.
- **FirstPage**: The template cell is only visible on the first page of the group specified by the RepeatWithGroup property .
- **LastPage**: The template cell is only visible on the last page of the group specified by the RepeatWithGroup property .

Example

F6: {{(R=A6:L15, RepeatType = FirstPage)}}

NoRepeatAction

The NoRepeatAction property determines how to set the deletion mode of common content when it is not displayed on the current page.

Value: Enum

- **ClearCells** : (Default value) If a cell is not repeated, its value and format is deleted automatically.
- **DeleteRows**: If a cell is not repeated, its rows are deleted.
- **DeleteColumns**: If a cell is not repeated, its columns are deleted.

Example

F6: {{(R=A6:L15, RepeatType = FirstPage, RepeatWithGroup =A3, NoRepeatAction = DeleteRows)}}

RepeatWithGroup

In case of count-per-page pagination, the RepeatWithGroup property specifies cell reference of the group with which a particular cell or cell range must be repeated.

Value: Cell location

Example

F6: {{(R=A6:L15, RepeatWithGroup =A3)}}

> **Note**:
>
> - If RepeatType = PerPage, RepeatWithGroup setting is less visible because range of cell is repeated on all pages.
> - If RepeatType = FirstPage or LastPage and RepeatWithGroup is not specified then, range of cell gets displayed only on the first or last page of the entire workbook.
> - If RepeatType = FirstPage or LastPage, RepeatWithGroup is specified, then range of cell gets rendered on first or last page of each group.

The below image shows the implementation of the **RepeatWithGroup** property along with the **RepeatType** and **NoRepeatAction** properties discussed in the sections below. You can download the **invoice template** used in the example below.

Template

{{ds.Customer(R=A1:L29)}}

Invoice

Customer Code : {{ds.CustomerCode}}

{{R=A6:L15, RepeatType=FirstPage, RepeatWithGroup=A3, NoRepeatAction=DeleteRows}}

Street: 4891 Armbrester Drive
City: Anaheim **State:** California
Zip Code: 92801
Tel: 310-499-2375

Thank you for your Time
We will bill you as follows

Price	Tax	Total amount
{{=SUM(J17)}}	{{=A14*0.1}}	{{=A14+E14}}

Product	Quantity	Price	Amount
{{ds.Product(CP=10)}}	Quantity}}	{{ds.Price}}	{{=ds.Quantity*ds.Price}}

{{R=A19:L26, RepeatType=LastPage, RepeatWithGroup=A3, NoRepeatAction=DeleteRows}}

Remarks

Please make the transfer within two weeks.

Information Systems Co. Ltd.

Invoice

Customer Code : C-001

Street: 4891 Armbrester Drive
City: Anaheim State: California
Zip Code: 92801
Tel: 310-499-2375

Thank you for your time
We will bill you as follows:

Price	Tax	Total amount
$83,00,700	$8,30,070	$91,30,770

Page 1

Product	Quantity	Price	Amount
Carbon Paper	1	$500	$500
Salary Envelope	1500	$450	$6,75,000
Display Sticker (Red)	10	$250	$2,500
Display Sticker (Blue)	5	$250	$1,250
Display Sticker (Yellow)	5	$250	$1,250
Video Label (Back)	2	$500	$1,000
Video Label (Front)	2	$500	$1,000
Printer Toner			
Address Label			
Black Ribbon			

C-001_Page1

Information Systems Co. Ltd.

Invoice

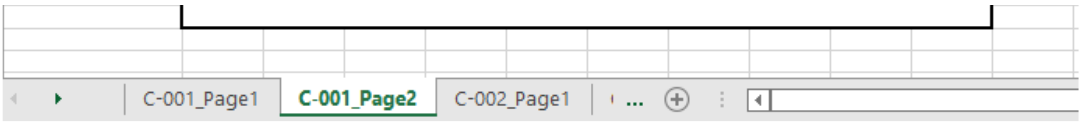
Product	Quantity	Price	Amount
Red Ribbon	10	$1,000	$10,000
A4 Sheets	50	$90	$4,500
B4 Sheets	30	$90	$2,700
Eraser	20	$50	$1,000

Remarks

Please make the transfer within two weeks.

<Transfer Destination>
Wells Fargo Bank

Page 2



PageNumber

PageNumber function is used to add the current page available in the scope of the group.

Syntax: {{=PageNumber(A3)}}

Where in: **PageNumber(string cell = null):** If cell is not passed, the result is index of the current sheet in the workbook.

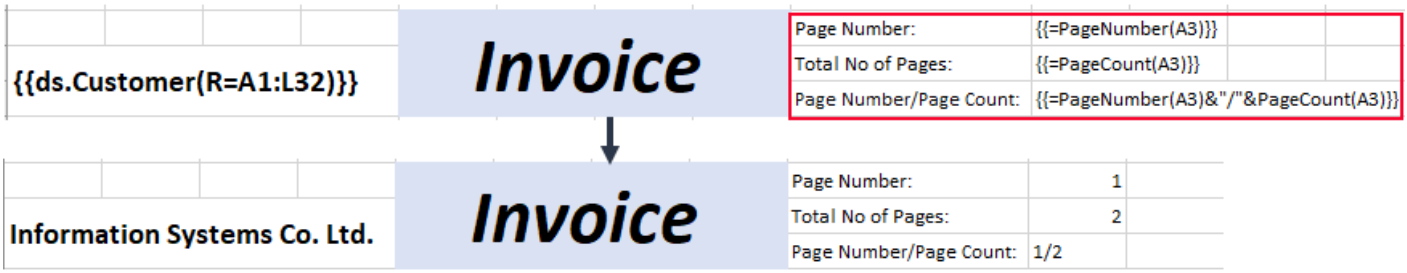
PageCount

PageCount functions is used to add total number of pages available in the current group scope of the group.

Syntax: {{=PageCount(A3)}}

Where in: **PageCount (string cell = null):** If cell is not passed, the result is sheet count of the workbook.

The below image shows the implementation of **PageNumber** and **PageCount** functions while generating an invoice. You can download the **Excel template layout** used in below example.



Limitation

The PageNumber and PageCount functions cannot be mixed with other methods such as Count or Sum etc.

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}}

DsExcel 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. DsExcel 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);
```

JSON

DsExcel allows you to create a new instance of JsonDataSource class as a custom object. Hence, users with json as their data source can directly fetch data from json file and construct a JsonDataSource from the json text and then use the JsonDataSource for the template.

This eradicates the need to create a mapping class to fetch the data from Json and user can directly use a field or member of the json as given in template syntax below:

Template Syntax

[Alias of data source].[Field name]

For example:

{{ds.student.family.father.name}}

{{ds.student.family.father.occupation}}

{{ds.student.family.mother.name}}

Sample JSON for Reference

```
{
  "student": [
    {
      "name": "Jane",
      "address": "101, Halford Avenue, Fremont, CA",
      "family": [
        {
          "father": {
            "name": "Patrick James",
            "occupation": "Surgeon"
```

```
      },
      "mother": {
        "name": "Diana James",
        "occupation": "Surgeon"
      }
    },
    {
      "father": {
        "name": "father James",
        "occupation": "doctor"
      },
      "mother": {
        "name": "mother James",
        "occupation": "teacher"
      }
    }
  ]
},
{
  "name": "Mark",
  "address": "101, Halford Avenue, Fremont, CA",
  "family": [
    {
      "father": {
        "name": "Jonathan Williams",
        "occupation": "Product Engineer"
      },
      "mother": {
```

```
            "name": "Joanna Williams",
            "occupation": "Surgeon"
          }
        }
      ]
    }
  ]
}
```

Bind JSON datasource

C#

```
// Load json text
var jsonText = File.OpenText("Template_FamilyInfo.json").ReadToEnd();

// Create a JsonDataSource
var datasource = new JsonDataSource(jsonText);

// 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);
```

Cancel Template Processing

DsExcel allows you to cancel **ProcessTemplate** method by using an overload of **ProcessTemplate** method that takes a parameter of **CancellationToken** type. The cancellation is thread-safe, i.e., the cancellation request is sent by a thread that does not own the workbook. You can use the following methods of **CancellationTokenSource** to send signals to CancellationToken:

Methods	Description
Cancel	Cancels the method immediately.
CancelAfter	Cancels the method after specific delay time.
Dispose	Releases all resources used by current instance.

The overload of ProcessTemplate method cancels the process when the following conditions are met:

- The CancellationTokenSource of CancellationToken requests the cancellation.
- The internal data structures of ProcessTemplate method are consistent, i.e., it is safe to continue using the workbook object when the cancellation request is handled.
- ProcessTemplate method is in progress.

> **Note:** You must decide whether to accept the partially expanded template or revert to the previous state. If you want to revert to the previous state, you must serialize the workbook before calling the ProcessTemplate method and then deserialize after canceling the operation.

Refer to the following example code to cancel template processing on request or after a timeout is reached:

C#

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

// Load template file from resource.
workbook.Open("Template_SalesDataGroup_DataTable.xlsx");

Console.WriteLine("Creating test data.");

// Add data to DataTable.
var datasource = new System.Data.DataTable();
datasource.Columns.Add(new DataColumn("Area", typeof(string)));
datasource.Columns.Add(new DataColumn("City", typeof(string)));
datasource.Columns.Add(new DataColumn("Category", typeof(string)));
datasource.Columns.Add(new DataColumn("Name", typeof(string)));
datasource.Columns.Add(new DataColumn("Revenue", typeof(double)));
```

```
var areas = new[] { "North America", "South America" };
var cities = new[] { "Chicago", "New York", "Santiago", "Quito", "Fremont", "Buenos
Aires", "Medillin", "Minnesota" };
var categories = new[] { "Consumer Electronics", "Mobile" };
var category1NamePrefixes = new[] { "Bose ", "Canon ", "Haier ", "IFB ", "Mi ",
"Sennheiser " };
var category2NamePrefixes = new[] { "iPhone ", "OnePlus ", "Redmi ", "Samsung " };

Random rand = new Random();
var rows = datasource.Rows;

// You can increase the loop count if the demo is too fast on your computer.
for (var i = 0; i < 50000; i++)
{
    var area = areas[rand.Next(0, areas.Length)];
    var city = cities[rand.Next(0, cities.Length)];
    var categoryId = rand.Next(0, categories.Length);
    var category = categories[categoryId];
    var names = (categoryId == 0) ? category1NamePrefixes : category2NamePrefixes;
    var name = names[rand.Next(0, names.Length)] + rand.Next(10, 10000).ToString();
    var revenue = rand.Next(10000, 100000);
    rows.Add(area, city, category, name, revenue);
}

// Add template global settings.
workbook.Names.Add("TemplateOptions.KeepLineSize", "true");

// Add data source.
workbook.AddDataSource("ds", datasource);

// Cancel data source binding when cancel key is pressed or timeout is reached.
using (CancellationTokenSource cancellation = new CancellationTokenSource())
{
    void cancelHandler(object sender, ConsoleCancelEventArgs e)
    {
        // Exit the process.
        e.Cancel = true;

        // Prevent entering cancelHandler when ProcessTemplate is completed or canceled.
#if NETCOREAPP3_0_OR_GREATER
        Console.CancelKeyPress -= cancelHandler;
#endif
        // Cancel when cancel key is pressed.
        cancellation.Cancel();
    };
    Console.CancelKeyPress += cancelHandler;
```

```
    // Cancel when timeout is reached.
    cancellation.CancelAfter(TimeSpan.FromSeconds(10));
    Console.WriteLine("Start ProcessTemplate.");
    try
    {
        workbook.ProcessTemplate(cancellation.Token);
        Console.WriteLine("ProcessTemplate finished.");
    }
    catch (OperationCanceledException ex) when (ex.CancellationToken ==
cancellation.Token)
    {
        Console.WriteLine("ProcessTemplate was canceled.");
    }

    // Prevent entering cancelHandler when ProcessTemplate is completed or canceled.
#if NETCOREAPP3_0_OR_GREATER
    Console.CancelKeyPress -= cancelHandler;
#endif
}

// Save the workbook.
workbook.Save("CancelTemplateProcessing.xlsx");
```

Create Excel Report using Template

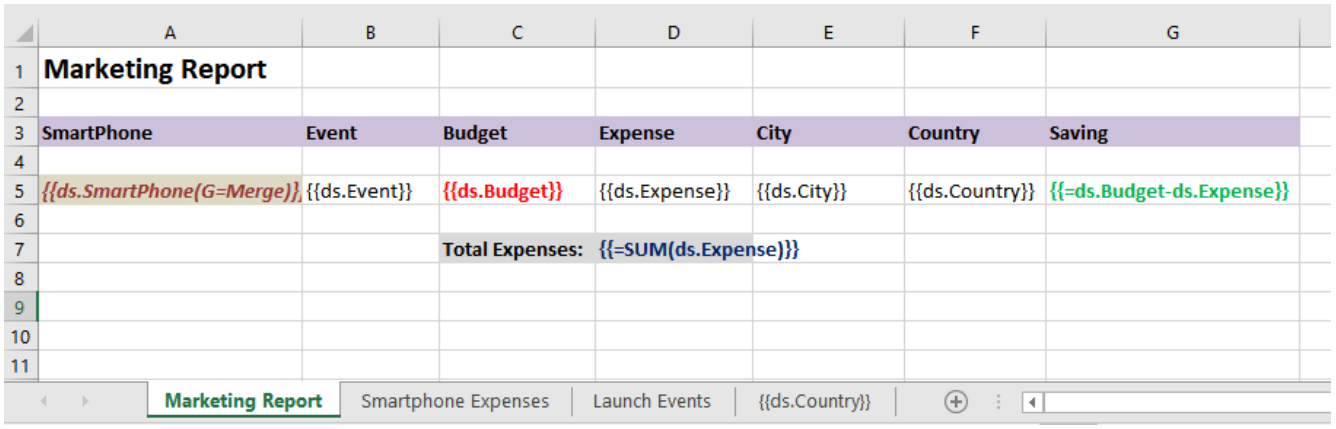
This walkthrough considers the use case to create a Marketing Report of a company which is launching a new series of smartphones. Hence, an Excel report for the planned marketing activities needs to be created. The report details out the planned events for the launch, its budget and expenses. The datasource 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:
    - **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.
    - **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.
    - **Expression Fields:** Specify the functions in fields whose value will be calculated using formulas.
    - **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

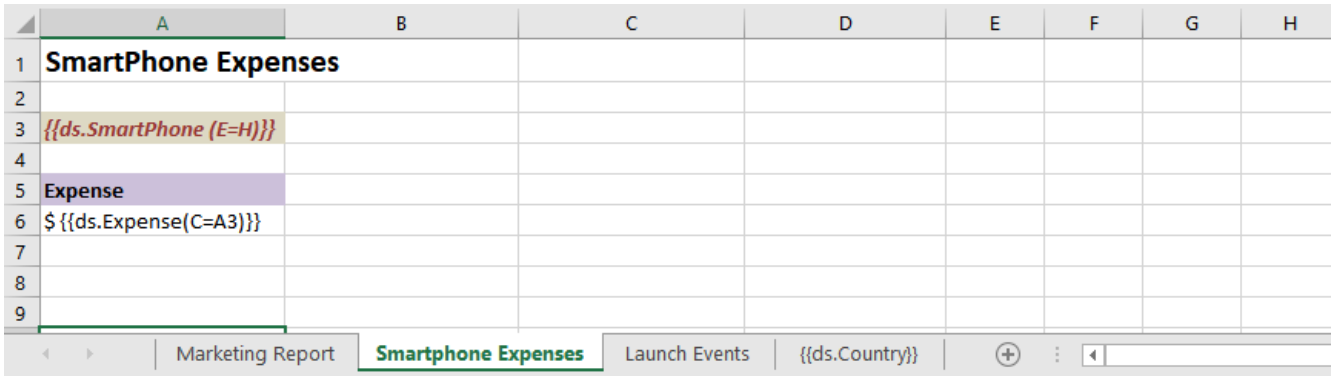
    The below layout uses the Group property (G=Merge), which will group the smartphones against the corresponding records by displaying it once per group. The merge value merges the cells of each group.



Template Layout: SmartPhone Expenses

The below layout uses two template properties, Cell expansion (E=H) and Cell context (C=A3)

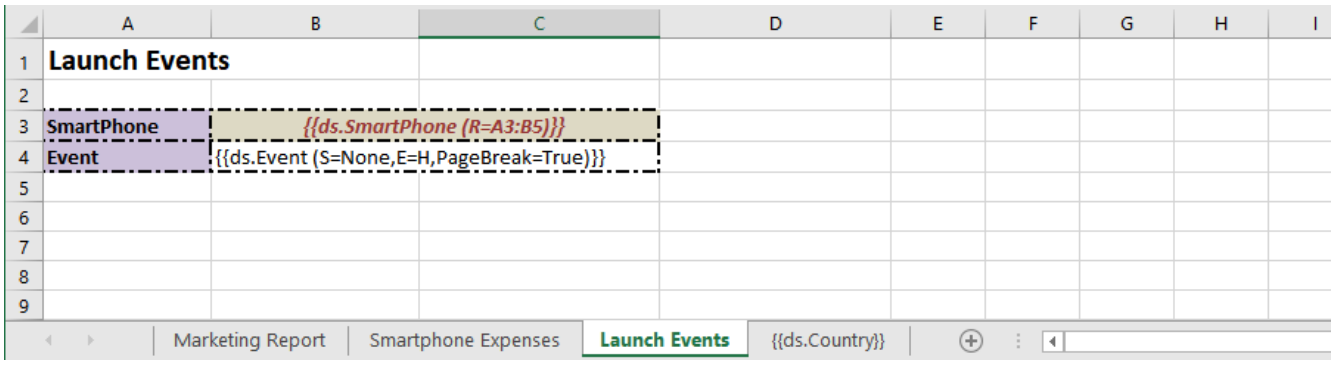
- The cell expansion property will expand the smartphone field horizontally.
- The cell context property will make sure that the expense field expands horizontally depending upon the smartphone field.



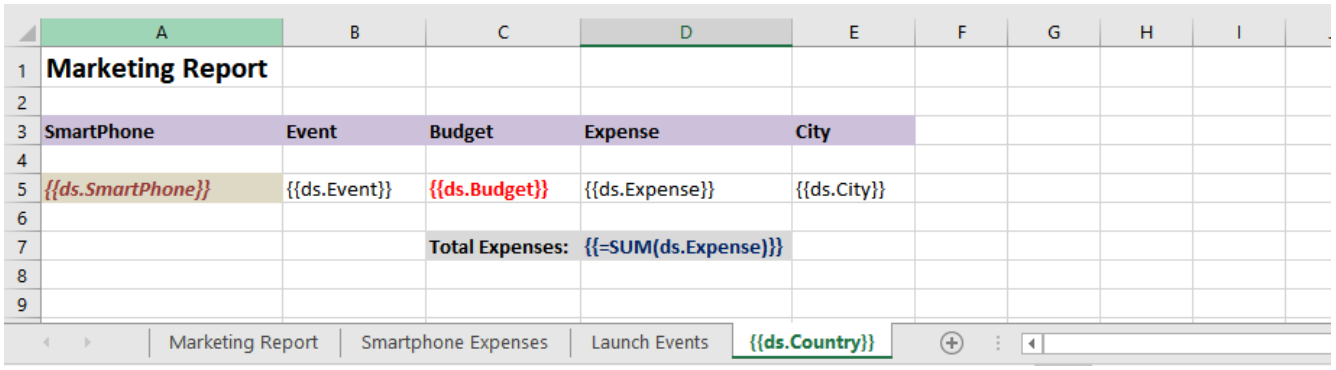
Template Layout: Launch Events

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



Template Layout: {{ds.Country}}



	A	B	C	D	E
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}}

2. Load the template in DsExcel.

C#

```
Console.WriteLine("Generating Marketing Report using 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);
```

