Table of Contents

Spread.Services Documentation

Thank you for choosing Spread.Services - a new high-performance spreadsheet component that comes with essential features to help users handle all the spreadsheet challenges without any hassle.

Developed for use in .Net Core and several other platforms, this component provides developers with a comprehensive API to allow them to quickly create, manipulate, convert, and share Microsoft Excel-compatible spreadsheets that can be accessed from nearly any application or platform.

What Spread.Services 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 the use of Spread.Services, the following user documentation is available:

- Developer's Guide

For product details, the following reference documentation is available:

- Assembly Reference

Developer's Guide

This guide provides a comprehensive overview of Spread.Services along with complete introduction, implementation and customization of features to assist developers in creating powerful spreadsheets using .Net Core.

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

The major inclusions of this guide are listed below:

- Get Started
- Understand the Product
- Customize User Interaction
- Manage Formulas
- Manage Data in Spread Component
- Apply Conditional Formatting
- Apply Data Validations

For complete API reference, refer to the Assembly Reference.

For a complete list of documentation, refer to Spread.Services Documentation.

Get Started

This topic describes how to get started with Spread.Services. It includes:

- Product Requirements
- Installing the Product
- Spread.Services License Information
- Apply License to Spread.Services
- End User License Agreement
- Handle Redistribution

Refer to the ReadMe on our web site for installation and licensing information.

Product Requirements

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

.NET Core

- Operating System Support
  - Ubuntu 14.04 & 16.04
  - Mac OS X 10.11+
  - Windows 7+ / Server 2012 R2+
  - Windows Nano Server TP5
  - Windows Server 2016
- .NET Core 1.0 NuGet Package

.NET Framework 4.5

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

Mono

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

Installing the Product

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

Installing the product involves the following tasks. You can either get the Spread.Services NuGet package from https://www.nuget.org or download it locally on your machine from http://spread.grapecity.com/downloads/.

If you are installing the Spread.Services package through NuGet, refer to the following tasks:

- **To find and install Spread.Services NuGet package**

If you are installing the Spread.Services package manually, refer to the following tasks:

- **To manually create NuGet package source**
- **To install Spread.Services package using command line interface**
- **To add Spread.Services package reference**

To find and install Spread.Services NuGet package

Complete the following steps to add **Spread.Services** NuGet package in your Visual Studio application.

1. In **Solution Explorer**, right-click either **Dependencies** or a project and select **Manage NuGet Packages.**
2. In the **Browse** tab, select **nuget.org** from the **Package source** dropdown.
3. In the **Browse** tab, type **Spread Services** in the search text box at the top.
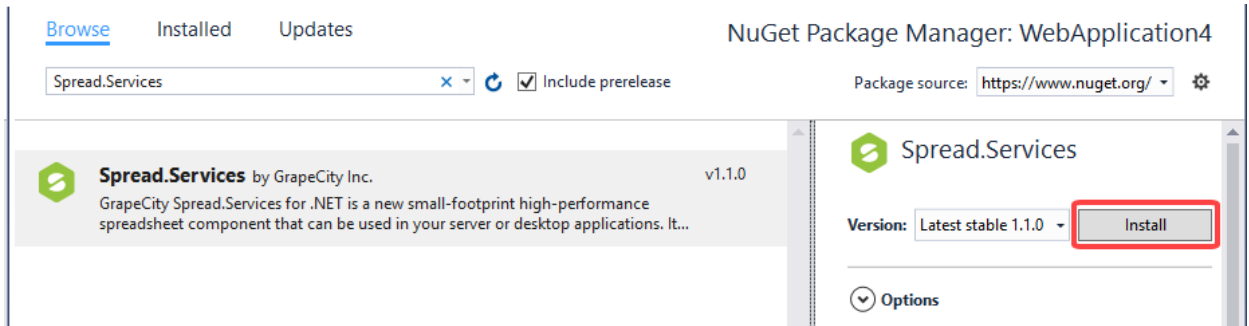
4. Click **Install** to install the Spread.Services package and its dependencies into the project. When installation is complete, make sure you check the NuGet folder in your solution explorer and confirm whether or not the Spread.Services package is added to your project dependencies.