3. Configure DataSource and add DataColumns 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 DataColumns 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 DsExcel, 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");
```

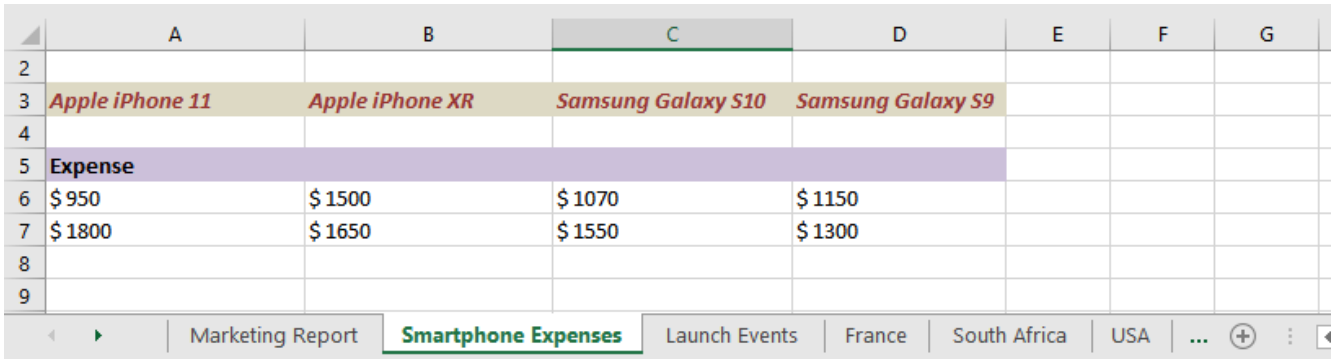
The output of the Marketing Report is shown as below.

Excel Report: Marketing Report

	A	B	C	D	E	F	G	H
1	**Marketing Report**							
2								
3	**SmartPhone**	**Event**	**Budget**	**Expense**	**City**	**Country**	**Saving**	
4								
5		CEO Meet	2000	1800	New York	USA	200	
6	*Apple iPhone 11*	Phone Launch	1000	950	Seattle	USA	50	
7		CEO Meet	1600	1500	New Jersey	USA	100	
8	*Apple iPhone XR*	Phone Launch	1800	1650	Cape Town	South Africa	150	
9		CEO Meet	1600	1550	Paris	France	50	
10	*Samsung Galaxy S10*	Phone Launch	1100	1070	Durban	South Africa	30	
11		CEO Meet	1200	1150	Seattle	USA	50	
12	*Samsung Galaxy S9*	Phone Launch	1500	1300	Paris	France	200	
13								
14			**Total Expenses:**	**10970**				
15								
16								

Marketing Report | Smartphone Expenses | Launch Events | France | South Africa | USA | ...

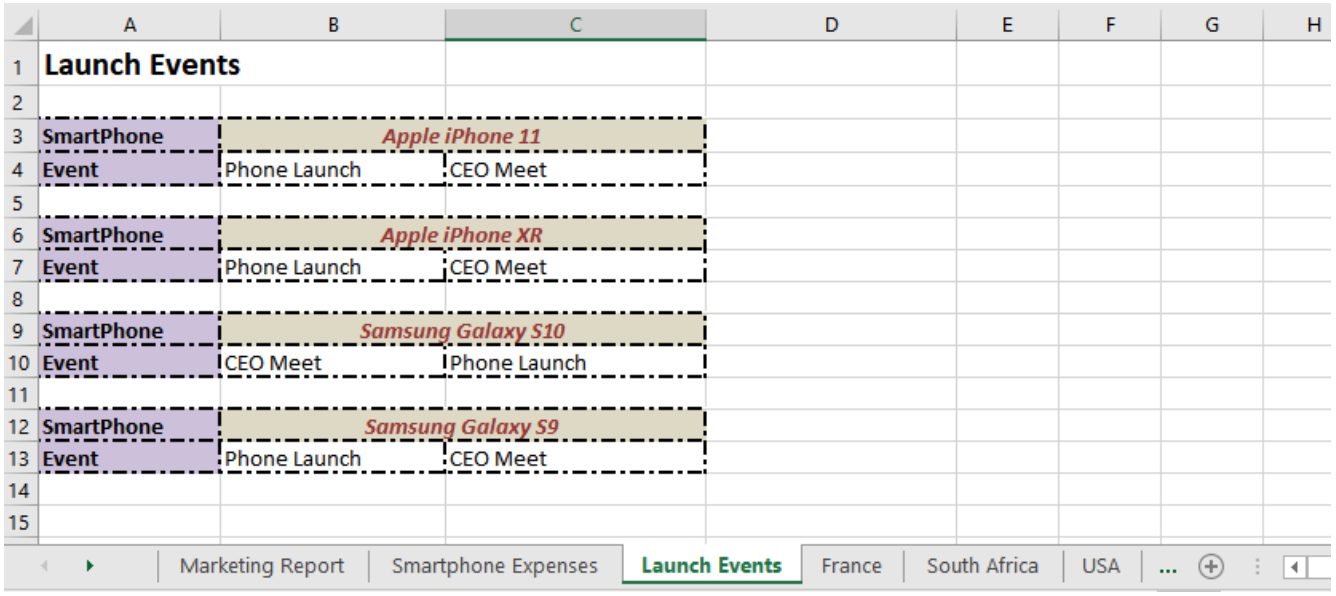
Excel Report: Smartphone Expenses



	A	B	C	D
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 | ...

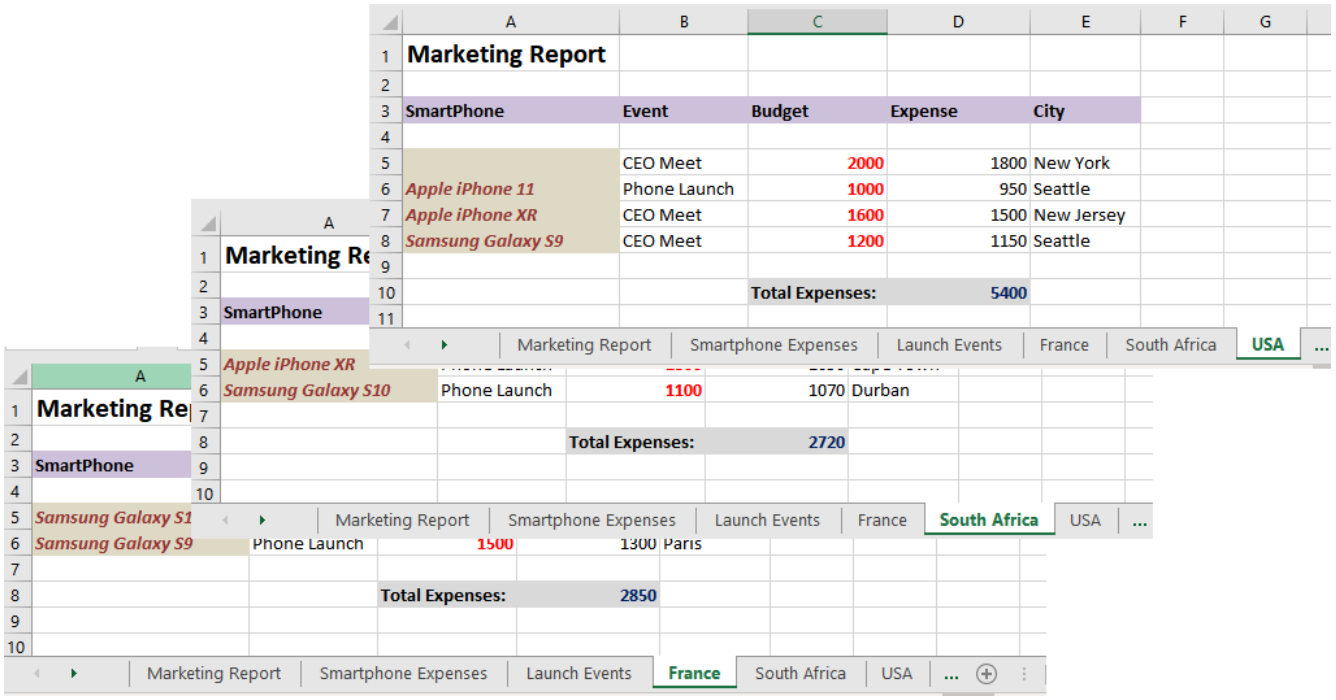
Excel Report: Launch Events



	A	B	C
1	**Launch Events**		
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)



In the above process, the template is overwritten while generating the Excel report. To create a report while keeping the template intact, see the **Create Report While Retaining Template** section below.

Create Report While Retaining Template

DsExcel provides **GenerateReport** method in the **IWorkbook** interface that returns an instance of a new workbook report while retaining the template. The method also provides an overload so that you can generate report for a specific worksheet only.

To generate report using the GenerateReport method, see the example code below:

C#

```
//Process the template and return the instance of report workbook
IWorkbook report = workbook.GenerateReport();

// Process the template for a specific worksheet only
// IWorkbook report = workbook.GenerateReport(workbook.Worksheets["Sales"]);
```

To view the code in action, see Generate Report demo.

File Operations

DsExcel .NET allows users to export (save) data from a spreadsheet into several different file types (.xlsx, .xlsm, .xltx, .csv, .pdf, .json, and .sjs files) and import data (open) files from several different file types (.xlsx, .xlsm, .xltx, .csv, .json, and .sjs files) into DsExcel .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 DsExcel.NET:

- Import and Export .xlsx Document
- Export to PDF
- Export to a HTML File
- Import and Export CSV File
- Import CSV File with Custom Parser
- Import and Export CSV File with Delimiters
- Import and Export JSON Stream
- Import and Export SpreadJS Files
- Import and Export Macros
- Import and Export Excel Templates
- Import and Export OLE Objects
- Convert to Image
- Import and Export Excel Options

> **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 DsExcel .NET handles the spreadsheet documents (.xlsx files).

When you create a workbook using DsExcel .NET and save it, you automatically export it to an external location or folder. While opening or loading a file in DsExcel .NET, you can choose whether to import whole model of the target spreadsheet, or just bring in only data. DsExcel .NET provides **Open** method to open a file with various import flags that can be accessed through **ImportFlags** property of the **XlsxOpenOptions** class. Similarly, to export a workbook as .xlsx file, you can use the **Save** method and provide various save option provided by DsExcel to specify what to skip and what to export. For more information about import and export options provided by DsExcel, see Import and Export Excel Options.

Refer to the following example code in order to import and export .xlsx document from the file name:

C#

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

// Open xlsx file.
workbook.Open(Path.Combine("Resource", "Basic sales report1.xlsx"),
OpenFileFormat.Xlsx);

// Save workbook as xlsx file.
```

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

Refer to the following example code in order to import and export .xlsx document from a file stream:

C#

```
// Create a new file stream to open a file.
using FileStream openFile = new FileStream(Path.Combine("Resource", "Basic sales
report1.xlsx"), FileMode.OpenOrCreate, FileAccess.Read);

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

// Open xlsx file.
streamworkbook.Open(openFile, OpenFileFormat.Xlsx);

// Create a new file stream to save a file.
using FileStream saveFile = new FileStream("Exported-Stream.xlsx",
FileMode.OpenOrCreate, FileAccess.ReadWrite);

// Save workbook as xlsx file.
streamworkbook.Save(saveFile, SaveFileFormat.Xlsx);
```

As another common scenario, you might need to import only data from a spreadsheet or a cell range. To handle such scenarios, DsExcel.NET provides ImportData method to facilitate efficient loading from an external worksheet or a cell range. For more information about importing data only, see **Import Data Only** section below.

Import Data Only

To import only data from a specified worksheet or a cell range, DsExcel.NET provides **ImportData** method which simply opens the worksheet and fetches the data for you. This method is useful in scenarios where only data is required and you do not need to deal with rest of the object model. The **ImportData** method uses name of the file or filestream and source name as main parameters. You can specify name of a worksheet, table or a range as the source of data. To fetch names of sheets and tables used in a file or file stream, the **Workbook** class provides **GetNames** method which returns an array of possible source names.

C#

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

// Open an excel file.
var fileStream = GetResourceStream("xlsx\\AgingReport.xlsx");

// Get the possible import names in the file.
// The names[0] and names[1] are sheet names: "Aging Report", "Invoices".
// The names[2] and names[3] are table names: "'Aging Report'!tblAging",
"Invoices!tblInvoices".
var names = GrapeCity.Documents.Excel.Workbook.GetNames(fileStream);
```

```
// Import the data of a table "'Aging Report'!tblAging" from the fileStream.
var data = GrapeCity.Documents.Excel.Workbook.ImportData(fileStream, names[2]);

// Assign the data to current workbook.
workbook.Worksheets[0].Range[0, 0, data.GetLength(0), data.GetLength(1)].Value = data;

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

While working with heavy files having multiple sheets, or many formulas, you may optimize the load performance by using ImportData method as it reads only data. The method also provides overloads where you can specify the range of target cells and can read that particular part only, even if your file contains huge amounts of data.

Limitation

- Formula are not taken into consideration while using ImportData method, as CalcEngine does not work in such case. Hence, the cell value is set to null. In case a formula has cached value stored in the file, DsExcel returns that value.
- If the worksheet name contains character !, such as "Sheet!1", the worksheetName cannot be parsed by calling ImportData(worksheetName), and this function returns null.

Note: In version v5, name of the parameter of **ImportData** method has been changed from **worksheetName** to **sourceName**. This has resulted into a breaking change for users using the prior version if their code used parameter name "worksheetName" while calling the ImportData method, For details, see Release Notes.

Export to PDF

DsExcel .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, DsExcel 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);
```

DsExcel 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
- Export Form Controls to Form Fields

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

DsExcel .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, DsExcel .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, DsExcel .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 confirgure 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 Acounting, 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

DsExcel .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 DsExcel .NET, the following properties can be used:

Property	Description
IPivotTable.ShowTableStyleColumnHeaders	This property can be used to get or set whether the column headers should be displayed in the Pivot table.
IPivotTable.ShowTableStyleRowHeaders	This property can be used to get or set whether the row headers should be displayed in the Pivot table.
IPivotTable.ShowTableStyleColumnStripes	This property can be used to get or set whether the banded columns in which even columns are formatted differently from odd columns.
IPivotTable.ShowTableStyleRowStripes	This property can be used to get or set whether the banded rows in which even row are formatted differently from odd rows.
IPivotTable.ShowTableStyleLastColumn	This property can be used to get or set whether to display the grand total columns style.
ITableStyle.ShowAsAvailablePivotStyle	This property can be used to get or set whether the specified style is shown as available in the pivot styles gallery.
IPivotField.NumberFormat	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

DsExcel .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

DsExcel .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。MESCIUS";
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, DsExcel will ignore the "@".

Shrink To Fit With Text Wrap

DsExcel .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 = "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

DsExcel .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

DsExcel .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 **GcPdfDocument.Save()** method.

> **Note:** In order to render Excel cell ranges inside PDF, you should have a valid license for Document Solutions 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 Documents Solutions 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 Documents Solutions 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

DsExcel .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 **GcPdfDocument.Save()** method.

> **Note:** In order to export multiple sheets to one page, you should have a valid license for Document Solutions 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 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

DsExcel .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, DsExcel .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

DsExcel .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];
```

```
IList<RepeatSetting> repeatSettings = new List<RepeatSetting>();

// The title rows of the "B2:F87" is "$2:$2"
RepeatSetting repeatSetting = new RepeatSetting();
repeatSetting.TitleRowStart = 1;
repeatSetting.TitleRowEnd = 1;
repeatSetting.Range = worksheet.Range["B2:F87"];
repeatSettings.Add(repeatSetting);

// The title rows of the "B89:F146" is "$89:$89"
RepeatSetting repeatSetting2 = new RepeatSetting();
repeatSetting2.TitleRowStart = 88;
repeatSetting2.TitleRowEnd = 88;
repeatSetting2.Range = worksheet.Range["B89:F146"];
repeatSettings.Add(repeatSetting2);

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();
worksheet.PageSetup.RightMargin = 10;

// Get the pagination information of the worksheet
IList<PageInfo> pages = printManager.Paginate(worksheet, null, repeatSettings);

// Save the modified pages into PDF file
printManager.SavePDF(@"ManageHeadersOnDifferentPages.pdf", pages);
```

Export Last Page Without Headers

DsExcel .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

DsExcel .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

DsExcel .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

DsExcel .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 = "Google";
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

DsExcel .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 columnIndexs is [9, 13], this means that the horizontal direction is split after
the column 10th and 14th
IList<int> columnIndexs = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Horizontal);
Console.WriteLine("In horizontal direction, page is split after column : " +
columnIndexs[0].ToString() + " & " + columnIndexs[1].ToString());

// The rowIndexs is [50, 79], this means that the vertical direction is split after the
row 51th and 80th
IList<int> rowIndexs = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Vertical);
Console.WriteLine("In vertical direction, page is split after row : " +
rowIndexs[0].ToString() + " & " + rowIndexs[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 columnIndexs is [9, 13], this means that the horizontal direction is split after
the column 10th and 14th.
IList<int> columnIndexs = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Horizontal, keepTogetherRanges, repeatSettings);
// The rowIndexs is [23, 66, 79], this means that the vertical direction is split after
the row 24th, 67th and 80th.
IList<int> rowIndexs = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Vertical, keepTogetherRanges, repeatSettings);

IList<PageInfo> pages = printManager.Paginate(worksheet, keepTogetherRanges,
repeatSettings);
printManager.SavePDF(@"GetPaginationRangesRepeatSettings.pdf", pages);
```

Working With Page Setup

DsExcel .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 factorsas per your preferences while exporting a spreadsheet to a PDF file.

In order to set pagination in a workheet, users can explore the following properties of the **IPageSetup** interface and the **IWorksheet** interface:

Settings	Description
IPageSetup.PaperSize	This property can be used to determine the size of each page. For more information on implementation of this property, refer to Configure Paper Settings.
IPageSetup.Orientation	This property can be used to specify whether the worksheet should be printed in landscape orientation or portrait orientation. For more information on implementation of this property, refer to Configure Page Settings.
IPageSetup.PrintTitleRows	This property can be used to specify the rows that you want to print at the top of each page. For more information on implementation of this property, refer to Configure Rows to Repeat at Top and Bottom.
IPageSetup.PrintTitleColumns	This property can be used to specify the columns that you want to print at the left of each page. For more information on implementation of this property, refer to Configure Columns to Repeat at Left and Right.
IPageSetup.PrintArea	This property can be used to specify the print area in a worksheet. If the print area is not specified by the user, the used range of the sheet is printed by default. For more information on implementation of this property, refer to Configure Print Area.
IPageSetup.Zoom	This property can be used to use the result of zoom in order to paginate a worksheet. For more information on implementation of this property, refer to Configure Paper Settings.
IPageSetup.FitToPagesWide	This property can be used to specify the maximum number of pages for horizontal pagination. After this property is set, the worksheet can be scaled to fit all columns to the pages. For more information on implementation of this property, refer to Configure Paper Settings.
IPageSetup.FitToPagesTall	This property can be used to specify the maximum number of pages for vertical pagination. After this property is set, the worksheet can be scaled to fit all rows to the pages. For more information on implementation of this property, refer to Configure Paper Settings.
IPageSetup.IsPercentScale	This property specifies a boolean value to control how the worksheet is scaled while exporting to PDF. If the value is set to True, you can use the Zoom property of the IPageSetup interface and if the value is set to false, you can use the FitToPagesWide and FitToPagesTall property of the IPageSetup interface. For more information on implementation of this property, refer to Configure Paper Settings.
IWorksheet.HPageBreaks	This property can be used to specify the horizontal page breaks that will split rows to multiple pages. However, this property doesn't work when the property IsPercentScale

	is set to false. For more information on implementation of this property, refer to Configure Page Breaks.
IWorksheet.VPageBreaks	This property can be used to specify the vertical page breaks that will split columns to multiple pages. However, this property doesn't work when the property IsPercentScale is set to false. For more information on implementation of this property, refer to Configure Page Breaks.

For more information on setting pagination, refer to Page Setup.

> **Note:** The Export to PDF feature doesn't support inserting double underline, superscripts and subscripts etc. while working with page set up settings in a spreadsheet.

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 DsExcel 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 DsExcel'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
UserPassword	Gets or sets the user password of the PDF document.
OwnerPassword	Gets or sets the owner password of the PDF document. This password is required to change the permissions for the PDF document.
PrintPermission	Gets or sets the permission to print the PDF document. The default value is true for this property.
FullQualityPrintPermission	Gets or sets the permission to print in high quality. The default value is true for this property, and it only works when **PrintPermission** property is set to true.
ExtractContentPermission	Gets or sets the permission to copy or extract content. The default value is true for this property.
ModifyDocumentPermission	Gets or sets the permission to modify the PDF document. The default value is true for this property.
AssembleDocumentPermission	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 **ModifyDocumentPermission** property to false as well.
ModifyAnnotationsPermission	Gets or sets the permission to modify text annotations and fill the form fields. The default value is true for this property.
FillFormsPermission	Gets or sets the permission to fill the form fields even if the **ModifyAnnotationsPermission** 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 **ModifyAnnotationsPermission** property to false.

Using Code

Refer to the following example to add security options while exporting Excel spreadsheets to PDF documents.

C#

```
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;

    //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: DsExcel uses RC4 encryption with key from 40 to 128 bit length and allows to define additional permission flags.

Support Document Properties

DsExcel 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 a PDF document.

C#

```
//create a pdf file stream
FileStream outputStream = new FileStream("setdocumentpropertiestopdf.pdf",
FileMode.Create);

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

IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A1"].Value = "Documents for Excel";
worksheet.Range["A1"].Font.Size = 25;

DocumentProperties documentProperties = new DocumentProperties
{
    //Sets the name of the person that created the PDF document.
    Author = "Jaime Smith",
    //Sets the title of the  PDF document.
```

```
    Title = "GcPdf Document Info Sample",
    //Do not embed a font.
    EmbedStandardWindowsFonts = false,
    //Set the PDF version.
    PdfVersion = 1.5f,
    //Set the subject of the PDF document.
    Subject = "GcPdfDocument.DocumentInfo",
    //Set the keyword associated with the PDF document.
    Keywords = "Keyword1",
    //Set the creation date and time of the PDF document.
    CreationDate = DateTime.Now.AddYears(10),
    //Set the date and time the PDF document was most recently modified.
    ModifyDate = DateTime.Now.AddYears(11),
    //Set the name of the application that created the original PDF document.
    Creator = "GcPdfWeb Creator",
    //Set the name of the application that created the PDF document.
    Producer = "GcPdfWeb Producer"
};

PdfSaveOptions pdfSaveOptions = new PdfSaveOptions
{
    //Sets the document properties of the pdf.
    DocumentProperties = documentProperties
};

//Save the workbook into pdf file.
workbook.Save(outputStream, pdfSaveOptions);

//close the pdf stream
outputStream.Close();
```

Adjust Column Width and Row Height

DsExcel 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:** DsExcel 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 DsExcel, 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
- Funnel 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 currently 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

Supported features

The below table shows the supported features in different chart types when exported to PDF.

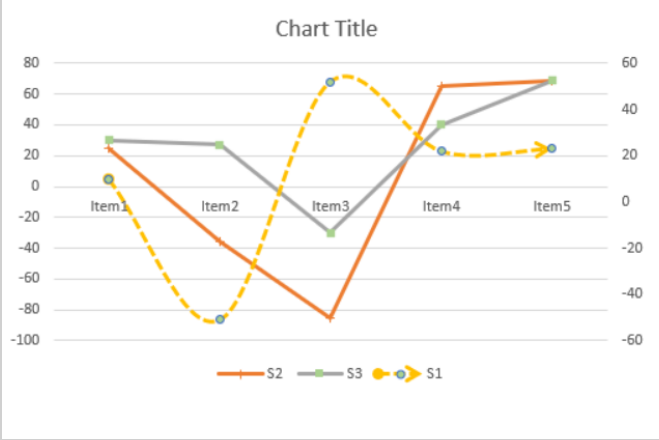
Common Features (Supported by all Chart Types)

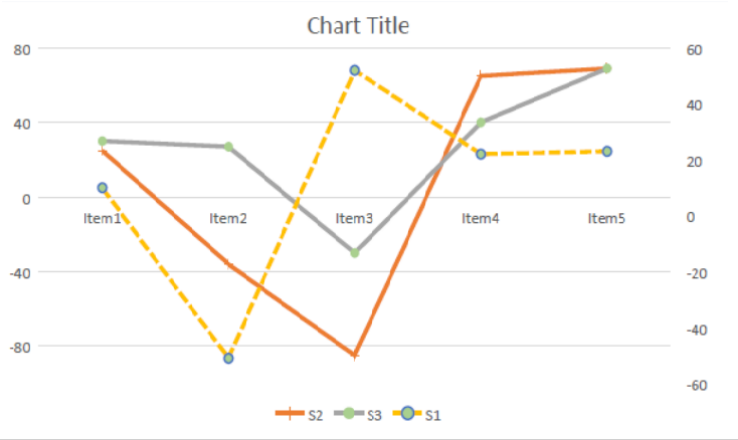
Features	Settings	Supported
Chart Title	font size	Yes
	font color	Yes
	border	Yes
	fill	Yes
	overlap with plot area	No

	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 MSExcel and DsExcel)
	major/minor unit	
	horizontal axis cross position	No
	display units	Yes
	logrithimic 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 (There can be some difference in the exported PDF.)
	end style	
	error amount	
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
	fill	Yes
	border	Yes

Line Chart

Excel Export	PDF Export

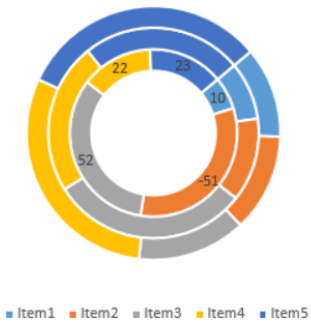
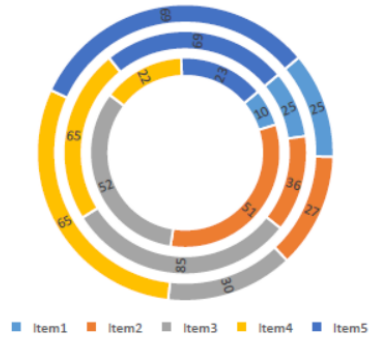




The below table shows the supported features in line chart type when exported to PDF.

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	Yes
Marker	size	Yes
	fill	Yes
	border	Yes
Drop Lines	-	No
High-Low Lines	-	No
Up-Down Bars	-	No

Pie Chart

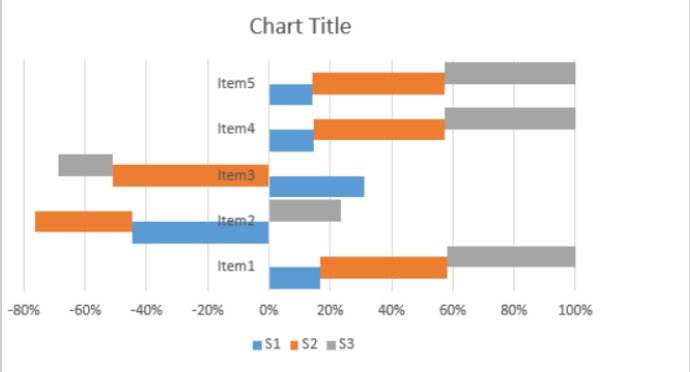
Excel Export	PDF Export
	

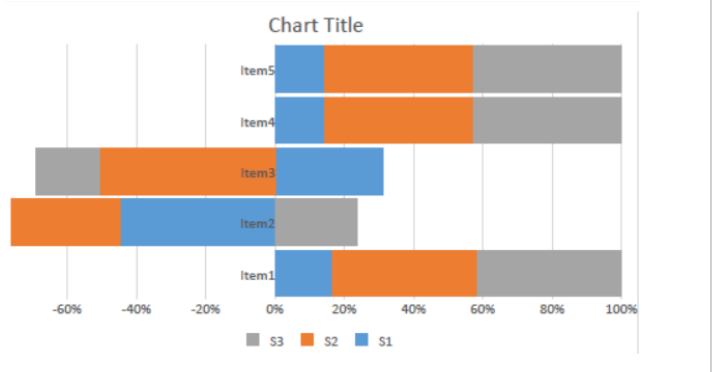
The below table shows the supported features in pie chart type when exported to PDF.

Features	Settings	Supported
Pie Settings	angle of first slice	Yes
	explosion	No
Dought Chart	doughnut hole size	Yes

Bar Chart

Excel Export	PDF Export

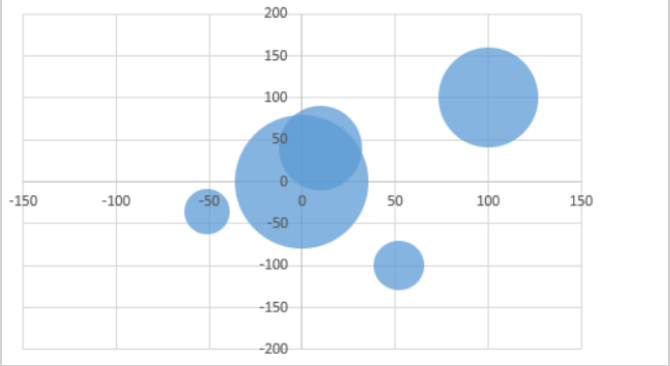
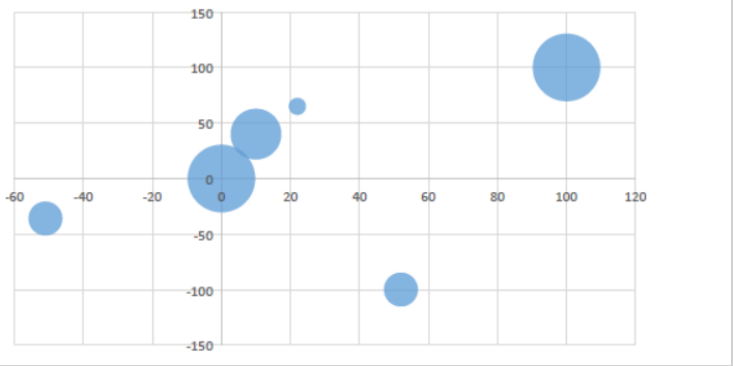




The below table shows the supported features in bar chart type when exported to PDF.

Features	Settings	Supported
Series Option	overlap	No
	gap width	Yes

Scatter Chart

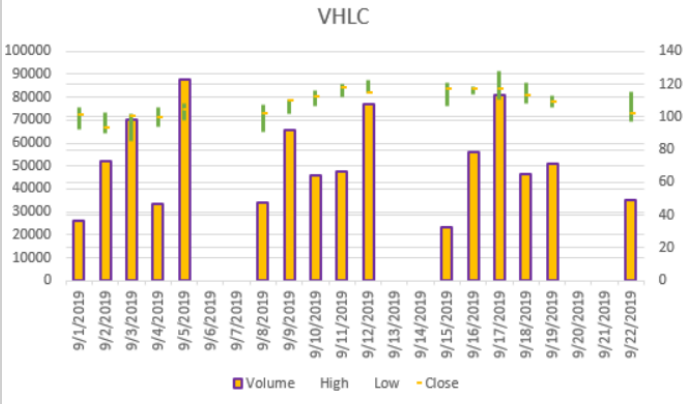
Excel Export	PDF Export
	

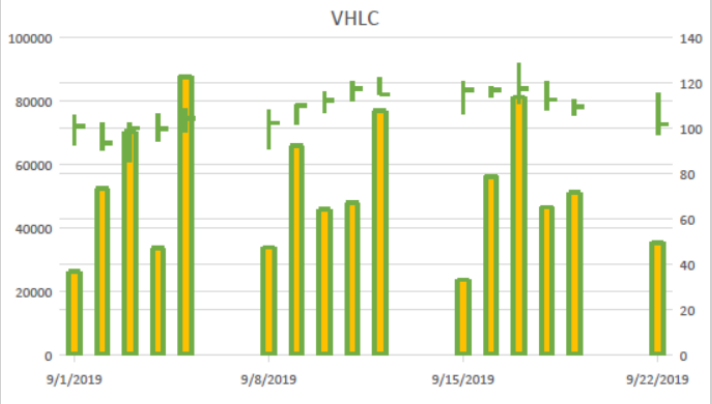
The below table shows the supported features in scatter chart type when exported to PDF.

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
Line	smooth line	Yes

Stock Chart

Excel Export	PDF Export

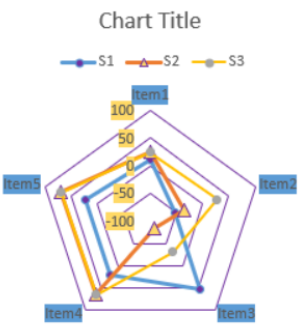
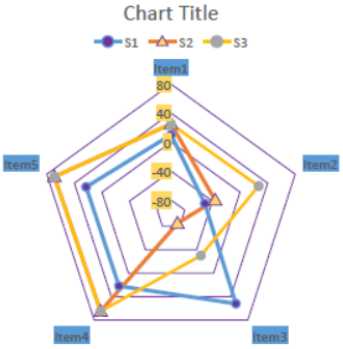




The below table shows the supported features in stock chart type when exported to PDF.

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

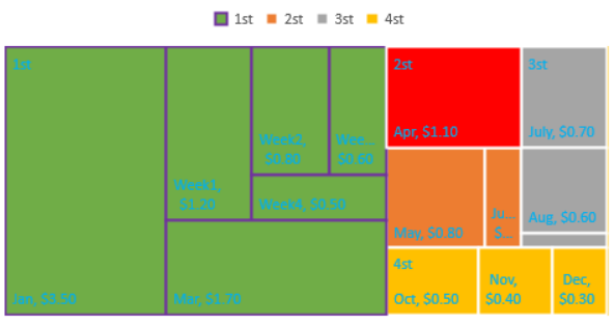
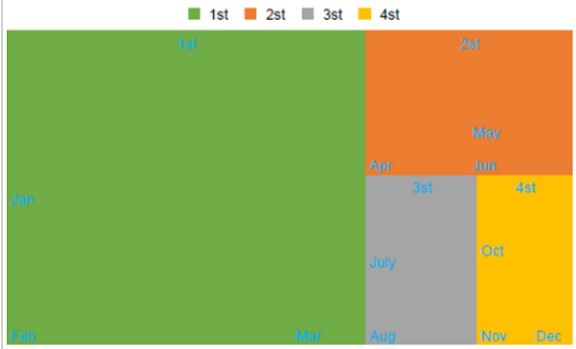
Radar Chart

Excel Export	PDF Export
	

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

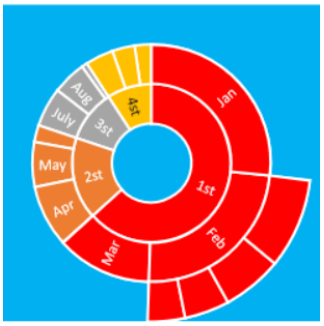
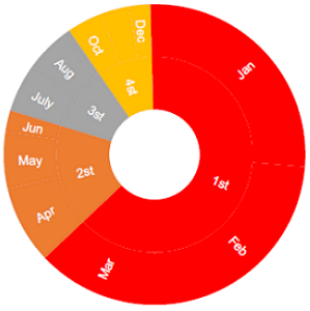
TreeMap Chart

Excel Export	PDF Export
	

The below table shows the supported features in TreeMap chart type when exported to PDF.

Features	Setttings	Supported
Series Option	banner	No
	overlapping	No
Label Option	contains(series name/category name/value)	No
	number fromat	No
	text font	Yes
	text color	Yes
Point Formatting	fill	Yes (Only supports first level points.)
	line	No

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 fromat	No
	text font	Yes
	text color	Yes
Point Formatting	fill	Yes (Only supports first level points.)
	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.

DsExcel 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"], "cate1", "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

DsExcel 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

DsExcel 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");
```

Export Form Controls to Form Fields

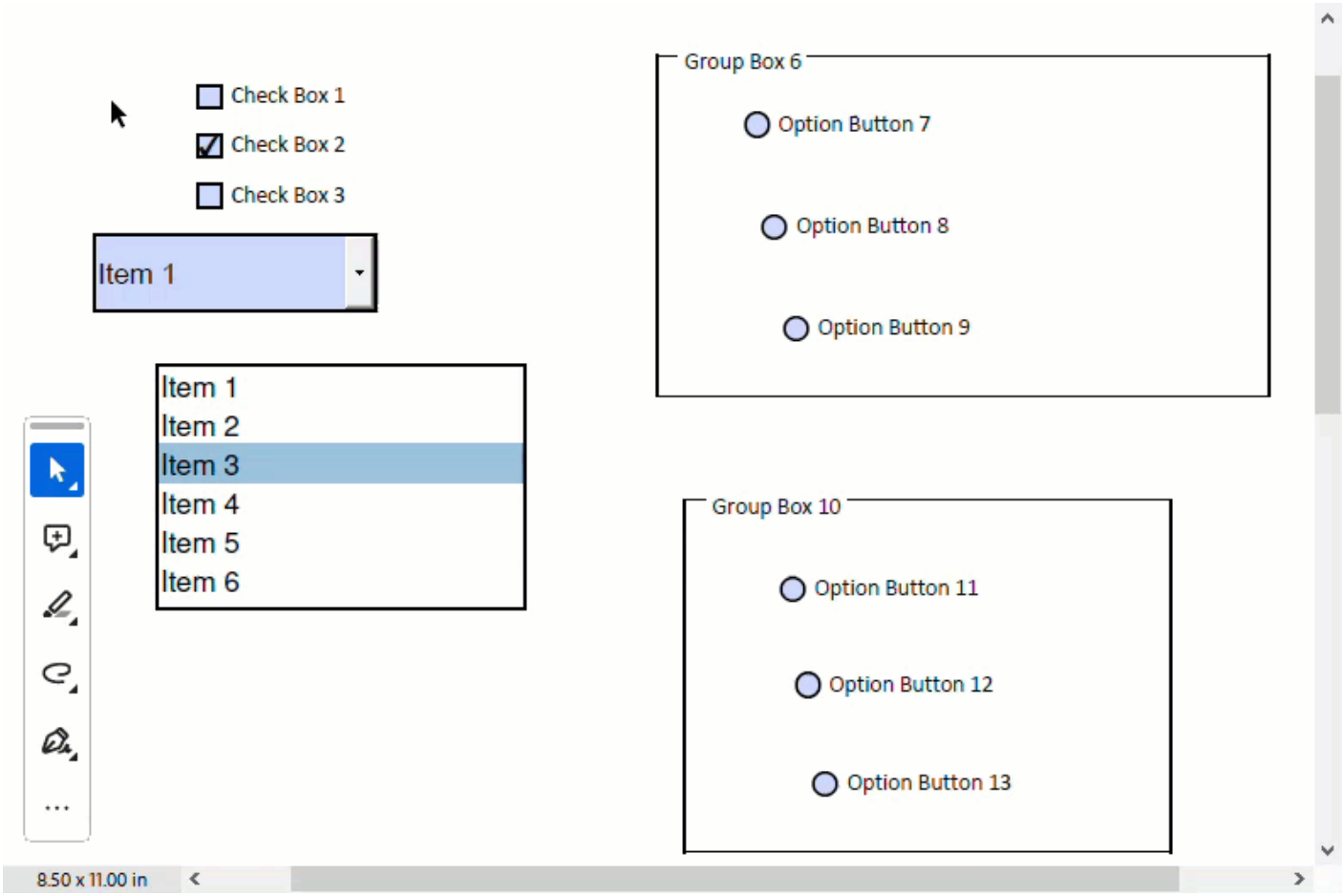
Sometimes you need the Excel form containing form controls as an interactive PDF form (or AcroForms). Form controls are objects that can be added to a worksheet to enable interaction with a cell or a data range available in the worksheet. With **FormFields** property of **PdfSaveOptions** class, DsExcel allows you to export the form controls as form fields to the PDF document, which will be interactive for the user.

Refer to the following example code to export Excel form controls as PDF form fields:

C#

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

// Create worksheet.
IWorksheet ws = workbook.Worksheets["Sheet1"];

// Add three checkboxes.
var checkBox1 = ws.Controls.AddCheckBox(62.4, 16.8, 69, 19.2);
checkBox1.Value = false;

var checkBox2 = ws.Controls.AddCheckBox(62.4, 36.6, 69, 19.2);
checkBox2.Value = true;

var checkBox3 = ws.Controls.AddCheckBox(62.4, 57, 69, 19.2);
checkBox3.Value = false;

// Add dropdown.
var dropDown = ws.Controls.AddDropDown(28.8, 81.8, 103.8, 31.4);
dropDown.PrintObject = true;
dropDown.Items.Add(new DropDownItem("Item 1"));
dropDown.Items.Add(new DropDownItem("Item 2"));
dropDown.Items.Add(new DropDownItem("Item 3"));
dropDown.SelectedIndex = 0;

// Add listbox.
var lstBox1 = ws.Controls.AddListBox(51.6, 134.2, 135, 99.6);
for (int i = 0; i < 6; i++)
{
```

```
    lstBox1.Items.Add(new ListBoxItem("Item " + (i + 1)));
}
lstBox1.SelectedIndex = 2;

// Add option button groups.
ws.Controls.AddGroupBox(234.2, 8.4, 222.6, 138.6);
ws.Controls.AddOptionButton(261.2, 29.4, 71.4, 16.8);
ws.Controls.AddOptionButton(267.8, 70.8, 71.4, 16.8);
ws.Controls.AddOptionButton(275.6, 111.6, 71.4, 16.8);

ws.Controls.AddGroupBox(244.4, 187.6, 176.4, 143.4);
ws.Controls.AddOptionButton(274.4, 216.6, 71.4, 16.8);
ws.Controls.AddOptionButton(279.8, 255, 71.4, 16.8);
ws.Controls.AddOptionButton(286.4, 295.2, 71.4, 16.8);

// Set FormFields to true to export Excel form controls as PDF form fields.
var options = new PdfSaveOptions { FormFields = true };

// Save the PDF document.
workbook.Save("PdfFormFieldExample.pdf", options);
```



Note: The ZIndex of mapped PDF form fields is always higher than other shapes.

If form controls are not present in the exported PDF document, then check and enable **PrintObject** setting of the specific form control. By default, the value of this property is **true** for all form controls except the Button form control.

Refer to the following example code to print a Button form control while saving it as a PDF:

C#

```
// Initialize Workbook.
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
// Add a button control to worksheet and set its PrintObject property
var btn = worksheet.Controls.AddButton(360 * ColScale, 91.8, 103.94 * WidthScale,
19.79);
btn.Text = "Test settings";
btn.PrintObject = true;

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

Limitations

DsExcel does not support the export of the following controls and properties because they are not available in PDF form fields:

- IButton
- IGroupBox
- ILabel
- IScrollBar
- ISpinner
- Mixed option for GrapeCity.Documents.Excel.Forms.ICheckBox.IsChecked
- GrapeCity.Documents.Excel.Forms.SelectionMode.Extended

For more information on exporting form controls to PDF document, see Simulate UI with PDF form fields.

Support Sheet Background Image

DsExcel 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 = "Documents for Excel";
worksheet.Range["A1"].Font.Size = 25;

using (FileStream pictureStream = File.Open(@"background-image.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 DsExcel 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.

DsExcel 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

DsExcel 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.

DsExcel 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

DsExcel 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

DsExcel 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.

DsExcel 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.

CssExportType property in **HtmlSaveOptions** class provides options for a user to decide how to export a CSS file with the HTML file using **CssExportType** enumeration. CssExportType enumeration allows a user to export CSS to a separate file (in the additional folder), within the style tag in HTML, or in the style attribute inside an HTML element.

> **Note:** If unlicensed version of DsExcel 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#

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

// Open en Excel file.
FileStream fileStream = new FileStream("BreakEven.xlsx", FileMode.Open);
workbook.Open(fileStream);

IWorksheet worksheet = workbook.Worksheets[0];

// Initialize HtmlSaveOptions.
HtmlSaveOptions options = new HtmlSaveOptions();
```

```
// Export first sheet.
options.ExportSheetName = workbook.Worksheets[0].Name;

// Set exported image as base64.
options.ExportImageAsBase64 = true;

// Set the css export type to internal CSS.
options.CssExportType = CssExportType.Internal;

// Or, set the css export type to inline CSS.
// options.CssExportType = CssExportType.Inline;

// 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#

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

// Open en Excel file.
FileStream fileStream = new FileStream("BreakEven.xlsx", FileMode.Open);
workbook.Open(fileStream);

IWorksheet worksheet = workbook.Worksheets[0];

// Initialize HtmlSaveOptions.
HtmlSaveOptions options = new HtmlSaveOptions();

// Export first sheet.
options.ExportSheetName = workbook.Worksheets[0].Name;

// Set export area.
options.ExportArea = "D2:G23";

// Set exported image as base64.
options.ExportImageAsBase64 = true;

// Set the css export type to internal CSS.
options.CssExportType = CssExportType.Internal;

// Or, set the css export type to inline CSS.
// options.CssExportType = CssExportType.Inline;

// Set not to export single tab in HTML.
options.ExportSingleTab = false;

// Save the specified range of first worksheet to HTML.
workbook.Save("SaveSpecificRangetoSingleHTML.html", options);
```

Note: ExportCssSeparately property is now obsolete, but the existing applications with this property will continue to function properly.

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 DsExcel .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
CsvOpenOptions.ConvertNumericData	This property can be used to get or set a value that indicates whether the string in text file is converted to numeric data.
CsvOpenOptions.ConvertDateTimeData	This property can be used to get or set a value that indicates whether the string in text file is converted to date data.
CsvOpenOptions.SeparatorString	This property can be used to get or set the string value as a separator.
CsvOpenOptions.Encoding	This property can be used to get or set the default encoding which is UTF-8.
CsvOpenOptions.ParseStyle	This property can be used to specify whether the style for parsed values should be applied while converting the string

	values to number or date time.
CsvOpenOptions.HasFormula	This property can be used to specify whether the text is formula if it starts with "=".
CsvSaveOptions.Encoding	This property can be used to specify the default encoding which is UTF-8.
CsvSaveOptions.ValueQuoteType	This property can be used to get or set how to quote values in the exported text file. **Note:** DsExcel ignores this property when QuoteColumns property is set.
CsvSaveOptions.TrimLeadingBlankRowAndColumn	This property can be used to specify whether the leading blank rows and columns should be trimmed like in Excel.
CsvSaveOptions.QuoteColumns	This property can be used to specify which column values will be in quotes while the values in the remaining columns are not. The column number starts at 0, and specifying an invalid column number has no effect. The column values in quotes are indicated by CellSeparator property, which is typically set to double quote (") by default. **Note:** If the value contains special characters such as quotes or separators, it will be in quotes always.

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;
 options2.QuoteColumns = new int[] { 1, 3, 4 }; // ValueQuoteType is ignored when
QuoteColumns is set.
 worksheet.Save(@"test.csv", options2);
```

Note: SeparatorString property is obsolete. You can use **RowSeparator**, **ColumnSeparator**, and **CellSeparator** properties instead. For more details, see Import and Export CSV File with Delimiters.

Import CSV File with Custom Parser

DsExcel .NET allows you to import CSV files using custom parsing rules to get results in a specific format. For instance, a cell with numeric type data is automatically parsed as a numeric cell. However, in some cases you want to load it as a string. In such cases, you can use this feature to define your own rules when you do not get the desired result with default parser settings of DsExcel.

You can import CSV files with customized parsing rules by implementing **ICsvParser** interface to define custom parsing rules using the **Parse** method. The method accepts objects of **CsvParseResult** and **CsvParseContext** class as parameters. As **CsvParseResult** class represents the parsed text, you can pass the result generated by custom parsing rules to the CsvParseResult object and specify the location and text information of the target cell through the **CsvParseContext** class. Once the ICsvParser interface is implemented, pass it to **Parser** property of the **csvOpenOptions** class to get the expected results on importing the CSV file.

C#

```
// Create CsvOpenOptions and custom parser rules.
CsvOpenOptions csvOpenOptions = new CsvOpenOptions();
csvOpenOptions.Parser = new CustomParser();
```

```
// Open csv file with option.
workbook.Open(fileStream, csvOpenOptions);
```

Custom Parser

```
public class CustomParser : ICsvParser
{
    public void Parse(CsvParseResult csvParseResult, CsvParseContext context)
    {
        if (context.Text.StartsWith("00"))
        {
            csvParseResult.Value = context.Text;
        }
        else if (context.Column == 5 || context.Column == 6)
        {
            csvParseResult.NumberFormat = "#.00";
        }
        else if (csvParseResult.NumberFormat.Equals("m/d/yyyy h:mm"))
        {
            csvParseResult.NumberFormat = "m/d/yyyy";
        }
    }
}
```

Import and Export CSV File with Delimiters

DsExcel .NET allows users to open and save CSV files with custom delimiters for rows, cells and columns. You can use any custom character of your choice as a delimiter. For instance - Comma (,) , Semicolon (;) , Quotes (", ') , Braces ((), {}), pipes (|), slashes (/ \ ), Carat (^), Pipe (|), Tab (\t) etc.

Users can import and export the following three types of custom delimiters in CSV files as per their custom requirements and preferences. All these types of delimiters work independently and cannot be combined with each other.

1. **Column Delimiters** - These are the delimiters that separate the columns of a worksheet. By default, a column delimiter is of string type. Users can get or set the column delimiters using the options in the table shared below.

Settings	Description
CsvOpenOptions.ColumnSeparator	This property can be used to get or set the column delimiter while opening CSV files.
CsvSaveOptions.ColumnSeparator	This property can be used to get or set the column delimiter while saving CSV files.

2. **Row Delimiters** - These are the delimiters that separate the rows of a worksheet. By default, a row delimiter is of string type. Users can get or set the row delimiters using the options in the table shared below.

Settings	Description
CsvOpenOptions.RowSeparator	This property can be used to get or set the row delimiter while opening CSV files.
CsvSaveOptions.RowSeparator	This property can be used to get or set the row delimiter while saving CSV files.

3. **Cell Delimiters** - These are the delimiters that separate the cells of a worksheet. By default, the cell delimiter is of char type. Users can get or set the cell delimiters using the options in the table shared below.

Settings	Description
CsvOpenOptions.CellSeparator	This property can be used to get or set the cell delimiter while opening CSV files.
CsvSaveOptions.CellSeparator	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

DsExcel .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**. The method even lets you export the workbooks having formulas with external reference. 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

//Import an excel file.
//change the path to real source file path.
string source = "savingfile.xlsx";
workbook.Open(source);

//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.
//Use the json string to initialize.
workbook.FromJson(jsonstr);
//Export workbook to an excel file.
//change the path to real export file path.

string export = "export.xlsx";
workbook.Save(export);
```

Note: To get better performance in case of large workbooks with many worksheets having complex formula, you can export and import the workbook and worksheets to separate JSON streams. For more information, see Import and Export from JSON without Worksheets.

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 DsExcel and the json string of a worksheet can be exported using the **ToJson** method. Further, the json string of the worksheet can be transfered 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, DsExcel 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.

//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 } };

    //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

    //Documents for Excel use the json string to update content of the corresponding
worksheet
    new_workbook.Worksheets[worksheet.Name].FromJson(json);
}

//Documents for Excel export workbook to an excel file
string export = "ExcelJsonOutput.xlsx";
new_workbook.Save(export);
```

Retreive Errors while Importing JSON Files

DsExcel 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 DsExcel.

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 from JSON string

DsExcel allows you to import and export below features from or to a json string.

Shape, Chart or Picture

Refer to the following example code which uses **IShape.FromJson** method to update a shape, chart and picture from json string.

C#

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

IWorksheet worksheet = workbook.Worksheets[0];

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}
};

var shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 10, 10, 100, 100);
//update shape from json
shape.FromJson("{\"isLocked\":true,\"canPrint\":true,\"dynamicMove\":true,\"dynamicSize\":true," +
    "\"allowResize\":true,\"allowRotate\":true,\"allowMove\":true,\"showHandle\":true,\"alt\":\"\"," +
    "\"formulaItems\":{\"line\":{\"color\":\"rgb(31,79,122)\",\"lineStyle\":0,\"width\":1,\"capType\":2," +
    "\"joinType\":0,\"transparency\":0}},\"shapeData\":{\"anchorType\":0,\"startPoint\":{\"row\":1," +
    "\"col\":0,\"rowOffset\":11,\"colOffset\":38},\"endPoint\":{\"row\":8,\"col\":4,\"rowOffset\":2," +
    "\"colOffset\":27},\"editAs\":0,\"sp\":{\"shapeType\":5,\"nvSpPr\":{\"cNvPr\":{\"id\":2,\"name\":" +
    "\"rightArrowCallout 1\",\"hidden\":false,\"title\":\"\"},\"cNvSpPr\":{\"txBox\":false}},\"spPr\"" +
    ":{\"xfrm\":{\"flipH\":false,\"flipV\":false,\"rot\":0,\"off\":{\"x\":38,\"y\":31},\"ext\":{\"cx\"" +
    ":237,\"cy\":131}},\"prstGeom\":{\"prst\":56,\"avLst\":{}},\"extLst\":{\"ext\":[]},\"solidFill\":" +
    "{\"schemeClr\":{\"val\":9,\"lumMod\":[60000],\"lumOff\":[40000]}},\"ln\":{\"solidFill\":{\"srgbClr\"" +
    ":{\"val\":[31,79,122]}},\"w\":1,\"prstDash\":0,\"cap\":2,\"round\":true},\"effectLst\":{}},\"style\":" +
    "{\"fillRef\":{\"ColorProp\":{\"colorFillType\":0,\"schemeClr\":{\"val\":-4142}},\"idx\":1},\"lnRef\":" +
    "{\"ColorProp\":{\"colorFillType\":0,\"schemeClr\":{\"val\":-4142}},\"idx\":2},\"fontRef\":{" +
    "\"TextCharacterProperties\":{\"latin\":{\"typeface\":\"+mn-lt\"},\"sz\":14.666666666666666," +
    "\"solidFill\":{\"srgbClr\":{\"val\":[255,255,255]}}}},\"idx\":1},\"effectRef\":{\"idx\":0,\"ColorProp\":{"
```

```
+
    "\"colorFillType\":0,\"schemeClr\":{\"val\":4}}}},\"txBody\":{\"p\":[{\"elements\":
[{\"elementType\":0,\"t\" +
    "\":\"\",\"rPr\":{\"latin\":{\"typeface\":\"Calibri\"},\"sz\":14.6667,\"b\":false,\"i\":false,\"solidFill"
+
    "\":{\"srgbClr\":{\"val\":[255,255,255]}}}}],\"pPr\":{\"defRPr\":{\"latin\":
{\"typeface\":\"Calibri\"},\"sz" +
    "\":14.6667,\"b\":false,\"i\":false,\"solidFill\":{\"srgbClr\":{\"val\":[255,255,255]}}},\"algn\":0}," +
    "\"endParaRPr\":{}}],\"bodyPr\":{\"anchor\":0,\"horzOverflow\":1,\"vertOverflow\":2},\"lstStyle\":{}}}},"
+
    "\"name\":\"rightArrowCallout 1\",\"shapeType\":5}");


var chart = worksheet.Shapes.AddChart(ChartType.Line, 10, 10, 300, 300);
//update chart from json
chart.FromJson("{\"name\":\"Chart 1\",\"x\":145,\"y\":133,\"width\":480,\"height\":300," +
    "\"startRow\":6,\"startRowOffset\":13,\"startColumn\":2,\"startColumnOffset\":21," +
    "\"endRow\":21,\"endRowOffset\":13,\"endColumn\":10,\"endColumnOffset\":5," +
    "\"isSelected\":true,\"typeName\":\"2\",\"chartSpace\":{\"typeName\":" +
    "\"chartSpace\",\"roundedCorners\":false,\"chart\":{\"title\":{\"txPr" +
    "\":{\"p\":[{\"elements\":[{\"elementType\":0,\"t\":\"\",\"rPr\":{" +
    "\"latin\":{\"typeface\":\"+mn-lt\"},\"sz\":18.67,\"b\":false,\"solidFill" +
    "\":{\"schemeClr\":{\"val\":1,\"lumMod\":[65000],\"lumOff\":[35000]}}}}}]," +
    "\"pPr\":{\"defRPr\":{\"latin\":{\"typeface\":\"+mn-lt\"},\"sz\":18.67,\"b" +
    "\":false,\"solidFill\":{\"schemeClr\":{\"val\":1,\"lumMod\":[65000],\"lumOff" +
    "\":[35000]}}}},\"endParaRPr\":{}}],\"bodyPr\":{},\"lstStyle\":{}},\"overlay" +
    "\":false,\"spPr\":{\"noFill\":true,\"ln\":{\"noFill\":true},\"effectLst\":{}}}," +
    "\"autoTitleDeleted\":false,\"plotArea\":{\"axes\":[{\"axisType\":0,\"axId\":31410946," +
    "\"delete\":false,\"majorTickMark\":2,\"minorTickMark\":2,\"tickLblPos\":2,\"axPos\":0," +
    "\"scaling\":{\"orientation\":1},\"spPr\":{\"ln\":{\"solidFill\":{\"schemeClr\":{\"val" +
    "\":1,\"lumMod\":[15000],\"lumOff\":[85000]}}}},\"numFmt\":{\"formatCode\":\"General\"}," +
    "\"txPr\":{\"p\":[{\"elements\":[{\"elementType\":0,\"t\":\"\",\"rPr\":{\"latin\":{\"typeface" +
    "\":\"+mn-lt\"},\"sz\":12,\"b\":false,\"solidFill\":{\"schemeClr\":{\"val\":1,\"lumMod\":[65000]," +
    "\"lumOff\":[35000]}}}}],\"pPr\":{\"defRPr\":{\"latin\":{\"typeface\":\"+mn-lt\"},\"sz\":12,\"b" +
    "\":false,\"solidFill\":{\"schemeClr\":{\"val\":1,\"lumMod\":[65000],\"lumOff\":[35000]}}}}," +
    "\"endParaRPr\":{}}]},\"auto\":true,\"lblOffset\":0,\"tickMarkSkip\":1,\"noMultiLvlLbl\":true," +
    "\"AxisGroup\":0,\"AxisType\":0,\"crosses\":1,\"crossAx\":38384719},{\"axisType\":3,\"axId\":38384719," +
    "\"delete\":false,\"majorTickMark\":2,\"minorTickMark\":2,\"tickLblPos\":2,\"axPos\":1,\"scaling" +
    "\":{\"orientation\":1},\"spPr\":{\"ln\":{\"solidFill\":{\"schemeClr\":{\"val\":1,\"lumMod\":[15000]," +
    "\"lumOff\":[85000]}}}},\"numFmt\":{\"formatCode\":\"General\"},\"txPr\":{\"p\":[{\"elements\":[{" +
    "\"elementType\":0,\"t\":\"\",\"rPr\":{\"latin\":{\"typeface\":\"+mn-lt\"},\"sz\":12,\"b\":false," +
    "\"solidFill\":{\"schemeClr\":{\"val\":1,\"lumMod\":[65000],\"lumOff\":[35000]}}}}],\"pPr\":{\"defRPr" +
    "\":{\"latin\":{\"typeface\":\"+mn-lt\"},\"sz\":12,\"b\":false,\"solidFill\":{\"schemeClr\":{\"val\":1," +
    "\"lumMod\":[65000],\"lumOff\":[35000]}}}},\"endParaRPr\":{}}]},\"majorGridlines\":{\"spPr\":{\"ln\":{" +
    "\"solidFill\":{\"srgbClr\":{\"val\":[217,217,217]}},\"w\":1},\"effectLst\":{}}},\"AxisGroup\":0," +
    "\"AxisType\":1,\"crosses\":1,\"crossBetween\":0,\"crossAx\":31410946}],\"chartGroups\":[{\"chartType" +
    "\":6,\"ser\":[{\"seriesType\":0,\"idx\":0,\"order\":0,\"tx\":{\"strRef\":{\"f\":\"Sheet1!$A$2\"}}," +
    "\"cat\":{\"strRef\":{\"f\":\"Sheet1!$B$1:$D$1\"}},\"val\":{\"numRef\":{\"f\":\"Sheet1!$B$2:$D$2\"," +
    "\"numCache\":{\"formatCode\":\"General\"}}},\"shape\":2,\"invertIfNegative\":false},{\"seriesType" +
    "\":0,\"idx\":1,\"order\":1,\"tx\":{\"strRef\":{\"f\":\"Sheet1!$A$3\"}},\"cat\":{\"strRef\":{\"f\":" +
    "\"Sheet1!$B$1:$D$1\"}},\"val\":{\"numRef\":{\"f\":\"Sheet1!$B$3:$D$3\",\"numCache\":{\"formatCode" +
    "\":\"General\"}}},\"shape\":2,\"invertIfNegative\":false},{\"seriesType\":0,\"idx\":2,\"order\":2," +
    "\"tx\":{\"strRef\":{\"f\":\"Sheet1!$A$4\"}},\"cat\":{\"strRef\":{\"f\":\"Sheet1!$B$1:$D$1\"}},\"val" +
    "\":{\"numRef\":{\"f\":\"Sheet1!$B$4:$D$4\",\"numCache\":{\"formatCode\":\"General\"}}},\"shape\":2," +
    "\"invertIfNegative\":false},{\"seriesType\":0,\"idx\":3,\"order\":3,\"tx\":{\"strRef\":{\"f\":" +
    "\"Sheet1!$A$5\"}},\"cat\":{\"strRef\":{\"f\":\"Sheet1!$B$1:$D$1\"}},\"val\":{\"numRef\":{\"f\":" +
    "\"Sheet1!$B$5:$D$5\",\"numCache\":{\"formatCode\":\"General\"}}},\"shape\":2,\"invertIfNegative" +
    "\":false},{\"seriesType\":0,\"idx\":4,\"order\":4,\"tx\":{\"strRef\":{\"f\":\"Sheet1!$A$6\"}}," +
    "\"cat\":{\"strRef\":{\"f\":\"Sheet1!$B$1:$D$1\"}},\"val\":{\"numRef\":{\"f\":\"Sheet1!$B$6:$D$6\"," +
```

```
    "\"numCache\":{\"formatCode\":\"General\"}}},\"shape\":2,\"invertIfNegative\":false}],\"axId\":[31410946,"
+
    "38384719],\"barDir\":1,\"grouping\":1,\"gapWidth\":150,\"varyColors\":false,\"overlap\":-27}],\"spPr\":{"
+
    "\"noFill\":true,\"ln\":{\"noFill\":true}}},\"legend\":{\"legendPos\":4,\"spPr\":{\"noFill\":true,\"ln\":
{" +
    "\"noFill\":true}},\"txPr\":{\"p\":[{\"elements\":[{\"elementType\":0,\"t\":\"\",\"rPr\":{\"latin\":
{\"typeface" +
    "\":\"+mn-lt\"},\"sz\":12,\"b\":false,\"solidFill\":{\"schemeClr\":{\"val\":1,\"lumMod\":[65000],\"lumOff"
+
    "\":[35000]}}}}],\"pPr\":{\"defRPr\":{\"latin\":{\"typeface\":\"+mn-
lt\"},\"sz\":12,\"b\":false,\"solidFill" +
    "\":{\"schemeClr\":{\"val\":1,\"lumMod\":[65000],\"lumOff\":[35000]}}}},\"endParaRPr\":
{}}]}},\"plotVisOnly" +
    "\":true,\"dispBlanksAs\":1,\"dispNaAsBlank\":false},\"spPr\":{\"solidFill\":{\"schemeClr\":{\"val\":0}},"
+
    "\"ln\":{\"solidFill\":{\"schemeClr\":{\"val\":1,\"lumMod\":[15000],\"lumOff\":[85000]}},\"w\":1}},\"txPr"
+
    "\":{\"p\":[{\"elements\":[{\"elementType\":0,\"t\":\"\",\"rPr\":{\"latin\":{\"typeface\":\"+mn-lt\"}," +
    "\"b\":false,\"solidFill\":{\"schemeClr\":{\"val\":1,\"lumMod\":[65000],\"lumOff\":[35000]}}}}],\"pPr" +
    "\":{\"defRPr\":{\"latin\":{\"typeface\":\"+mn-lt\"},\"b\":false,\"solidFill\":{\"schemeClr\":{\"val" +
    "\":1,\"lumMod\":[65000],\"lumOff\":[35000]}}}},\"endParaRPr\":{}}]}},\"useAnimation\":false}");


var picture = worksheet.Shapes.AddPicture(null, 0, 0, 100, 100);
//update picture from json
picture.FromJson("{\"name\":\"Picture1\",\"x\":350,\"y\":10,\"width\":25,\"height\":25,\"startRow" +
    "\":0,\"startRowOffset\":10,\"startColumn\":5,\"startColumnOffset\":40,\"endRow\":6,\"endRowOffset" +
    "\":3,\"endColumn\":13,\"endColumnOffset\":22,\"isSelected\":true,\"typeName\":\"1\",\"src\":" +
    "\"" +
    "RnQU1BAACxjwv8YQUAAAAJcEhZcwAAEnQAABJ0Ad5mH3gAAAGASURBVDhPY/wPBAwUACYojRX8+/cPysINaOsCYgBWA0" +
    "4dvsyQ5FjNYM4ZBsbJTtUMp49chspCwIe3n8E0hhea0qcxrJy1jYGNgZWBk48DKPKf4funnwy/Gf4wRGR4MTAyMzCsmLq" +
    "dQVZJgmHb3VlAaSRw/ui1/5oMXv8NGAP+b1mxHyr6H8g+8F8fKGYrEP3fiifyvwVn+H8uBgOwHIoXtiw7yMDPycugZ6XO4" +
    "B3uABVlALLtGQyAYoyMjAzCkgIMlbPTGL7+Pw+WgxuwecV+htO7LjGwcrAyMELFkAEjEyPDrx+/GcxcdRn8op2golADuBg" +
    "NGdqSZjK8ef4B5CeGC8duMmxffQisAAR2rDnMcP7IdYaP3z8zeEbaQUUhAByIXirpDI/uPmMIy/RkYPzHyLBy5jYGFgZmBl5B" +
    "brCiz++/AoPwL0NwihtD4+xcsBgcgAx49/oDiIKDU4cu/U90qPpvwhr835gl+H+8XcX/4wfOQ2VRwSBNiaQACg1gYAAANcPHor" +
    "58W6sAAAAASUVORK5CYII=\",\"backColor\":\"#FFFFFF\",\"borderRadius\":3,\"borderStyle\":" +
    "\"solid\",\"borderColor\":\"#000000\",\"originalWidth\":15,\"originalHeight\":15}");

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

Please note:

- Shape, chart and picture use the same **IShape** interface for importing or exporting json string. However, it is necessary that the json information matches the caller's type. For example, if IShape is a chart, and json contains a picture, using IShape.FromJson could cause some unexpected error.
- When the shape type is a slicer or comment, the **FromJson** and **ToJson** methods of **ISlicer** and **IComment** interface should be used.

Range

Refer to the following example code which uses **IRange.FromJson** method to update a range from json string.

C#

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

IWorksheet worksheet = workbook.Worksheets[0];

worksheet.Range["B2:D4"].FromJson("{\"0\":{\"0\":{\"value\":1},\"1\":{\"value\":2}},\"1\":{\"0\":{\"value" +
    "\":\"aaa\",\"style\":{\"backColor\":\"rgb(173,216,230)\",\"font\":\"normal normal 11pt Calibri,sans-
```

```
serif" +
    "\",\"foreColor\":\"Text 1\",\"themeFont\":\"Body\",\"borderLeft\":
{\"color\":null,\"style\":0},\"borderTop" +
    "\":{\"color\":null,\"style\":0},\"borderRight\":{\"color\":null,\"style\":0},\"borderBottom\":{\"color" +
    "\":null,\"style\":0},\"borderHorizontal\":{\"color\":null,\"style\":0},\"borderVertical\":
{\"color\":null," +
    "\"style\":0},\"locked\":true,\"hAlign\":3,\"vAlign\":2,\"textIndent\":0,\"wordWrap\":false,\"shrinkToFit"
+
    "\":false,\"formatter\":\"General\",\"quotePrefix\":false}},\"1\":{\"value\":\"bbb\",\"style\":
{\"backColor\":" +
    "\"rgb(173,216,230)\",\"font\":\"normal normal 11pt Calibri,sans-serif\",\"foreColor\":\"Text
1\",\"themeFont\"" +
    ":\"Body\",\"borderLeft\":{\"color\":null,\"style\":0},\"borderTop\":
{\"color\":null,\"style\":0},\"borderRight" +
    "\":{\"color\":null,\"style\":0},\"borderBottom\":{\"color\":null,\"style\":0},\"borderHorizontal\":
{\"color\"" +
    ":null,\"style\":0},\"borderVertical\":
{\"color\":null,\"style\":0},\"locked\":true,\"hAlign\":3,\"vAlign\":2," +
    "\"textIndent\":0,\"wordWrap\":false,\"shrinkToFit\":false,\"formatter
\":\"General\",\"quotePrefix\":false}}," +
    "\"2\":{\"style\":{\"backColor\":\"rgb(173,216,230)\",\"font\":\"normal normal 11pt Calibri, sans-serif\""
+
    ",\"foreColor\":\"Text 1\",\"themeFont\":\"Body\",\"borderLeft\":{\"color\":null,\"style\":0},\"borderTop"
+
    "\":{\"color\":null,\"style\":0},\"borderRight\":{\"color\":null,\"style\":0},\"borderBottom\":{\"color" +
    "\":null,\"style\":0},\"borderHorizontal\":{\"color\":null,\"style\":0},\"borderVertical\":{\"color" +
    "\":null,\"style\":0},\"locked\":true,\"hAlign\":3,\"vAlign\":2,\"textIndent\":0,\"wordWrap\":false," +
    "\"shrinkToFit\":false,\"formatter\":\"General\",\"quotePrefix\":false}}},\"2\":{\"0\":{\"style\":{" +
    "\"backColor\":\"rgb(173,216,230)\",\"font\":\"normal normal 11pt Calibri,sans-serif\",\"foreColor\":" +
    "\"Text 1\",\"themeFont\":\"Body\",\"borderLeft\":{\"color\":null,\"style\":0},\"borderTop\":
{\"color\":null," +
    "\"style\":0},\"borderRight\":{\"color\":null,\"style\":0},\"borderBottom \":
{\"color\":null,\"style\":0}," +
    "\"borderHorizontal\":{\"color\":null,\"style\":0},\"borderVertical\":{\"color\":null,\"style\":0},
\"locked" +

"\":true,\"hAlign\":3,\"vAlign\":2,\"textIndent\":0,\"wordWrap\":false,\"shrinkToFit\":false,\"formatter\":" +
    "\"General\",\"quotePrefix\":false}},\"1\":{\"style\":
{\"backColor\":\"rgb(173,216,230)\",\"font\":\"normal" +
    " normal 11pt Calibri,sans-serif\",\"foreColor\":\"Text 1\",\"themeFont\":\"Body\",\"borderLeft\":
{\"color\"" +
    ":null,\"style\":0},\"borderTop\":{\"color\":null,\"style \":0},\"borderRight\":{\"color\":null,\"style" +
    "\":0},\"borderBottom\":{\"color\":null,\"style\":0},\"borderHorizontal\":{\"color\":null,\"style\":0}," +
    "\"borderVertical\":
{\"color\":null,\"style\":0},\"locked\":true,\"hAlign\":3,\"vAlign\":2,\"textIndent\":0," +
    "\"wordWrap\":false,\"shrinkToFit\":false,\"formatter\":\"General\",\"quotePrefix\":false}},\"2\":
{\"style\":" +
    "{\"backColor\":\"rgb(173,216,230)\",\"font\":\"normal normal 11pt Calibri,sans-
serif\",\"foreColor\":\"Text 1" +
    "\",\"themeFont\":\"Body\",\"borderLeft\":{\"color\":null,\"style\":0},\"borderTop\":
{\"color\":null,\"style" +
    "\":0},\"borderRight\":{\"color\":null,\"style\":0},\"borderBottom\":{\"color\":null,\"style\":0}," +
    "\"borderHorizontal\":{\"color\":null,\"style\":0},\"borderVertical\":{\"color\":null,\"style\":0}," +
    "\"locked\":true,\"hAlign\":3,\"vAlign\":2,\"textIndent\":0,\"wordWrap\":false,\"shrinkToFit\":false," +
    "\"formatter\":\"General\",\"quotePrefix:false}}}}");

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

Please note:

- When IRange.FromJson is used, the range can only be a single area (like Range["*A1:B2*"]). Otherwise, a NotSupportedException would be thrown (when Range["*A1:B2, C3:D4*"]).
- The cell position in json is treated as a relative position when using IRange.FromJson. If range is "B2:C3", the first cell data in json would be set to "B2" regardless of the cell index in json.
- If the position of cell in json is out of the range, the data is lost.

Slicer

Refer to the following example code which uses **ISlicer.FromJson** method to update a slicer from json string.

C#

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

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" },
};

IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A1:F16"].Value = sourceData;
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F16"], true);

ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category");

ISlicer slicer1 = cache.Slicers.Add(worksheet, "cate1", "Category", 200, 200, 100, 200);
//update slicer from json
slicer1.FromJson("{\"name\":\"cate2\",\"x\":400,\"y\":100,\"width\":133.33333333333334,\"height" +
    "\":266.66666666666663,\"dynamicMove\":false,\"dynamicSize\":false,\"sourceName\":\"Product\"," +
    "\"nameInFormula\":\"Slicer_Category\",\"captionName\":\"Category\",\"columnCount\":1,\"itemHeight" +
    "\":23.666666666666668,\"showHeader\":true,\"sortState\":2,\"style\":{\"name\":\"SlicerStyleLight2\"}," +
    "\"tableName\":\"Table1\",\"columnName\":\"Category\"}");

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

Please note:

- ISlicer.FromJson method cannot be used for filtering because the filter information is not stored in slicer's json (based on SpreadJS design)
- If the slicer in json has the same name as an existing slicer, an exception is thrown.

Comments

Refer to the following example code which uses **IComment.FromJson** method to update a comment from json string.

C#

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

IWorksheet worksheet = workbook.Worksheets[0];

IComment comment = worksheet.Range["A1"].AddComment("Comment1");

//update comment from json
comment.FromJson("{\"text\":\"Comment Test\",\"location\":{\"x\":595.666666666667,\"y" +
    "\":259.666666666667},\"width\":100,\"height\":80,\"fontFamily\":\"Tahoma\",\"fontWeight" +
    "\":\"bold\",\"foreColor\":\"rgb(165,165,165)\",\"backColor\":\"rgb(255,255,225)\",\"dynamicMove" +
    "\":false,\"dynamicSize\":false,\"borderWidth\":1.33333333333333,\"borderStyle\":\"solid\"," +
    "\"borderColor\":\"rgb(0,0,0)\",\"zIndex\":0,\"rowIndex\":0,\"colIndex\":0}");

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

Defined Names

Refer to the following example code which uses **IName.FromJson** method to generate the defined names from a json string.

C#

```
var workbook = new Workbook();
var worksheet = workbook.ActiveSheet;

//generate INames from json
workbook.Names.FromJson("[{\"name\":\"Test\",\"formula\":\"100\",\"row\":0,\"col\":0},{\"name\":" +
    "\"Test2\",\"formula\":\"200\",\"row\":0,\"col\":0}]");

//IName
IName name = worksheet.Names.Add("temp", "test");
name.FromJson("{\"name\":\"Test3\",\"formula\":\"Sheet1!$H$8\",\"row\":0,\"col\":0}");

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

Page Setup

Refer to the following example code which uses **IPageSetup.FromJson** method to update page setup from json string.

C#

```
var workbook = new Workbook();
var sheet = workbook.Worksheets[0];

//update pagesetup from json
sheet.PageSetup.FromJson("{\"bestFitRows\":true,\"bestFitColumns\":true,\"showBorder" +
    "\":false,\"showColumnHeader\":33,\"showRowHeader\":17,\"headerLeft\":23,\"headerCenter" +
    "\":14,\"headerRight\":66,\"footerLeft\":22,\"footerCenter\":11,\"footerRight\":12,\"headerLeftImage" +
    "\":51,\"headerCenterImage\":23,\"headerRightImage\":12,\"footerLeftImage\":63,\"footerCenterImage" +
    "\":21,\"footerRightImage\":12,\"margin\":{\"top\":80,\"bottom\":80,\"left\":30,\"right\":30,\"header" +
    "\":20,\"footer\":40},\"paperSize\":{\"width\":850,\"height\":1100,\"kind\":1}}");

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

Protection Options

Refer to the following example code which uses **IProtectionSettings.FromJson** method to update protection settings of a worksheet from json string.

C#

```
var workbook = new Workbook();
var sheet = workbook.Worksheets[0];

//update protection settings from json
sheet.ProtectionSettings.FromJson("{\"allowSelectLockedCells\":true,\"allowSelectUnlockedCells" +
    "\":true,\"allowSort\":true,\"allowFilter\":true,\"allowResizeRows\":true,\"allowResizeColumns" +
    "\":true,\"allowEditObjects\":true,\"allowDragInsertRows\":true,\"allowDragInsertColumns\":true," +

"\"allowInsertRows\":true,\"allowInsertColumns\":true,\"allowDeleteRows\":true,\"allowDeleteColumns\":true}");

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

Data Validation

Refer to the following example code which uses **IValidation.FromJson** method to update a validation from json string.

C#

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

IWorksheet worksheet = workbook.Worksheets[0];
object[,] data = new object[,] {
    {1, 10 },
    {5, 20 }
};

worksheet.Range["A1:B2"].Value = data;

//update validation from json
worksheet.Range["A1:B2"].Validation.FromJson("{\"inputTitle\":\"tip\",\"inputMessage\"" +
    ":\"Value must be between 5 and 20.\",\"type\":1,\"condition\":{\"conType\":0,\"compareType\":1,\"item1" +
    "\":{\"conType\":1,\"compareType\":3,\"expected\":\"5\",\"integerValue\":true},\"item2\":{\"conType\":1,"
+
    "\"compareType\":5,\"expected\":\"20\",\"integerValue\":true},\"ignoreBlank\":true},\"ranges\":\"A1\"," +
    "\"highlightStyle\":\"{\\\"type\\\":0,\\\"color\\\":\\\"red\\\"}\"}");


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

Refer to the following example code which uses **IValidation.ToJson** method to export the validation to json string.

C#

```
//create a memory stream to store json
MemoryStream outputStream = new MemoryStream();

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

var worksheet = workbook.ActiveSheet;

// Create a validation
worksheet.Range["C2:E4"].Validation.Add(ValidationType.Whole, ValidationAlertStyle.Stop,
ValidationOperator.Between, 1, 8);

IValidation validation = worksheet.Range["C2:E4"].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;

//export validation to json
string json = worksheet.Range["C2:E4"].Validation.ToJson();

StreamWriter writer = new StreamWriter(outputStream);
writer.Write(json);
writer.Flush();
```

Please note:

- When IValidation.FromJson method is used, data validation in the current range is cleared first and new data validation is then applied at the current range.
- The usual usage of IValidation.FromJson method is like:
  `sheet.Range["A1:B2"].Validation.FromJson("...\"ranges\":\"C3:D4\"...");`
  where DsExcel API and json data both provide the range information. When applying data validation, the former is applied and the latter is ignored.

Conditional Formatting

Refer to the following example code which uses **IFormatConditions.FromJson** method to update conditional formats in a range from json string.

C#

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

IWorksheet worksheet = workbook.Worksheets[0];

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}
};

worksheet.Range["B:C"].ColumnWidthInPixel = 80;
worksheet.Range["A1:F7"].Value = data;

//update conditional formats from json
worksheet.Range["E2:E7"].FormatConditions.FromJson("{\"rules\":[{\"ruleType\":13,\"ranges\":[{\"row" +
    "\":1,\"rowCount\":6,\"col\":4,\"colCount\":1}],\"iconSetType\":5,\"iconCriteria\":
[{\"isGreaterThanOrEqualTo" +
    "\":true,\"iconValueType\":4,\"iconValue\":33},
{\"isGreaterThanOrEqualTo\":true,\"iconValueType\":4,\"iconValue" +
    "\":67}],\"priority\":2,\"icons\":[{\"iconSetType\":5,\"iconIndex\":0},
{\"iconSetType\":5,\"iconIndex\":1},{\"iconSetType" +
    "\":5,\"iconIndex\":2}]},{\"ruleType\":1,\"operator\":6,\"stopIfTrue\":true,\"ranges\":
[{\"row\":1,\"rowCount\":6,\"col" +
    "\":4,\"colCount\":1}],\"value1\":\"66\",\"value2\":\"70\"}]}");

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

Refer to the following example code which uses **IFormatConditions.ToJson** method to export conditional formats to json string.

C#

```
//create a memory stream to store json
MemoryStream outputStream = new MemoryStream();

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

IWorksheet worksheet = workbook.Worksheets[0];

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}
};
worksheet.Range["A1:F7"].Value = data;

//weight between 66 and 70, set its interior color to LightGreen.
IFormatCondition condition = worksheet.Range["E2:E7"].FormatConditions.Add(FormatConditionType.CellValue,
FormatConditionOperator.Between, 66, 70) as IFormatCondition;
condition.Interior.Color = System.Drawing.Color.LightGreen;

//icon set rule.
IIconSetCondition condition2 = worksheet.Range["E2:E7"].FormatConditions.AddIconSetCondition();
condition2.IconSet = workbook.IconSets[IconSetType.Icon3Symbols];
condition2.IconCriteria[1].Operator = FormatConditionOperator.GreaterEqual;
condition2.IconCriteria[1].Value = 30;
condition2.IconCriteria[1].Type = ConditionValueTypes.Percent;
condition2.IconCriteria[2].Operator = FormatConditionOperator.GreaterEqual;
condition2.IconCriteria[2].Value = 70;
condition2.IconCriteria[2].Type = ConditionValueTypes.Percent;

//export conditional formats to json
string json = worksheet.Range["E2:E7"].FormatConditions.ToJson();

StreamWriter writer = new StreamWriter(outputStream);
writer.Write(json);
writer.Flush();
```

Refer to the following example code which uses **ITop10.FromJson** method to update top 10 conditional format from json string.

C#

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

IWorksheet worksheet = workbook.Worksheets[0];

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}
};

worksheet.Range["B:C"].ColumnWidthInPixel = 80;
worksheet.Range["A1:F7"].Value = data;

//update top 10 rule from json
ITop10 top10 = worksheet.Range["F2:F7"].FormatConditions.AddTop10();
top10.FromJson("{\"ruleType\":5,\"style\":{\"backColor\":\"Accent 5\",\"hAlign\":3,\"vAlign" +
    "\":0,\"locked\":true,\"textIndent\":null,\"cellButtons\":null},\"type\":0,\"rank\":\"3" +
    "\",\"ranges\":[{\"row\":1,\"rowCount\":6,\"col\":5,\"colCount\":1}]}");

//save to an excel file
workbook.Save("top10fromjson.xlsx");
```

Refer to the following example code which uses **ITop10.ToJson** method to export the top 10 conditional format to json string.

C#

```
//create a memory stream to store json
MemoryStream outputStream = new MemoryStream();

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

IWorksheet worksheet = workbook.Worksheets[0];

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}
};
worksheet.Range["A1:F7"].Value = data;

ITop10 top10 = worksheet.Range["F2:F7"].FormatConditions.AddTop10();

top10.Rank = 3;
top10.NumberFormat = "0.00";
top10.Interior.Color = System.Drawing.Color.FromArgb(91, 155, 213);

//export top 10 rule to json
string json = top10.ToJson();

StreamWriter writer = new StreamWriter(outputStream);
writer.Write(json);
writer.Flush();
```

Please note:

- When IFormatConditions.FromJson is used, the format conditions in the range are cleared first and new format conditions are then applied from json string.
- When the FromJson method of IFormatCondition, ITop10, IAboveAverage, IUniqueValues, IColorScale, IDataBar and IIconSetCondition interface is used, the FormatConditionType in json must be the same type as the caller. Otherwise, an InvalidOperationException is thrown.
- DsExcel uses the caller's range to generate the new conditional formats and the range information in json is lost.

Named Style

Refer to the following example code which uses **IStyle.FromJson** method to update an existing named style from json string.

C#

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

IWorksheet worksheet = workbook.Worksheets[0];

// Create a temp style
var style = workbook.Styles.Add("test");

////update named styles from json
style.FromJson("{\"backColor\":\"#4472c4\",\"foreColor\":\"#ffffff\",\"hAlign\":3,\"vAlign" +
    "\":0,\"font\":\"italic 11pt Calibri\",\"borderLeft\":{\"color\":\"Accent 2\",\"style\":5}," +
    "\"borderTop\":{\"color\":\"Accent 2\",\"style\":5},\"borderRight\":{\"color\":\"Accent 2\"," +
    "\"style\":5},\"borderBottom\":{\"color\":\"Accent 2\",\"style\":5},\"locked\":true,\"textIndent" +
    "\":null,\"cellButtons\":[]}");

worksheet.Range["D4"].Value = "mescius";
worksheet.Range["D4"].Style = style;

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

Refer to the following example code which uses **IStyle.ToJson** method to export the named style to json string.

C#

```
//create a memory stream to store json
MemoryStream outputStream = new MemoryStream();

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

// Create a temp style
var style = workbook.Styles.Add("CustomStyle1");
style.Interior.Color = System.Drawing.Color.FromArgb(68, 114, 196);
style.Font.Color = System.Drawing.Color.White;
style.Font.Italic = true;
style.Font.Size = 18;
style.Borders.Color = System.Drawing.Color.DarkOrange;
style.Borders.LineStyle = BorderLineStyle.Medium;

//export style to json
string json = style.ToJson();

StreamWriter writer = new StreamWriter(outputStream);
writer.Write(json);
writer.Flush();
```

Sparkline

Refer to the following example code which uses **ISparkline.FromJson** method to update a sparkline from json string.

C#

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

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

```
object[,] data = new object[,]
{
    {"Number", "Date", "Customer", "Description", "Trend", "0-30 Days", "30-60 Days", "60-90 Days", ">90
Days", "Amount"},
    {"1001", new DateTime(2017, 5, 21), "Customer A", "Invoice 1001", null, 1200.15, 1916.18, 1105.23,
1806.53, null},
    {"1002", new DateTime(2017, 3, 18), "Customer B", "Invoice 1002", null, 896.23, 1005.53, 1800.56, 1150.49,
null},
    {"1003", new DateTime(2017, 6, 15), "Customer C", "Invoice 1003", null, 827.63, 1009.23, 1869.23, 1002.56,
null}
};

worksheet.Range["B2:K5"].Value = data;
worksheet.Range["B:K"].ColumnWidth = 15;

worksheet.Tables.Add(worksheet.Range["B2:K5"], true);
worksheet.Tables[0].Columns[9].DataBodyRange.Formula = "=SUM(Table1[@[0-30 Days]:[>90 Days]])";
worksheet.Range["F3:F5"].SparklineGroups.Add(SparkType.Line, "G3:J5");

//update sparkline from json
worksheet.Range["F3"].SparklineGroups[0][0].FromJson("{\"row\":2,\"col\":5,\"orientation\":1," +
    "\"data\":{\"row\":2,\"col\":6,\"rowCount\":1,\"colCount\":5}}");

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

Refer to the following example code which uses **ISparkline.ToJson** method to export a sparkline to json string.

C#

```
//create a memory stream to store json
MemoryStream outputStream = new MemoryStream();

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

object[,] data = new object[,]
{
    {"Number", "Date", "Customer", "Description", "Trend", "0-30 Days", "30-60 Days", "60-90 Days", ">90
Days", "Amount"},
    {"1001", new DateTime(2017, 5, 21), "Customer A", "Invoice 1001", null, 1200.15, 1916.18, 1105.23,
1806.53, null},
    {"1002", new DateTime(2017, 3, 18), "Customer B", "Invoice 1002", null, 896.23, 1005.53, 1800.56, 1150.49,
null},
    {"1003", new DateTime(2017, 6, 15), "Customer C", "Invoice 1003", null, 827.63, 1009.23, 1869.23, 1002.56,
null}
};

IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["B2:K5"].Value = data;
worksheet.Range["B:K"].ColumnWidth = 15;

worksheet.Tables.Add(worksheet.Range["B2:K5"], true);
worksheet.Tables[0].Columns[9].DataBodyRange.Formula = "=SUM(Table1[@[0-30 Days]:[>90 Days]])";
worksheet.Range["F3:F5"].SparklineGroups.Add(SparkType.Line, "G3:J5");

//export sparkline to json
string json = worksheet.Range["F3:F5"].SparklineGroups[0].ToJson();

StreamWriter writer = new StreamWriter(outputStream);
```

```
writer.Write(json);
writer.Flush();
```

Please note:

- SpreadJS has two kinds of sparkline, one is consistent with Excel, and the other is extended by SpreadJS. DsExcel supports the former's ToJson and FromJson. The latter can be set through formula.
- If you want to use ToJson and FromJson methods of sparklineGroup and sparkline, there must exist a sparklineGroup or sparkline in the current range, otherwise an out-of-bounds array exception is thrown.
- Location range information applies the same rules as data validation.
- Data range is updated by the data range from json string.
- DsExcel uses sparkline from json data as much as possible, but if the data size exceeds the size of selected range, it is discarded.

Table

Refer to the following example code which uses **ITables.FromJson** method to generate a table from json string.

C#

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

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

worksheet.Range["A1:F7"].Value = data;
worksheet.Range["B:C"].ColumnWidth = 10;
worksheet.Range["D:D"].ColumnWidth = 11;

//generate tables from json
worksheet.Tables.FromJson("[{\"name\":\"Table1\",\"row\":0,\"col\":0,\"rowCount\":7,\"colCount" +
    "\":6,\"style\":{\"buildInName\":\"Medium2\"},\"rowFilter\":{\"range\":{\"row\":1,\"rowCount" +
    "\":6,\"col\":0,\"colCount\":6},\"typeName\":\"HideRowFilter\",\"dialogVisibleInfo\":{}," +
    "\"filterButtonVisibleInfo\":{\"0\":true,\"1\":true,\"2\":true,\"3\":true,\"4\":true,\"5" +
    "\":true},\"showFilterButton\":true},\"columns\":[{\"id\":1,\"name\":\"Name\"},{\"id\":2," +
    "\"name\":\"City\"},{\"id\":3,\"name\":\"Birthday\"},{\"id\":4,\"name\":\"Eye color\"},{\"id" +
    "\":5,\"name\":\"Weight\"},{\"id\":6,\"name\":\"Height\"}]}]");

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

Refer to the following example code which uses **ITables.ToJson** method to export a table to json string.

C#

```
//create a memory stream to store json
MemoryStream outputStream = new MemoryStream();

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

var worksheet = workbook.ActiveSheet;

// Create table
```

```
worksheet.Tables.Add(worksheet.Range["A1:F7"], true);
worksheet.Tables[0].Columns[0].Name = "Name";
worksheet.Tables[0].Columns[1].Name = "City";
worksheet.Tables[0].Columns[2].Name = "Birthday";
worksheet.Tables[0].Columns[3].Name = "Eye color";
worksheet.Tables[0].Columns[4].Name = "Weight";
worksheet.Tables[0].Columns[5].Name = "Height";

//export table to json
string json = worksheet.Tables.ToJson();

StreamWriter writer = new StreamWriter(outputStream);
writer.Write(json);
writer.Flush();
```

Please note:

- When ITables.FromJson and ITable.FromJson are used, the table(s) is cleared first to apply new table(s) from json string.
- When ITables.FromJson and ITable.FromJson are used, the value of cell is not cleared.

Import and Export from JSON without Worksheets

DsExcel allows you to convert skeleton of the workbook into a JSON stream. That is, you can export a workbook with just the worksheet without any data in them. This can be implemented by setting the **SerializationOptions.IgnoreSheets** to true and then pass it as a parameter while calling **IWorkbook.ToJson** interface method. You can also export the worksheet only to a separate JSON stream by using the **IWorksheet.ToJson(Stream *stream*)** method. Similarly, **IWorkSheet.FromJson(Stream *stream*)** lets you import the worksheets from JSON stream when required.

This feature is especially useful in optimizing the performance while loading large workbooks with many worksheets containing complex formula. Using the above-mentioned methods, you can load an empty workbook first with worksheet names and load the worksheet data when user selects a worksheet tab.

The example below demonstrates how you can export the workbook and worksheet to separate JSON streams and load them in a new workbook instance when required.

C#

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

    //The old workbook.
    var oldWorkbook = new GrapeCity.Documents.Excel.Workbook();

    var fileStream = GetResourceStream("xlsx\\12-month cash flow statement1.xlsx");
    oldWorkbook.Open(fileStream);

    using (Stream workbookStream = new FileStream("workbook.json", FileMode.Create))
    {
        //Export the skeleton of the workbook without worksheets to json stream.
        oldWorkbook.ToJson(workbookStream, new SerializationOptions { IgnoreSheets =
true });
```

```
        //Import the workbook stream.
        workbook.FromJson(workbookStream);
    }

    using (Stream worksheetStream = new FileStream("worksheet.json", FileMode.Create))
    {
        //Export the worksheet to json stream.
        oldWorkbook.ActiveSheet.ToJson(worksheetStream);

        //Import the worksheet stream.
        workbook.ActiveSheet.FromJson(worksheetStream);
    }

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

}

static Stream GetResourceStream(string resourcePath)
{
    string resource = "WorkbookToJsonWithoutSheets.Resource." +
resourcePath.Replace("\\", ".");
    var assembly = typeof(JSONWithoutSheets).GetTypeInfo().Assembly;
    return assembly.GetManifestResourceStream(resource);
}
```

Import and Export SpreadJS Files

DsExcel supports JSON, .ssjson, and .sjs I/O of SpreadJS files. You can import, modify, and export files created with the SpreadJS designer. The following topics explain the I/O of JSON, .ssjson, and .sjs files created with SpreadJS designer:

- Import and Export JSON Files
- Import and Export .sjs Files
- Support for SpreadJS Features

Import and Export JSON Files

DsExcel .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 DsExcel. You can use **FromJson** and **ToJson** methods for the same, as is also demonstrated in the sample code above.

Shapes

DsExcel .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

DsExcel 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 DsExcel for JSON I/O, HTML, image and PDF exporting. You can also download the JSON file containing cell buttons from **here**.

Cell Dropdowns

DsExcel 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**.

Form Controls

DsExcel supports JSON I/O of SpreadJS files containing form controls like button, dropdown, checkbox, etc., allowing you to import and export form controls to an ssjson file. You can also download the JSON file containing form controls from **here**.

Validation Styles

Validation styles can be used to highlight invalid data in a worksheet. DsExcel 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 DsExcel. 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 DsExcel 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. DsExcel 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. DsExcel 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 or to a JSON object, you can set several serialization or deserialization options. DsExcel API supports some of these options for workbook and worksheet JSON I/O. The below table explains the supported options in SpreadJS and DsExcel.

	SpreadJS (toJSON and fromJSON)	**DsExcel (ToJSON and FromJSON)**
Serialization	ignoreStyle ignoreFormula rowHeadersAsFrozenColumns columnHeadersAsFrozenRows	IgnoreStyle IgnoreFormula IgnoreColumnRowInfoOutOfUsedRange IgnoreRangeOutOfRowColumnCount ExportSharedFormula
Deserialization	ignoreStyle ignoreFormula frozenColumnsAsRowHeaders frozenRowsAsColumnHeaders doNotRecalculateAfterLoad	ignoreStyle ignoreFormula doNotRecalculateAfterLoad

DsExcel provides **SerializationOptions** and **DeserializationOptions** classes in API with above-mentioned supported properties.

The following example code serializes a workbook to JSON with options in DsExcel.

C#

```
// ignore style and formula when serialize workbook to json
string jsonWithOption = workbook.ToJson(new SerializationOptions() { IgnoreFormula = true,
IgnoreStyle = true });

workbook.FromJson(jsonWithOption);

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

The following example code deserializes a workbook from JSON with options in DsExcel.

C#

```
// 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");
```

You can control the size of exported JSON file by choosing whether you want to keep the style and size of rows and columns which are out of the used range. The **IgnoreColumnRowInfoOutOfUsedRange** property is provided in **SerializationOptions** class which:

- When set to true (default value), does not export the style and size of rows and columns which are out of the used range and hence, the file size is smaller.
- When set to false, exports the style and size of rows and columns which are out of the used range and hence, the file size is larger.

The following example code shows how the size of JSON file is impacted by setting the above mentioned property.

C#

```
var book = new Workbook();
IWorksheet worksheet = book.Worksheets[0];
//Add custom name style
IStyle style = book.Styles.Add("testStyle1");

style.Font.ThemeColor = ThemeColor.Accent1;
style.Font.TintAndShade = 0.8;
style.Font.Italic = true;
style.Font.Bold = true;
style.Font.Name = "LiSu";
style.Font.Size = 28;
style.Font.Strikethrough = true;
style.Font.Subscript = true;
style.Font.Superscript = false;
style.Font.Underline = UnderlineType.Double;

object[,] data = new object[,]{
    {"test", "test", "test", "test" },
    {"test", "test", "test", "test" },
    {"test", "test", "test", "test" },
    {"test", "test", "test", "test" },
    {"test", "test", "test", "test" },
};

worksheet.Range["B2:E6"].Value = data;
worksheet.Range["A:XFD"].Style = book.Styles["testStyle1"];
worksheet.Range["A:XFD"].ColumnWidth = 20;

//Export sizes/styles of only used range to json
using FileStream jsonFile = new FileStream("TestJson_true.json", FileMode.Create);
book.ToJson(jsonFile, new SerializationOptions() { IgnoreColumnRowInfoOutOfUsedRange = true }); //
Size of output file is 9KB

//Export all sizes/styles to json
using FileStream jsonFile2 = new FileStream("TestJson_false.json", FileMode.Create);
book.ToJson(jsonFile2, new SerializationOptions() { IgnoreColumnRowInfoOutOfUsedRange = false }); //
Size of output file is 793KB

//Default behavior (same as true option)
book.ToJson(new FileStream("TestJson_default.json", FileMode.Create)); // Size of output file is 9KB
```

You can also control whether to export formulas as shared formulas when exporting to a JSON file using the **ExportSharedFormula** property in SerializationOptions class. This enables you to export the formulas as shared formulas when it is set to true (the default value). However, if the value is set to false, the formulas will be exported as individual formulas.

In DsExcel v6.0.1 and higher versions, the formula is exported as a shared formula to a JSON file (or SSJSON file). Because the shared

formula is not compatible with DsExcel versions less than or equal to v5 and SpreadJS versions less than or equal to v15, you can use this option for backward compatibility and skip shared formulas in the exported JSON file.

The following example code exports formulas as shared formulas:

C#

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

// Set options for iterative calculation.
workbook.Options.Formulas.EnableIterativeCalculation = true;
workbook.Options.Formulas.MaximumIterations = 20;
var worksheet = workbook.Worksheets[0];

// Set values and formulas.
worksheet.Range["B2"].Value = "Initial Cash";
worksheet.Range["C2"].Value = 10000;
worksheet.Range["B3"].Value = "Interest";
worksheet.Range["C3"].Value = 0.0125;

worksheet.Range["B5"].Value = "Month";
worksheet.Range["C5"].Value = "Total Cash";

worksheet.Range["B6:B17"].Value = new double[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

worksheet.Range["C6"].Formula = "=C2*(1+$C$3)";
worksheet.Range["C7:C17"].Formula = "=C6*(1+$C$3)";

// Initialize SerializationOptions and set ExportSharedFormula to true.
SerializationOptions options = new SerializationOptions();
options.ExportSharedFormula = true;

// Save the JSON file.
workbook.ToJson(File.Create("ExportSharedFormulas.json"), options);

// Initialize SerializationOptions and set ExportSharedFormula to false.
SerializationOptions options2 = new SerializationOptions();
options2.ExportSharedFormula = false;

// Save the JSON file.
workbook.ToJson(File.Create("ExportIndividualFormulas.json"), options2);
```

Note: SpreadJS supports multi-level row or column headers but DsExcel does not. However, you can still retain the header information in DsExcel 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 DsExcel to load the JSON file.
2. Manipulate the frozen area in DsExcel.
3. Use DsExcel 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

DsExcel 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**.

DsExcel 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 DsExcel.

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 DsExcel.

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

DsExcel 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, DsExcel also provides **CellPadding** and **Margin** class, **ILabelOptions** interface, **LabelAlignment** and **LabelVisibility** enumerations to support cell padding and labels in DsExcel.

The following example code adds cell padding and labels in a DsExcel worksheet.

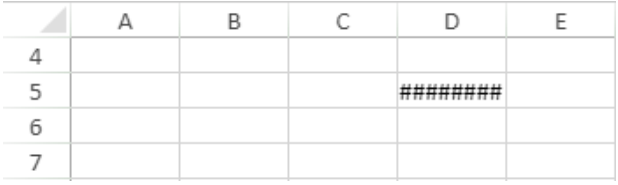
C#

```
// 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="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");
```

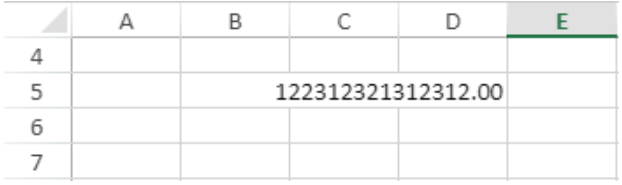
Numbers Fit Mode

In MS Excel, when a number or date does not fit in the available cell width, it masks the cell value and displays "####" in the cell. To overcome this, DsExcel provides **NumbersFitMode** enumeration so that you can choose to either mask or show entire number or date value when cell is not wide enough to accommodate the entire value. The enumeration can be accessed through **IWorkbookView.NumbersFitMode** property and can have "Mask" or "Overflow" values. To avoid displaying "####", you can set the property to "Overflow" so that overflowing value occupies the space of blank neighboring cell. No overflow happens and only partial value is displayed in case the cell itself or the neighboring cell is a merged cell or has value in it.

NumbersFitMode = Mask



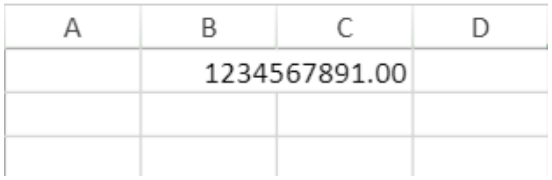
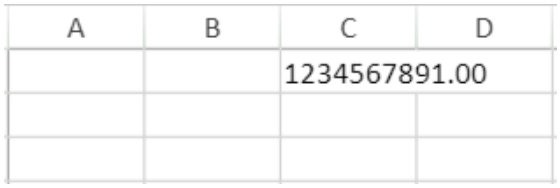
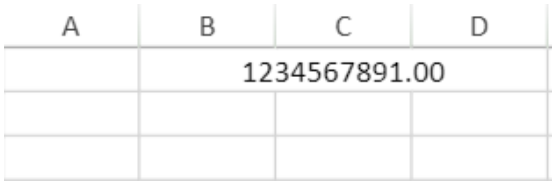
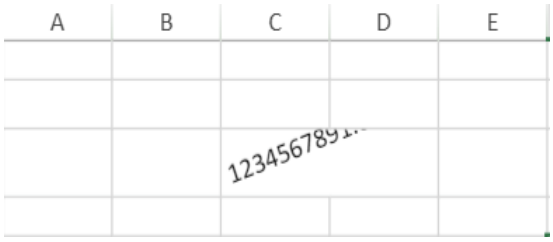
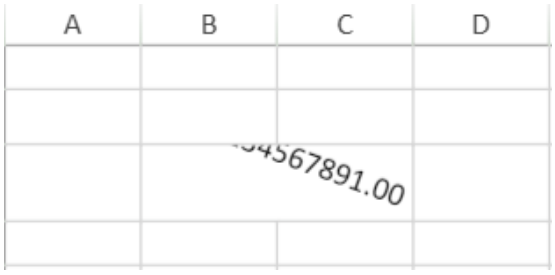
NumbersFitMode = Overflow



C#

```
// Set numbersFitMode is overflow.
workbook.BookView.NumbersFitMode = GrapeCity.Documents.Excel.NumbersFitMode.Overflow;
```

This overflow behavior and direction vary according to the horizontal alignment and orientation of the cell values. The following table displays a value longer than the available width and its overflow behavior with different horizontal alignment and orientation.

Horizontal Alignment/Orientation	Overflow Behavior
General or right alignment	
Left alignment	
Center Alignment	
Orientation greater than zero	
Orientation less than zero	

Note: As MS Excel does not support the NumbersFitMode, IWorkbookView.NumbersFitMode = Overflow is not effective on exporting the worksheet to MS Excel.

Background Image

DsExcel 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**.

DsExcel also provides **BackgroundPictures** property in IWorksheet interface to add background pictures in DsExcel. For more information, refer Support Sheet Background Image.

The following example code sets background image in DsExcel 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

DsExcel 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**.

DsExcel also provides **BackColor** and **GrayAreaBackColor** properties in IWorkbookView interface to set background color in DsExcel.

The following code example sets background color for all the worksheets in DsExcel.

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.

Row and Column Count

DsExcel allows you to set the count of rows and columns in a worksheet while performing JSON I/O. The **RowCount** and **ColumnCount** properties of the **IWorksheet** interface can be used to achieve the same. You can also use the **IgnoreRangeOutOfRowColumnCount** property of **SerializationOptions** class to choose whether to export the data outside the range of specified row and column count or not. The default value of this property is false which exports the data outside the range of specified row and column count to JSON.

Refer to the following example code which sets the row and column count in a worksheet and exports it to a JSON file.

C#

```
IWorkbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
worksheet.Range["A1"].Value = 1;
worksheet.Range["A11"].Value = 2;

// Modify the row count and column count of the worksheet.
worksheet.RowCount = 10;
worksheet.ColumnCount = 10;

// Save to a json file.
// Open this json file with spreadjs, you will find that the row count is 10, and the column count is 10.
FileStream file = new FileStream("RowColumnCount.json", FileMode.Create);
workbook.ToJson(file, new SerializationOptions { IgnoreRangeOutOfRowColumnCount = true});
```

Limitation

The row and column count setting is only supported for JSON I/O and cannot be exported to Excel or PDF file.

Set Tab Strip Position

DsExcel allows you to set various properties of Tab Strip like its position, width, display new tab button, editing of worksheet name etc. while performing JSON I/O. The IWorkbook interface provides properties like TabNavigationVisible, NewTabVisible, AllowSheetReorder, TabStripWidth, TabStripPosition etc.

Refer to the following example code which sets the position of tab strip to left and other tab strip properties.

C#

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

workbook.BookView.AllowSheetReorder = false;
workbook.BookView.TabEditable = false;
workbook.BookView.TabNavigationVisible = false;
workbook.BookView.TabStripPosition = SpreadJSTabStripPosition.Left;
workbook.BookView.TabStripWidth = 150;
workbook.BookView.NewTabVisible = false;
using var file = new FileStream("sheettabposition.json", FileMode.Create);
workbook.ToJson(file);
```

Set Size of Check Box, Check Box List and Radio Box List Cells

DsExcel supports setting the size of Check Box, Check Box List and Radio Box List Cells while performing JSON I/O. The **BoxSize** and **AutoBoxSize** properties are provided in the **CheckBoxCellType**, **CheckBoxListCellType** and **RadioButtonListCellType** classes. The BoxSize property can be used to set the size of cell whereas the AutoBoxSize property can be used to enable whether the box size should change with font size.

Refer to the following example code which sets the box size and AutoBoxSize property to true for Check Box List cell.

C#

```
//create a new workbook
var workbook = new 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,
    BoxSize = 40,
    AutoBoxSize = true
};
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:C3"].ColumnWidth = 25;
worksheet.Range["A1:C3"].CellType = cellType;
worksheet.Range["A1:C3"].Value = new object[,]
{
{new object[]{"1", "3", "5" } }
};
using var file = new FileStream("checkboxlistsize.json", FileMode.Create);
workbook.ToJson(file);
}
```

Get Picture URL

DsExcel allows you to get the URL of a picture from a json file using **url** property in the **IPictureFormat** interface. This URL is then converted to a byte array and set to the picture by using **Fill** property of the **IPictureFormat** interface. This allows you to export the json file containing picture URL to an Excel or a PDF file.

Refer to the following example code which gets the URL of a picture from JSON file and exports it to Excel and PDF formats.

C#

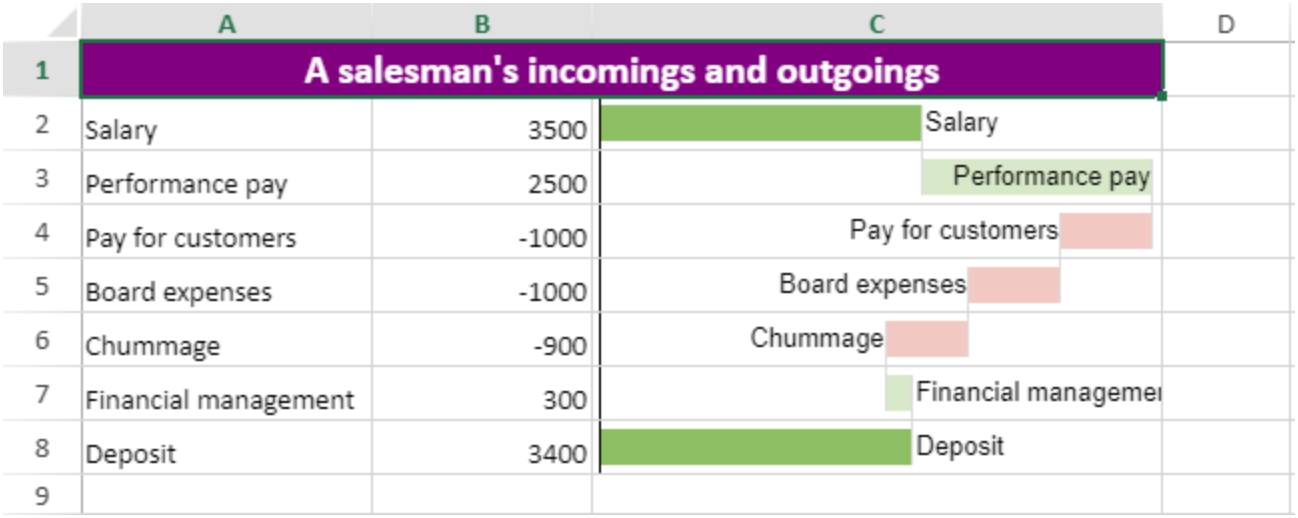
```
static void Main(string[] args)
{
    Workbook workbook = new Workbook();
    workbook.Open("Picture.json");
    var pic = workbook.ActiveSheet.Shapes[0];
    //Get URL of picture from json file
    string url = pic.PictureFormat.Url;
    byte[] picByte = GetPictureFromUrl(url);
    //Set byte array of picture
    pic.PictureFormat.Fill = picByte;
    //Save to PDF and Excel
    workbook.Save("PicturePDF.pdf");
    workbook.Save("PictureExcel.xlsx");
}

private static byte[] GetPictureFromUrl(string url)
{
    WebClient wc = new WebClient();
    byte[] imageBytes = wc.DownloadData(url);
    return imageBytes;
}
```

SpreadJS Sparklines

SpreadJS supports cascade sparkline in addition to the standard sparklines supported by MS Excel. DsExcel.Net supports export of SpreadJS files containing cascade sparklines to JSON I/O, HTML, image, and PDF formats. This topic discusses about these extended sparklines and how to create them in DsExcel.NET.

Cascade Sparkline

A cascade sparkline is generally used to analyze a value over time like yearly sales, total profit, net tax etc. It is used widely in finance, sales, legal and construction sectors, to name a few. For example, you can use cascade sparkline to compare expenses and earnings of a salesman.



DsExcel .NET provides CASCADESPARKLINE formula for creating cascade sparkline.

Syntax

= CASCADESPARKLINE(pointsRange, pointIndex, labelsRange, minimum, maximum, colorPositive, colorNegative, vertical)

Parameters

Parameter Name	Description
pointsRange(Required)	A reference that represents the range of cells that contains values, such as "B2:B8".
pointIndex (Required)	A number or reference that represents the points index. The pointIndex is >= 1 such as 1 or "D2".
LabelsRange (Optional)	A reference that represents the range of cells that contains the labels, such as "A2:A8". The default value is no label.
Minimum (Optional)	A number or reference that represents the minimum values of the display area. The default value is the minimum of the sum (the sum of the points' value), such as -2000. The minimum you set must be less than the default minimum; otherwise, the default minimum is used.
maximum (Optional)	A number or reference that represents the maximum values of the display area. The default value is the maximum of the sum (the sum of the points' value), such as 6000. The maximum you set must be greater than the default maximum; otherwise, the default maximum is used.
colorPositive(Optional)	A string that represents the color of the first or last positive sparkline's box (this point's value is positive). The default value is "#8CBF64". If the first or last box represents a positive value, the box's color is set to colorPositive. The middle positive box is set to a lighter color than colorPositive.
colorNegative (Optional)	A string that represents the color of the first or last negative sparkline's box (this point's value is negative). The default value is "#D6604D". If the first or last box represents the negative value, the box's color is set to colorNegative. The middle negative box is set to a lighter color than colorNegative.
vertical (Optional)	A boolean that represents whether the box's direction is vertical or horizontal. The default value is FALSE. You must set vertical to true or false for a group of formulas, because all the formulas represent the entire sparkline.

Parameter Name	Description
itemTypeRange (Optional)	An array or reference that represents all the item types of the data range. The values should be {"-", "+", "="} or "A1:A7" that reference the value of {"+", "-", "="}, where "+" indicates positive change, "-" indicates negative change and "=" indicates total columns.
colorTotal (Optional)	A string that either represents the color of the last sparkline's box when itemTypeRange does not exist or represents the color of the resulting sparkline's box when itemTypeRange exists.

Refer to the following example code to add cascade sparkline using formula.

C#

```
// Add cascade sparklines with horizontal bars.

for (int i = 1; i < 8; i++)
{
    worksheet.Range[i, 2].Formula = "=CASCADESPARKLINE(B2:B8, ROW() - 1, A2:A8, , , \"#8CBF64\", \"#D6604D\", FALSE)";
}
```

Note: MS Excel does not support the CASCADESPARKLINE formula, hence the formula results in **"#NAME?"** when exported to an excel file. However, on importing the file back to DsExcel, the formula displays correctly.

Import and Export .sjs Files

SpreadJS v16 introduced a new file format, .sjs, to work with large and complex files faster and generate smaller files (in size) when saved. The new .sjs format is a zipped file that contains multiple smaller JSON files and is structured similarly to the Excel XML structure.

DsExcel allows you to import and export the new .sjs file format just like the XLSX, CSV, and other file formats. You can import a .sjs file using the **Open** method of **Workbook** class. Once loaded in DsExcel, it can be exported to Excel (XLSX) or back to .sjs file using the **Save** method of Workbook class. While loading or saving a .sjs file, you can use the new option "Sjs" in **OpenFileFormat** and **SaveFileFormat** enums.

Refer to the following example code to import and export a .sjs file from the file name:

C#

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

// Open .sjs file.
workbook.Open("ProjectPlan.sjs", OpenFileFormat.Sjs);

// Save .sjs file.
workbook.Save("SaveProjectPlan.sjs", SaveFileFormat.Sjs);
```

Refer to the following example code to import and export a .sjs file from a file stream:

C#

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

// Open a .sjs file from stream.
var importStream = new FileStream("ProjectPlan.sjs", FileMode.Open);
workbook.Open(importStream, OpenFileFormat.Sjs);

// Save the .sjs file to stream.
var exportStream = new FileStream("SaveProjectPlan.sjs", FileMode.Create);
workbook.Save(exportStream, SaveFileFormat.Sjs);
```

In addition, DsExcel provides **SjsOpenOptions** and **SjsSaveOptions** classes to customize the import and export of a .sjs file. These options are especially useful in dealing with large files, such as those containing many formulas, styles, or unused names. These options are listed below:

	Class	Options	Description
Import Options	SjsOpenOptions	IncludeStyles	Indicates whether the style can be included when loading .sjs files. By default, it is true.
		IncludeFormulas	Indicates whether the formula can be included when loading .sjs files. By default, it is true.
Export Options	SjsSaveOptions	IncludeStyles	Indicates whether the style can be included when saving files. By default, the value is true.
		IncludeFormulas	Indicates whether the formula can be included when saving the file. By default, the value is true.
		IncludeUnusedNames	Indicates whether the unused custom name can be included when saving the file. By default, the value is true.
		IncludeEmptyRegionCells	Indicates whether any empty cells outside the used data range can be included while saving the file. By default, the value is true.

Refer to the following example code to import and export a .sjs file using SjsOpenOptions and SjsSaveOptions:

C#

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

// Open a .sjs file with formulas.
SjsOpenOptions openOptions = new SjsOpenOptions();
openOptions.IncludeFormulas = false;
openOptions.IncludeStyles = false;
workbook.Open("ProjectPlan.sjs", openOptions);

// Save the .sjs file with styles.
SjsSaveOptions saveOptions = new SjsSaveOptions();
saveOptions.IncludeStyles = false;
```

```
saveOptions.IncludeFormulas = true;
saveOptions.IncludeUnusedNames = false;
saveOptions.IncludeEmptyRegionCells = false;
workbook.Save("SaveProjectPlan.sjs", saveOptions);
```

DsExcel also provides **ToSjsJson** method that integrates all JSON files from the .sjs file into a single string or stream. You can also use the SjsSaveOptions with this method.

Class	Methods	Description
Workbook	ToSjsJson()	Generates a JSON string from a workbook. It integrates all JSON files from the .sjs file into a single string.
	ToSjsJson(SjsSaveOptions options)	Generates a JSON string from a workbook using save options. It integrates all JSON files from the .sjs file into a single string.
	ToSjsJson(Stream stream)	Integrates all JSON files from the .sjs file into a single string, then puts the string into the stream.
	ToSjsJson(Stream stream, SjsSaveOptions options)	Integrates all JSON files from the .sjs file into a single string using save options, then puts the string into the stream.

Refer to the following example code to export a .sjs file into a single string and save the string to a stream:

C#

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

// Open .sjs file.
workbook.Open("ProjectPlan.sjs", OpenFileFormat.Sjs);

// Generate a JSON string for .sjs file and save it to a stream.
var exportStream = new FileStream("SaveProjectPlan.json", FileMode.Create);
workbook.ToSjsJson(exportStream);
```

Support for SpreadJS Features

The following table describes the SpreadJS features supported by DsExcel either in its API or for JSON I/O, or .sjs I/O, or PDF export:

Scope	SpreadJS Features	.sjs I/O	JSON I/O	DsExcel API	PDF Export
Workbook	allowAutoCreateHyperlink	Yes	Yes	No	N/A
	allowAutoExtendFilterRange	Yes	Yes	No	N/A
	allowContextMenu	Yes	Yes	No	N/A
	allowCopyPasteExcelStyle	Yes	Yes	No	N/A
	allowDynamicArray	Yes	Yes	No	N/A

allowExtendPasteRange	Yes	Yes	No	N/A
allowInvalidFormula	Yes	Yes	No	N/A
allowSheetReorder	Yes	Yes	Yes	N/A
allowUndo	Yes	Yes	No	N/A
allowUserDeselect	Yes	Yes	No	N/A
allowUserDragDrop	Yes	Yes	No	N/A
allowUserDragFill	Yes	Yes	No	N/A
allowUserDragMerge	Yes	Yes	No	N/A
allowUserEditFormula	Yes	Yes	No	N/A
allowUserResize	Yes	Yes	No	N/A
allowUserZoom	Yes	Yes	No	N/A
allSheetsListVisible	Yes	Yes	No	N/A
autoFitType	Yes	Yes	No	N/A
backColor	Yes	Yes	Yes	Yes
backgroundImage	Yes	Yes	Yes	Yes
backgroundImageLayout	Yes	Yes	Yes	Yes
calcOnDemand	Yes	Yes	No	N/A
calculationMaximumChange	Yes	Yes	Yes	N/A
columnResizeMode	Yes	Yes	No	N/A
copyPasteHeaderOptions	Yes	Yes	No	N/A
customList	Yes	Yes	No	N/A
cutCopyIndicatorBorderColor	Yes	Yes	No	N/A
cutCopyIndicatorVisible	Yes	Yes	No	N/A
dataManager	Yes	Yes	No	N/A
defaultDragFillType	Yes	Yes	No	N/A
dynamicReferences	Yes	Yes	No	N/A
enableFormulaTextbox	Yes	Yes	No	N/A
externalReference	Yes	Yes	Yes	N/A
grayAreaBackColor	Yes	Yes	Yes	N/A
hideSelection	Yes	Yes	No	N/A
highlightInvalidData	Yes	Yes	No	Yes
iterativeCalculation	Yes	Yes	Yes	N/A

	iterativeCalculationMaximumIterations	Yes	Yes	Yes	N/A
	newTabVisible	Yes	Yes	Yes	N/A
	numbersFitMode	Yes	Yes	Yes	Yes
	pasteSkipInvisibleRange	Yes	Yes	No	N/A
	referenceStyle	No	No	No	N/A
	resizeZeroIndicator	Yes	Yes	No	N/A
	rowResizeMode	Yes	Yes	No	N/A
	saveChangesForSheet	Yes	Yes	No	N/A
	scrollbarAppearance	Yes	Yes	No	N/A
	scrollbarMaxAlign	Yes	Yes	No	N/A
	scrollbarShowMax	Yes	Yes	No	N/A
	scrollByPixel	Yes	Yes	No	N/A
	scrollIgnoreHidden	Yes	Yes	No	N/A
	scrollPixel	Yes	Yes	No	N/A
	showDragDropTip	Yes	Yes	No	N/A
	showDragFillSmartTag	Yes	Yes	No	N/A
	showDragFillTip	Yes	Yes	No	N/A
	showHorizontalScrollbar	Yes	Yes	Yes	N/A
	showResizeTip	Yes	Yes	No	N/A
	showScrollTip	Yes	Yes	No	N/A
	showVerticalScrollbar	Yes	Yes	Yes	N/A
	tabEditable	Yes	Yes	Yes	N/A
	tabNavigationVisible	Yes	Yes	Yes	N/A
	tabStripPosition	Yes	Yes	Yes	N/A
	tabStripRatio	Yes	Yes	Yes	N/A
	tabStripVisible	Yes	Yes	Yes	N/A
	tabStripWidth	Yes	Yes	Yes	N/A
	useTouchLayout	Yes	Yes	No	N/A
	autoGenerateColumns	Yes	Yes	Yes	N/A
TableSheet	-	Yes	Yes	No	N/A
GanttSheet	-	Yes	Yes	No	N/A
ReportSheet	-	Yes	Yes	No	N/A

Worksheet	colHeaderAutoText	Yes	Yes	No	N/A
	colHeaderData	Yes	Yes	No	N/A
	colHeaderRowCount	Yes	Yes	No	N/A
	columnCount	Yes	Yes	Yes	N/A
	frozenlineColor	Yes	Yes	Yes	No
	frozenTrailingColCount	Yes	Yes	Yes	N/A
	frozenTrailingColumnStickToEdge	Yes	Yes	No	N/A
	frozenTrailingRowCount	Yes	Yes	Yes	N/A
	frozenTrailingRowStickToEdge	Yes	Yes	No	N/A
	outlineColumnOptions	Yes	Yes	Yes	N/A
	rowCount	Yes	Yes	Yes	N/A
	rowHeaderAutoText	Yes	Yes	No	N/A
	rowHeaderColCount	Yes	Yes	No	N/A
	rowHeaderData	Yes	Yes	No	N/A
	showColumnOutline	Yes	Yes	Yes	Yes
	showRowOutline	Yes	Yes	Yes	Yes
	tag	Yes	Yes	Yes	N/A
	autoMergeRangeInfos	Yes	Yes	No	N/A
Table	autoGenerateColumns	Yes	Yes	Yes	N/A
	bindingPath	Yes	Yes	Yes	N/A
	expandBoundRows	Yes	Yes	Yes	N/A
	allowAutoExpand	Yes	Yes	No	N/A
Cell	RowColumnStates	Yes	Yes	Yes	N/A
	tag	Yes	Yes	Yes	N/A
	bindingPath	Yes	Yes	Yes	N/A
	altText	Yes	Yes	No	N/A
	defaultValue	Yes	Yes	Yes	Yes
Sparkline	column	Yes	Yes	Yes	Yes
	cascade	Yes	Yes	Yes	Yes
	columnstacked100	Yes	Yes	Yes	Yes
	line	Yes	Yes	Yes	Yes
Style	buttonBackColor	Yes	Yes	Yes	Yes

	hoverBackColor	Yes	Yes	No	N/A
	watermark	Yes	Yes	Yes	Yes
	Ellipsis	Yes	Yes	Yes	Yes
	Cell Buttons	Yes	Yes	Yes	Yes
	Dropdowns	Yes	Yes	Yes	Yes
	Cell Padding	Yes	Yes	Yes	Yes
	Label	Yes	Yes	Yes	Yes
	Mask	Yes	Yes	No	No
Cell Types	Button Cell Type	Yes	Yes	Yes	Yes
	CheckBoxCell Type	Yes	Yes	Yes	Yes
	Check Box List Cell Type	Yes	Yes	Yes	Yes
	Radio Button List Cell Type	Yes	Yes	Yes	Yes
	Button List Cell Type	Yes	Yes	No	Yes
	Range Template Cell Type	Yes	Yes	Yes	Yes
	Combo Box Cell Type	Yes	Yes	Yes	Yes
	Hyper Link Cell Type	Yes	Yes	Yes	Yes
Print Setting	showColumnHeader	Yes	Yes	No	No
	showRowHeader	Yes	Yes	No	No
Page Margins	bestFitRows	Yes	Yes	Yes	Yes
	bestFitColumns	Yes	Yes	Yes	Yes
	showBorder	Yes	Yes	No	N/A
	useMax	Yes	Yes	No	N/A
	pageRange	Yes	Yes	Yes	Yes
	qualityFactor	Yes	Yes	No	No
DataValidation	highlightStyle	Yes	Yes	Yes	Yes
Shapes	allowResize	Yes	Yes	Yes	No

Import and Export Macros

This section summarizes how DsExcel.NET handles the import and export of Excel files containing macros.
Using DsExcel.NET, users can load and save Excel files containing macros (.xlsm files) without any hassles. Please note that DsExcel 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 DsExcel 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. DsExcel.NET also provides various import and export options, which can be accessed from the properties present in **XlsmOpenOptions** and **XlsmSaveOptions** classes. For more information about import and export options provided by DsExcel, see Import and Export Excel Options.

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, DsExcel 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 Excel Templates

This section summarizes how DsExcel.NET handles the import and export of Excel files containing templates (.xltx file).

DsExcel.NET allows users to load and save spreadsheets containing templates without any hassles. Excel templates are pre-designed spreadsheets that help to create spreadsheets with the same layout, formatting, and formulas without having to recreate the basic elements each time, thereby saving significant amounts of time while working with spreadsheets. Now, users can load such spreadsheets in DsExcel directly as Xltx files, modify them easily and quickly, and then save them back.

DsExcel.NET also provides various import and export options, which can be accessed from the properties present

in **XltxOpenOptions** and **XltxSaveOptions** classes. For more information about import and export options provided by DsExcel, see Import and Export Excel Options.

When the **OpenFileFormat** is Xltx, templates will be imported. When the **SaveFileFormat** is Xltx, templates will be exported.

Refer to the following example code in order to import and export Xltx document from the file name.

C#

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

// Open xltx file.
workbook.Open(Path.Combine("Resources", "excel-loan-calculator.xltx"),
OpenFileFormat.Xltx);

// Save workbook as xltx file.
workbook.Save("Exported.xltx", SaveFileFormat.Xltx);
```

Refer to the following example code in order to import and export Xltx document from a file stream.

C#

```
// Create a new file stream to open a file.
using FileStream openFile = new FileStream(Path.Combine("Resources", "excel-loan-
calculator.xltx"), FileMode.OpenOrCreate, FileAccess.Read);

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

// Open xltx file.
streamworkbook.Open(openFile, OpenFileFormat.Xltx);

// Create a new file stream to save a file.
using FileStream saveFile = new FileStream("Exported-Stream.xltx",
FileMode.OpenOrCreate, FileAccess.ReadWrite);

// Save workbook as xltx file.
streamworkbook.Save(saveFile, SaveFileFormat.Xltx);
```

Import and Export OLE Objects

DsExcel .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, DsExcel 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

DsExcel allows you to convert a worksheet, any specified range, and various shape types to images using **ToImage** method. 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, SVG, and GIF.

DsExcel also provides various properties in **ImageSaveOptions** class that can be used to modify and adjust the image when exporting a worksheet, a range, or a shape to an image file through their respective interfaces using ToImage method.

Following are the properties of ImageSaveOptions class, along with the scope in which they can be used:

Properties	Worksheet	Range	Shape	Description
ScaleX and ScaleY	Yes	Yes	Yes	Gets or sets the scale of exported image file.
Resolution	Yes	Yes	Yes	Gets or sets the jpeg file's DPI in exported image file.
BackgroundColor	Yes	Yes	Yes	Gets or sets the background color of the exported image file.
ShowRowHeadings	Yes	Yes	No	Gets or sets whether to display row headings in exported image file.
ShowColumnHeadings	Yes	Yes	No	Gets or sets whether to display column headings in exported image file.

ShowGridlines	Yes	Yes	No	Gets or sets whether to display gridlines in exported image file.
GridlineColor	Yes	Yes	No	Gets or sets the gridlines color in exported image file.
ShowDrawingObjects	Yes	Yes	No	Gets or sets whether to display drawing objects (charts, shapes, or pictures) in exported image file.
BlackAndWhite	Yes	Yes	Yes	Gets or sets whether to export black and white image.

Note: To convert images with transparency, the PNG image format should be used, as the 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 an image with or without ImageSaveOptions. In the following example code, the ImageSaveOptions modify the scale, display property of row and column headings, drawing objects, and gridlines, and change the background and gridline color.

C#

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

// Add data
worksheet.Range["A1"].Value = "Sales Report";
worksheet.Range["A1"].Font.Color = Color.FromArgb(56, 93, 171);
worksheet.Range["A1"].Font.Size = 24;
worksheet.Range["A1"].Font.Bold = true;
worksheet.Range["A3:E7"].Value = new object[,]
{
{"Date","Product","Customer","Amount","Show"},
{"1/1/2021","Bose 785593-0050","Fabrikam, Inc.","$1,886.00","1"},
{"1/3/2021","Canon EOS 1500D","Alpine Ski House","$4,022.00",""},
{"1/4/2021","Haier 394L 4Star","Coho Winery","$8,144.00",""},
{"1/7/2021","IFB 6.5 Kg FullyAuto","Southridge Video","$8,002.00","1"}
};
// Instantiate ImageSaveOptions and configure the properties
ImageSaveOptions options = new ImageSaveOptions();
options.ScaleX = 3.0;
options.ScaleY = 2.0;
options.ShowRowHeadings = true;
options.ShowColumnHeadings = false;
options.ShowDrawingObjects = true;
options.BackgroundColor = Color.FromArgb(226, 231, 243);
```

```
options.ShowGridlines = true;
options.GridlineColor = Color.FromArgb(145, 167, 214);

// Save worksheet to image without ImageSaveOptions
worksheet.ToImage("WorksheetToImage.png");

// Save worksheet to image using ImageSaveOptions
worksheet.ToImage("WorksheetToImage_UsingImageSaveOptions.png", options);
```

Refer to the following example code to convert a worksheet to an image with or without ImageSaveOptions from an existing file.

C#

```
// 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];

// Instantiate ImageSaveOptions and configure the properties
ImageSaveOptions options = new ImageSaveOptions();
options.ScaleX = 3.0;
options.ScaleY = 2.0;
options.ShowRowHeadings = true;
options.ShowColumnHeadings = false;
options.ShowDrawingObjects = true;
options.BackgroundColor = Color.FromArgb(226, 231, 243);
options.ShowGridlines = true;
options.GridlineColor = Color.FromArgb(145, 167, 214);

// Create a png file stream
FileStream outputStream = new FileStream("ConvertWorksheetToImage.png",
FileMode.Create);

// Export the worksheet to image without ImageSaveOptions
worksheet.ToImage(outputStream, ImageType.PNG);

// Close the image stream
outputStream.Close();

// Create another png file stream
FileStream outputStreamoptions = new
FileStream("ConvertWorksheetToImage_UsingImageSaveOptions.png", FileMode.Create);
```

```
//Export the worksheet to image using ImageSaveOptions
worksheet.ToImage(outputStreamoptions, ImageType.PNG, options);

// Close the image stream
outputStreamoptions.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 an image with or without ImageSaveOptions. In the following example code, the ImageSaveOptions modify the scale, display property of row and column headings, drawing objects, gridlines, and change the background and gridline color.

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"];

// Instantiate ImageSaveOptions and configure the properties
ImageSaveOptions options = new ImageSaveOptions();
options.ScaleX = 3.0;
options.ScaleY = 2.0;
options.ShowRowHeadings = true;
options.ShowColumnHeadings = false;
options.ShowDrawingObjects = true;
options.BackgroundColor = Color.FromArgb(226, 231, 243);
options.ShowGridlines = true;
options.GridlineColor = Color.FromArgb(145, 167, 214);

// Save range to image without ImageSaveOptions
range.ToImage("RangeToImage.png");

// Save range to image using ImageSaveOptions
range.ToImage("RangeToImage_UsingImageSaveOptions.png", options);
```

Refer to the following example code to convert a specified range to an image with or without ImageSaveOptions from an

existing file.

C#

```
// 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];

// Instantiate ImageSaveOptions and configure the properties
ImageSaveOptions options = new ImageSaveOptions();
options.ScaleX = 3.0;
options.ScaleY = 2.0;
options.ShowRowHeadings = true;
options.ShowColumnHeadings = false;
options.ShowDrawingObjects = true;
options.BackgroundColor = Color.FromArgb(226, 231, 243);
options.ShowGridlines = true;
options.GridlineColor = Color.FromArgb(145, 167, 214);

// Create a png file stream
FileStream outputStream = new FileStream("ConvertRangeToImage.png", FileMode.Create);

// Export the range to image without ImageSaveOptions
worksheet.Range["A1:C5"].ToImage(outputStream, ImageType.PNG);

// Close the image stream
outputStream.Close();

// Create another png file stream
FileStream outputStreamoptions = new
FileStream("ConvertRangeToImage_UsingImageSaveOptions.png", FileMode.Create);

// Export the range to image using ImageSaveOptions
worksheet.Range["A1:C5"].ToImage(outputStreamoptions, ImageType.PNG, options);

// Close the image stream
outputStreamoptions.Close();
```

Convert Shape to Image

DsExcel 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 an image with or without ImageSaveOptions. In the following example code, the ImageSaveOptions modify the scale and change the background color.

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);

// Instantiate ImageSaveOptions and configure the properties
ImageSaveOptions options = new ImageSaveOptions();
options.ScaleX = 3.0;
options.ScaleY = 2.0;
options.BackgroundColor = Color.Lime;

// Save shape to image without ImageSaveOptions
shape.ToImage("ShapeToImage.png");

// Save shape to image using ImageSaveOptions
shape.ToImage("ShapeToImage_UsingImageSaveOptions.png", options);
```

Refer to the following example code to convert an autoshape to an image with or without ImageSaveOptions from an existing file.

C#

```
// 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];

// Instantiate ImageSaveOptions and configure the properties
ImageSaveOptions options = new ImageSaveOptions();
options.ScaleX = 3.0;
options.ScaleY = 2.0;
options.BackgroundColor = Color.Lime;

// Create a png file stream
FileStream outputStream = new FileStream("ConvertShapeToImage.png", FileMode.Create);

// Export the shape to image without ImageSaveOptions
worksheet.Shapes[0].ToImage(outputStream, ImageType.PNG);
```

```
// Close the image stream
outputStream.Close();

// Create another png file stream
FileStream outputStreamoptions = new FileStream("ConvertShapeToImage.png",
FileMode.Create);

// Export the shape to image using ImageSaveOptions
worksheet.Shapes[0].ToImage(outputStreamoptions, ImageType.PNG, options);

// Close the image stream
outputStreamoptions.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();
```

Note: ToImage method does not support exporting images in EMF or WMF image formats.

Import and Export Excel Options

While importing and exporting .xlsx, .xlsm, and .xltx files, it may not be required to transfer everything in exactly the same way. Sometimes you might need just data, while at other times, you might just want to skip formulas etc. DsExcel provides various import and export options which let you choose what to keep and what to skip. These options are especially useful in dealing with large files such as those containing multiple sheets, many formulas or a lot of shapes. DsExcel provides following properties to import and export the files, so that you can import or export only what is required.

	Class name	Property Name	Description
Import Options	**XlsxOpenOptions** / **XlsmOpenOptions** / **XltxOpenOptions**	**DoNotAutoFitAfterOpened**	Specify whether to automatically adjust the row height on opening an excel file.
		DoNotRecalculateAfterOpened	Specify whether to recalculate the formula values once the excel file has opened.
		Import Flags	Provides various flags to import various aspects of a worksheet. For more information, see **Work with Import Flags**.
Export Options	**XlsxSaveOptions** / **XlsmSaveOptions** / **XltxSaveOptions**	**IgnoreFormulas**	Export formula cells of DsExcel worksheet as value cells in Excel.
		ExcludeUnusedStyles	Exclude the unused styles while exporting the file.
		ExcludeUnusedNames	Exclude the unused names while

	Class name	Property Name	Description
			exporting the file.
		ExcludeEmptyRegionCells	Exclude empty cells, that is, the cells that lie outside the used range and have styles but do not contain data.

DsExcel .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.

Work with Import Flags

While opening a workbook, DsExcel .NET also provides you with several open options that can be used during the import operation.

The ImportFlags property accepts values from **ImportFlags** enumeration which allows users to import the workbook with the specified open options. These options are described in the table 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.

Refer to the following example code in order to import and export .xlsx document with import and export options.

C#

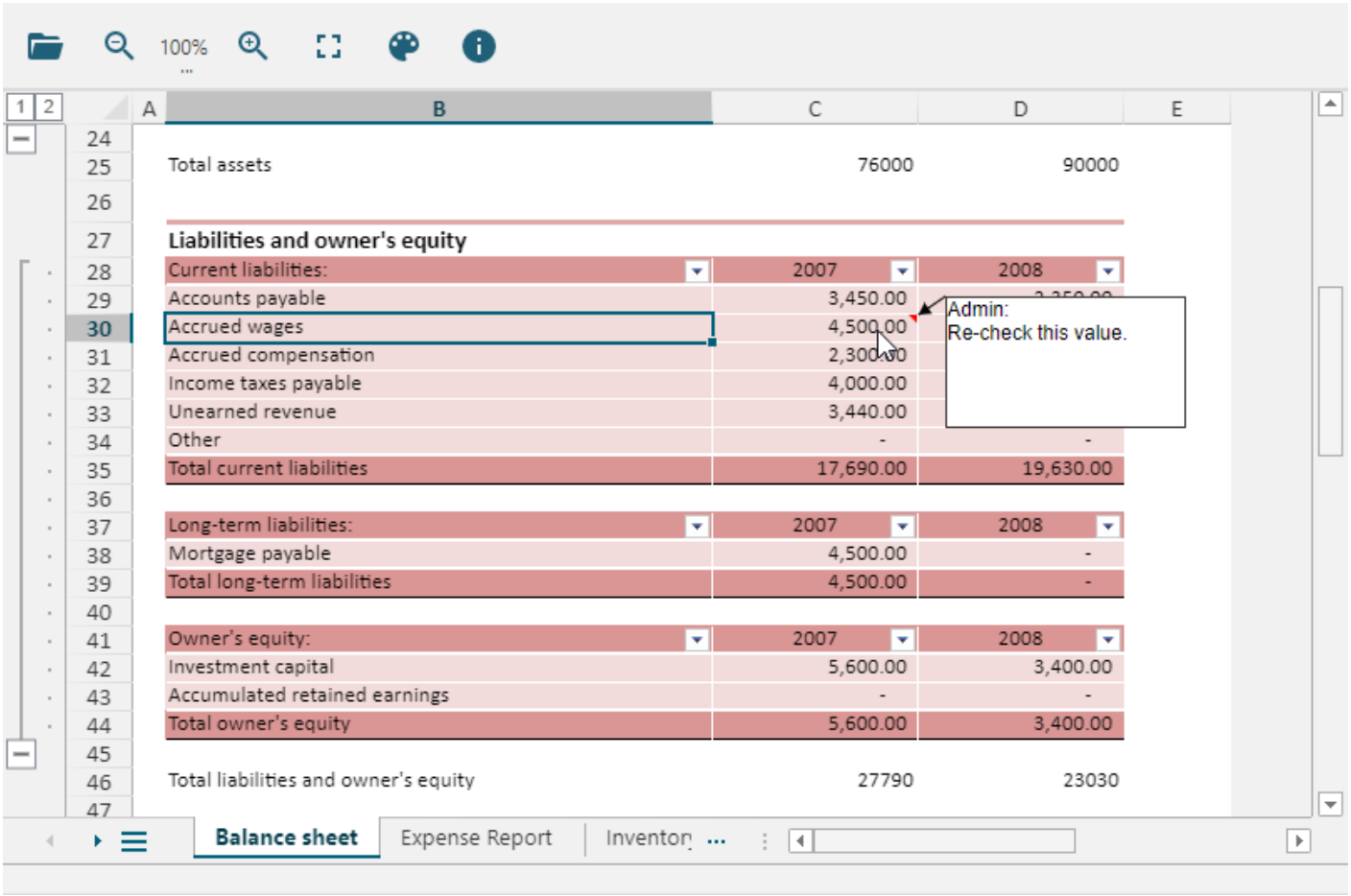
```
    // Create workbook and access its first worksheet
    Workbook workbook = new Workbook();
    IWorksheet worksheet = workbook.Worksheets[0];

    // 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
    XlsxOpenOptions options = new XlsxOpenOptions();

    // Import only data from .xlsx document.
    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);
```

Similarly, you can also import and export .xlsm and .xltx files.

Document Solutions Data Viewer

Document Solutions Data Viewer (DsDataViewer, previously GcDataViewer) is a JavaScript component which allows developers to build cross-platform web applications to load and view data documents across browsers using major JavaScript frameworks. The viewer targets to support all popular data formats and currently supports XLSX, SSJSON, and CSV formats. You can easily integrate it with DsExcel to meet your server-side and client-side needs seamlessly.

For more information about the viewer, see the Document Solutions Data Viewer documentation.



API Reference

The complete DsExcel .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
DS.Documents.Excel	Provides the namespaces for the DsExcel .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 7.1.0
- Release Notes for Version 7.0.0
- Release Notes for Version 6.2.0
- Release Notes for Version 6.1.0
- Release Notes for Version 6.0.0
- Release Notes for Version 5.2.0
- Release Notes for Version 5.1.0
- Release Notes for Version 5.0.3
- Release Notes for Version 5.0.0
- Release Notes for Version 4.2.0
- 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 :** DsExcel .NET currently does not provide support for Smart Art Graphics, exporting Excel files to XPS, and importing XLS files.

Breaking Changes

Refer to the following release notes for breaking changes:

- Version 7.0.0
- Version 6.0.0
- Version 5.1.0
- Version 5.0.3

Release Notes for Version 7.1.0

Enhancements from the Previous Release

- Enhanced template language with better performance.
- Added support for ignoring errors in a range.
- Added support for interrupting the execution of ProcessTemplate method.
- Added support for template language to support OverwriteWithFormat under Classic Mode.
- Added support for lossless .sjs and ssjson I/O of GanttSheet.
- Added support for CalculationMode options.
- Added support for searching cells with tags.
- Added support for multi-column sorting in the template language.
- Added support for custom sort orders in the template language.
- Added support to get and set cell background images.
- Added support for table references in cross-workbook formulas.
- Added support for exporting barcodes as pictures in Excel files.
- Added support for lossless .sjs and ssjson I/O of ReportSheet.

Resolved Issues

- Fixed the exception thrown on calling ProcessTemplate method when data source is not found.
- Fixed the issue of an incorrectly grouped result after calling ProcessTemplate method when using JSON data source.
- Fixed the issue of the incorrect result of SUMIFS formula after processing template file.
- Fixed the exception thrown on calling ProcessTemplate method when data field is numeric.
- Fixed the issue of an incorrect result in the resultant workbook after processing template file.
- Fixed the issue of the incorrect result of SUM formula after processing template file.
- Fixed the issue of an incorrect merge area after processing template file.
- Fixed the issue of the incorrect result of DIV formula after processing template file.
- Fixed the issue of the cell background color not being expanded as expected in the exported report workbook.
- Fixed the issue of incorrect results for the MIN and MAX formulas after processing template file.
- Fixed the issue of an incorrect calculated result in the exported report workbook after processing template file.
- Fixed the issue of the changed formula in the exported report workbook after processing template file.
- Fixed the exception thrown on calling ProcessTemplate method.
- Fixed a performance issue on processing template when setting G=R.
- Fixed the issue of the changed applied range of conditional format in the exported workbook.
- Fixed the issue of incorrect chart in the exported PDF file.
- Fixed the working of ISort.Apply() method.
- Fixed the performance issue on processing template containing merged cells.
- Fixed the issue of a lost border in the exported XLSX file.
- Fixed the issue of a lost image in the exported XLSX file when PaginationMode is true.
- Fixed the issue of a lost row or column header caption in Pivot table in the exported XLSX file.
- Fixed the exception thrown on loading a particular ssjson file.
- Fixed the exception thrown on loading a particular SJS file.
- Fixed the issue of the incorrect result of the DATEVALUE formula.
- Fixed the performance issue on processing templates when there are too many sheets.
- Fixed the issue of incorrect expansion of table formula when calling ProcessTemplate method.
- Fixed the exception thrown on loading worksheets from JSON when the JSON contains non-existent external references.

Release Notes for Version 7.0.0

Important Note

This is the initial release of the DS.Documents.Excel package. This package replaces GrapeCity.Documents.Excel, and provides the same functionality, ensures future enhancements, and is backwards compatible with GrapeCity.Documents.Excel. Existing subscriptions will continue to work with this new package.

Breaking Changes from GrapeCity.Documents.Excel 6.2.5 version

- The return value of Parent property of IDataLabel interface has been changed from IPoint to Object.

Enhancements from the Previous Release

- Added support to format trendline equations in charts (and export).
- Added support for Async User-Defined Function.
- Added support to export Excel to HTML with Inline CSS option.
- Added improvements in Grouping for OptionButton Controls.
- Added support to maintain Image aspect ratio in Template language.
- Added support for Acroform creation with DsExcel API.
- Added support for password in the protected sheet (SpreadJS Integration).
- Added support for exporting of Funnel Charts to PDF.
- Added support for Mask style (SpreadJS integration).
- Added support for IRange.DefaultValue.
- Added support for cell.altText property (SpreadJS integration).
- Added support for shapes and images in pagination mode (Template Language).
- Added support for exporting of smooth lines in charts to PDF.
- Added support for set first page number to 'Auto' in Page Setup.
- Added support to specify columns to quote on exporting to CSV.

Resolved Issues

- Fixed the disappearing formula linked with the radio button in the exported Excel file.
- Fixed the changed celltype in the exported SSJSON file.

Release Notes for Version 6.2.0

Enhancements from the Previous Release

- Added support for vertical text direction in a Shape and Chart.
- Added alignment options for Shape Text.
- Added support for double-sided printing.
- Added support for header reference.

Resolved Issue

- Fixed the incorrect result of the XLOOKUP formula.

Release Notes for Version 6.1.0

Enhancements from the Previous Release

- Added support for SpreadJS .sjs file format.
- Added export options in the ToImage() method.
- Added Copy and Move methods to copy or move multiple sheets at once.
- Added support for the XLTX file format.
- Added support Form Controls on SpreadJS JSON I/O.
- Added support for the allowResize property on SpreadJS JSON I/O.

Resolved Issues

- Fixed exception thrown on adding calculated items in Pivot Field.
- Fixed the performance of Workbook.fromJson() method is bad when JSON contains stripped style in big tables.
- Fixed the bad calculating performance when the sheet contains the SUMPRODUCT formula.
- Fixed the incorrect axis position of the chart in the exported PDF file.
- Fixed exception thrown on loading a particular Excel file with a chart.
- Fixed the calculated formula of the table column is lost in the exported SSJSON file.
- Fixed missing 3D chart in the exported PDF file.
- Fixed the incorrect result of the MATCH function.
- Fixed the incorrect result of the COUNT function.
- Fixed the incorrect overflow of cell value in the exported PDF file when the cell value type is text and has customer number format.
- Fixed the incorrect result of some functions when they refer to 3D references.
- Fixed exception thrown on exporting a PDF file when the workbook contains a combo chart whose category axis has a null value in the workbook.

Release Notes for Version 6.0.0

Breaking Changes from the previous release

- In DsExcel 5.2.5 and earlier, **XlsxOpenOption.DoNotAutofitAfterOpened** property was used to get or set whether to autofit the row height after opening the file. In DsExcel 6.0, the property name has changed to **XlsxOpenOption.DoNotAutoFitAfterOpened**, where F of Autofit has been capitalized.

Enhancements from the Previous Release

- Added a LAMBDA function to create custom and reusable function.
- Added Page-size pagination and Count-per-page pagination properties and functions for generating Paginated reports.
- Added new text and array manipulation Excel functions.
- Added support for RowColumnStates in JSON I/O to save the cell state info.
- Added shape text support that refers to with range or defined name.
- Added direct methods for refer shapes or pictures with cell/cell range.
- Added autofit options that control xlsx/xlsm file while opening.

- Added externalReference using cross-workbook formula support in JSON IO.
- Added support for GetUsedRange method to extract used range for the selected area.
- Added support for setting page number and page count formula for groups in template pagination.
- Added support for GenerateReport method that helps you keep original template and return the report workbook instance.
- Added overload of GenerateReport method that helps you process template for specific worksheets in template language.
- Added Intersect, Union, and Offset methods in IRange interface to get the intersection, union, and offset of the current range.
- Added "allSheetsListVisible" feature to support all sheets button of SpreadJS in JSON IO.

Release Notes for Version 5.2.0

Enhancements from the Previous Release

- Added new APIs to add or remove Excel form controls.
- Added CASCADESPARKLINE formula to support JSON I/O and export of Cascade Sparkline to PDF/HTML/Image.
- Added support for data tables in Charts.
- Added support for Paginated Templates in spreadsheets (with DsExcel templates).
- Added support for LET function in built-in formulas.
- Added support for spilled data in GetPivotData function.
- Added support for calculated items in Pivot Table.
- Added "CellInfo.GetAccurateRangeBoundary" method to increase accuracy by returning RectangleF object of double type.
- Added support for Json data source for data binding.
- Added IsVolatile property in CustomFunction class to support cache for formulas.
- Added support for debug mode in DsExcel templates.
- Added support to include and render SVG image.
- Added additional formula details in InvalidFormulaException for better debugging.

Release Notes for Version 5.1.0

Breaking Changes from the Previous Release

- The new interface **Workbook.ImportData(string fileName, string sourceName)** overwrites the previous **ImportData(string fileName, string worksheetName)**.
    - When source is mentioned directly in the form of worksheet name, the code remains compatible with the previous implementation.
    - After version upgrade, if the method is called along with the parameter name "worksheetName: "xxxx"", the **compilation fails** and user needs to modify the parameter name to "sourceName".

Enhancements from the Previous Release

- Added 'ShowValuesAs' option for the 'Values' field in Pivot Table.
- Added NumbersFitMode enumeration to provide an option to either mask or show the entire number or date.
- Added support for modifying password of Excel documents.
- Added support of calculated fields in Pivot Table.
- Added support for JSON data source for template language.
- Added support for showing #N/A as an empty cell in charts.
- Added ConvertToRange**()** method to convert a table into regular ranges without losing table style and data.
- Added support for Cell function.
- Added ICsvParser interface which uses Parse method to define custom rules for parsing the text.
- Added support for Pivot Table views.
- Added support for importing data functions of Table/Range/Sheet.
- Added support to include TableSheet in the list of supported SpreadJS features

Resolved Issues

- Resolved an issue of formulas while converting the imported Excel to JSON.
- Resolved a template language issue where out of memory exception was thrown while using a template in multi-thread.
- Resolved a template language issue of the layout being incorrect while the xlsx file is exported using a template.
- Resolved a template language issue of incorrect sum function when exported to the xlsx file.
- Resolved a template language issue for the cell values which are not expected in the exported Excel file.
- Resolved a template language issue where IF formula was lost in the exported xlsx file.
- Resolved a template language issue where formula was not expanded correctly in the exported xlsx file.
- Resolved a template language issue where result of the sum function was exported incorrectly.
- Resolved an issue where IndexOutOfRangeException was thrown on exporting radar chart to PDF file.
- Resolved an issue where the legend of the series was not shown in the exported PDF file.
- Resolved an issue where InvalidFormulaException is thrown on creating a table when "'" was there in the table column's name.
- Resolved an issue for exporting extra characters in data validation in an exported JSON file.
- Resolved an issue for throwing NullReferenceException on setting series formula which contains bubble size info.
- Resolved an issue of getting incorrect layout of a checkbox list in exported excel file.

Release Notes for Version 5.0.3

Breaking Changes from the Previous Release

- In DsExcel v5.0.2 and earlier, in order to create an empty Filter with no condition, user had to create a filter with condition first, and then clear the condition. With v5.0.3 onwards, **IRange.AutoFilter** and **IRange.AutoFilter(Field)** method behavior has changed. These methods will create empty filter with no condition in the worksheet. No other code needed to set empty filter condition.

Enhancements from the Previous Release

- Added **SerializationOptions.IgnoreSheets** property to export a workbook without worksheet data.
- Added **IWorksheet.ToJson(Stream *stream*)** method to export individual worksheets to a JSON stream.

Resolved Issues

- Resolved an issue where formulas were losing in exported Excel file
- Resolved an issue where ToJson method generated invalid JSON file when Excel file was having multi-line comments
- Resolved an issue where incorrect cell value was there when evaluating formula using DsExcel.
- Resolved an issue where cell style was incorrect in exported Excel file after loading the JSON exported by SpreadJS.
- Resolved an issue in which an exception was thrown on calling workbook.toJson() method.
- Resolved an issue of performance degradation when evaluating values from the particular Excel file.
- Resolved an issue where formula result of SUBSTITUTE was incorrect.

Release Notes for Version 5.0.0

Enhancements from the Previous Release

The following features have been added with this version of the product:

- Support for Linked Picture.
- Print documents directly to physical Printer.
- Support for Table expandBoundRows API.
- Support for Excel threaded comments.
- Import data function.
- Support for workbook views.
- New Formula2 property to set Dynamic Array Formula.
- Support for GETPIVOTDATA Function.

Resolved Issues

The following issues have been resolved since the last release.

- The result of NOW() function is incorrect.
- There is a large gap on the top of the cell in exported PDF file.
- NullPointerException is thrown on calling 'Workbook.Open()' method.
- The cell style is lost in the exported JSON file when compared with the original JSON file.
- The pattern fill settings of charts are not rendered in exported PDF file.
- The text with "\n" in charts is shifted in exported PDF file.
- The cell values are incorrect in exported CSV file.
- The cell styles are incorrect in exported Excel file after loading the JSON file.
- The exported Excel file is corrupted when copying a worksheet from another worksheet.
- The result of 'TEXT' function is incorrect.
- ArrayIndexOutOfBoundsException is thrown on calling 'Workbook.ToJson()' method.

Release Notes for Version 4.2.0

Enhancements from the Previous Release

The following features have been added with this version of the product:

- Dynamic Array Formulas along with the new functions:
  - FILTER
  - RANDARRAY
  - SEQUENCE
  - SINGLE
  - SORT
  - SORTBY
  - UNIQUE
- Support for new Calc Engine functions:
  - WEBSERVICE
  - FILTERXML
  - ASC
  - DBCS
  - JIS
  - XLOOKUP
  - XMATCH
- External workbook links from the web.
- Document properties for workbook.
- Retrieve the row and column grouping information.
- Copy hidden rows to new range.
- Control the size of the exported json file.
- Support for margin settings for text in a shape.
- Expand or Collapse grouped items in a Pivot Table.
- More features for SpreadJS integration: RowCount or ColumnCount, get URL of a picture, Pivot Table for json I/O, etc.

Resolved Issues

The following issues have been resolved since the last release.

- NullReference exception is thrown while adding a row in a table.
- When MarkerStyle.None is set for ISeries.MarkerStyle, the chart is not exported correctly in Excel.
- When a JSON file is opened and exported to another JSON file, the picture floating object is lost.
- Exported xlsx file does not meet the OpenXml standard if it contains a picture shape.
- Exception is thrown on saving an Excel file.
- When the pagebreak of template properties is set as true, page breaks are added before the field
- Dynamic array formula does not generate the correct value.
- Exception is thrown on opening a JSON file.
- Exception is thrown on calling the 'ToJson' method.

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 DsExcel 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.
- ROUNDUP result is different with Excel.
- DsExcel 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 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 DsExcel template functionality.
- PDF form is not displayed when 'printed' and 'hidden' settings are false in form field.
- Rows do not repeat while using DsExcel 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 DsExcel.
- 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 DsExcel.

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 DsExcel.
- 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 DsExcel 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 MSExcel.
- 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 SpreadJS has been added.
- The support for Cell types of SpreadJS has been added.
- The support for Best fit rows/columns feature of 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 DsExcel
- Row/Cols with empty date are retained after JSON(made by SJS) I/O in DsExcel
- Cell types are retained after JSON(made by SJS) I/O in DsExcel
- No longer messy code while debugging DsExcel 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.

- DsExcel .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 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:

- DsExcel .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 DsExcel. 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.
- DsExcel .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 DsExcel .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, 35-36
Access Areas in a Range, 38
Access Cells, Rows and Columns in a Range, 43-44
Add and Delete Table Columns and Rows, 357-359
Add Slicer in Pivot Table, 386-388
Add Slicer in Table, 385-386
Add Validations, 131-134
Adjust Column Width and Row Height, 538-539
API Reference, 617
Area Chart, 321-323
Auto Fit Row Height and Column Width, 59-60
Auto-Filter Table with Slicer, 390-391
Average Rule, 126-127
Axis and Other Lines, 307-310
Background Image, 227-228
Bar Chart, 323-325
Barcodes, 247-249
Box Whisker, 342-343
Breaking Changes, 618
Calculation Mode, 190-191
Cell Context, 438-441
Cell Expansion, 437-438
Cell Types, 74-76
Cell Value Rule, 125-126
Chart, 276-277
Chart Area, 280-281
Chart Sheet, 351-353
Chart Title, 278-280
Chart Types, 319-321
Charts, 471-476
Codabar, 256-259
Code128, 264-266
Code39, 259-261
Code49, 268-270

Code93, 261-263
Color Scale Rule, 127
Column Chart, 325-327
Combo Chart, 327-329
Comments, 98-101
Conditional Formatting, 125 , 441-443
Configure Chart, 278
Configure Chart Axis, 310-313
Configure Chart Series, 286-299
Configure Columns to Repeat at Left and Right, 403-404
Configure Fonts and Set Style, 512-514
Configure Page Breaks, 400-402
Configure Page Header and Footer, 397-399
Configure Page Settings, 399-400
Configure Paper Settings, 402-403
Configure Paper Source, 405-406
Configure Print Area, 403
Configure Rows to Repeat at Top and Bottom, 404-405
Configure Sheet Print Settings, 405
Configure Slicer Layout, 391-392
Contacting Sales, 24
Control Image Quality, 552
Control Pagination, 519-520
Control Position of Overlapping Shapes, 229-230
Convert Table to Range, 354
Convert to Image, 606-613
Create and Delete Chart, 277-278
Create and Delete Tables, 353-354
Create and Set Custom Named Style, 237-239
Create Excel Report using Template, 503-508
Create Pivot Table, 362-363
Create Row or Column Group, 113-114
Create Workbook, 89
Cross Workbook Formula, 191-193
Custom Form Input Types, 466-471
Custom Functions, 195-207

Customize Chart Objects, 282
Customize Shape Format and Shape Text, 211-222
Customize Worksheets, 70-71
Cut or Copy Across Sheets, 95
Cut or Copy Cell Ranges, 45-46
Cut or Copy Shape, Slicer, Chart and Picture, 46-48
Cut or Copy Slicer, 392-394
Data Bar Rule, 127-128
Data Binding, 135-139
Data Label, 314-316
Data Matrix, 272-275
Data Source Binding, 494-503
Data Table, 318-319
Data Validations, 130-131
Date Occurring Rule, 126
Default Values in Template Cells, 452-453
Defined Names, 412-415
Delete Blank Pages From Middle, 525-526
Delete Validation, 134
Digital Signatures, 139-150
Display #N/A Values, 305-306
Display Empty Cells, 305
Document Properties, 231-232
Document Solutions Data Viewer, 616
Document Solutions for Excel, .NET Edition Overview, 11
Duplicate Slicer, 394-395
Dynamic Array Formulas, 182-185
EAN-13, 252-254
EAN-8, 254-256
Enable or Disable Calculation Engine, 95-96
End User License Agreement, 25
Error Bars, 299-305
Export Barcodes, 545-546
Export Charts, 539-544
Export Custom Page Information, 528-529
Export Different Headers On Different Pages, 526-527

Export Form Controls to Form Fields, 547-549
Export Last Page Without Headers, 527-528
Export Multiple Sheets To One Page, 523-524
Export Pivot Table Styles And Format, 514-516
Export Shapes, 516-517
Export Signature Lines, 546-547
Export Slicers, 544-545
Export Specific Pages To PDF, 529-530
Export to HTML, 554-559
Export to PDF, 511-512
Export Vertical Text, 517-518
Export Worksheet to PDF, 532-533
Expression Rule, 130
Features, 26-27
File Operations, 509
Filter, 109-112
Find and Replace Data, 49-51
Fixed Layout, 449-452
Floor, 313-314
Form Controls, 239-247
Formula Functions, 158-178
Formula Parser, 151-158
Formulas, 150-151
Freeze Panes in a Worksheet, 67-69
Freeze Trailing Panes in a Worksheet, 69-70
Funnel, 349-351
Get Address of Cell Range, 44-45
Get Intersection, Union and Offset Range , 36-38
Get Row and Column Count, 51-52
Get Special Cell Ranges, 38-43
Getting Started, 14-17
Global Settings, 443-449
Group, 112-113
Group or Ungroup Shapes, 224-226
GS1-128, 266-268
Hide Rows and Columns, 52

Histogram, 343-344
Hyperlink on Shape, 222-224
Hyperlinks, 104-106
Icon Sets Rule, 129-130
Ignore Errors in Cell Range, 66-67
Image Transparency, 228-229
Import and Export .sjs Files, 597-599
Import and Export .xlsx Document, 509-511
Import and Export CSV File, 559-561
Import and Export CSV File with Delimiters, 562-564
Import and Export Excel Options, 613-615
Import and Export Excel Templates, 604-605
Import and Export from JSON string, 567-580
Import and Export from JSON without Worksheets, 580-581
Import and Export JSON Files, 581-595
Import and Export JSON Stream, 564-567
Import and Export Macros, 603-604
Import and Export OLE Objects, 605-606
Import and Export SpreadJS Files , 581
Import CSV File with Custom Parser, 561-562
Insert And Delete Cell Ranges, 52-54
Insert and Delete Rows and Columns, 54-55
Iterative Calculation, 189-190
Keep Rows Together Over Page Breaks, 524-525
Key Features, 12-13
Legends, 316-317
License Information, 20-23
Line Chart, 329-331
Linked Picture, 230-231
Localized Formulas, 193-195
Logging, 406-412
Measure Digital Width, 62-63
Merge Cells, 55
Modify Slicer with Custom Style, 389
Modify Table Layout, 361-362
Modify Table Layout for Slicer Style, 389-390

Modify Table with Custom Style, 359-361
Modify Tables, 354-356
Modify Validation, 134-135
Open and Save Workbook, 89-92
Outline Column, 117-122
Outline Subtotals, 115-117
Page Setup, 397
Paginated Templates, 480-481
Pagination Properties and Functions, 481-494
Pareto Chart, 345-346
Paste or Ignore Data in Hidden Range, 48-49
PDF Form Builder, 453-466
PDF417, 270-272
Pie Chart, 331-333
Pivot Chart, 377-380
Pivot Table, 362
Pivot Table Settings, 363-372
Pivot Table Style, 372-377
Plot Area, 281-282
Precedents and Dependents, 185-189
Print, 396-397
Protect Workbook, 92-95
QRCode, 249-252
Quick Start, 17-20
Quote Prefix, 81
Radar Chart, 339-341
Range Operations, 35
Range Template Cell, 76-81
Redistribution, 24-25
Release Notes, 618
Release Notes for Version 1.4.0, 632-633
Release Notes for Version 1.5.0.1, 632
Release Notes for Version 1.5.0.3, 631-632
Release Notes for Version 1.5.0.4, 631
Release Notes for Version 2.0.0, 631
Release Notes for Version 2.1.0, 630-631

Release Notes for Version 2.2.0, 629-630
Release Notes for Version 3.0.0, 628-629
Release Notes for Version 3.1.0, 627-628
Release Notes for Version 3.2.0, 627
Release Notes for Version 4.0.0, 626-627
Release Notes for Version 4.1.0, 625-626
Release Notes for Version 4.2.0, 624-625
Release Notes for Version 5.0.0, 624
Release Notes for Version 5.0.3, 623-624
Release Notes for Version 5.1.0, 622-623
Release Notes for Version 5.2.0, 622
Release Notes for Version 6.0.0, 621-622
Release Notes for Version 6.1.0, 621
Release Notes for Version 6.2.0, 620
Release Notes for Version 7.0.0, 619-620
Release Notes for Version 7.1.0, 618-619
Remove a Group, 114-115
Render Excel Range Inside PDF, 520-523
Rich Text, 84-88
Row or Column Group Information, 122-125
Save Multiple Workbooks to Single PDF, 530-531
Series, 282-286
Set Array Formula, 182
Set Cell Background Image for Cell Range, 64-66
Set Custom Objects to a Range, 56-58
Set Default Values for Cell Range, 63-64
Set Formula to Range, 178-179
Set Row Height and Column Width, 58-59
Set Sheet Styling, 233-237
Set Table Formula, 179-182
Set Values to a Range, 55-56
Shape Adjustment, 226-227
Shapes and Pictures, 207-211
Shrink To Fit With Text Wrap, 518-519
Size and Position of Image, 228
Slicer, 385

Slicer Style, 388-389
Sort, 106-109
Sparkline, 380-385
Sparklines, 478-480
Specialized Chart, 346-347
SpreadJS Sparklines, 595-597
Statistical Chart, 341-342
Stock Chart, 333-335
Styles, 232-233
Summary Row, 115
Sunburst, 347-348
Support Background Color Transparency, 551-552
Support Document Properties, 537-538
Support for SpreadJS Features, 599-603
Support Security Options, 535-537
Support Sheet Background Image, 549-551
Surface Chart, 335-337
Table, 353
Table Filters, 357
Table Sort, 356-357
Table Style, 359
Tables, 476-478
Tags, 81-84
Technical Support, 24
Template Configuration, 419-421
Template Fields, 421-424
Template Properties, 424-437
Templates, 416-419
Theme, 275-276
Threaded Comments, 101-104
Top Bottom Rule, 128-129
Track Export Progress, 552-554
TreeMap, 348-349
Unique Rule, 129
Upgrade to Latest Version, 23-24
Use Do Filter Operation, 395-396

Walls, 306-307

Waterfall Chart, 344-345

Work with Used Range, 60-62

Work with Worksheets, 27-34

Workbook, 88-89

Workbook Views, 96-98

Working With Page Setup, 533-535

Worksheet, 27

Worksheet Views, 71-74

XY (Scatter) Chart, 337-339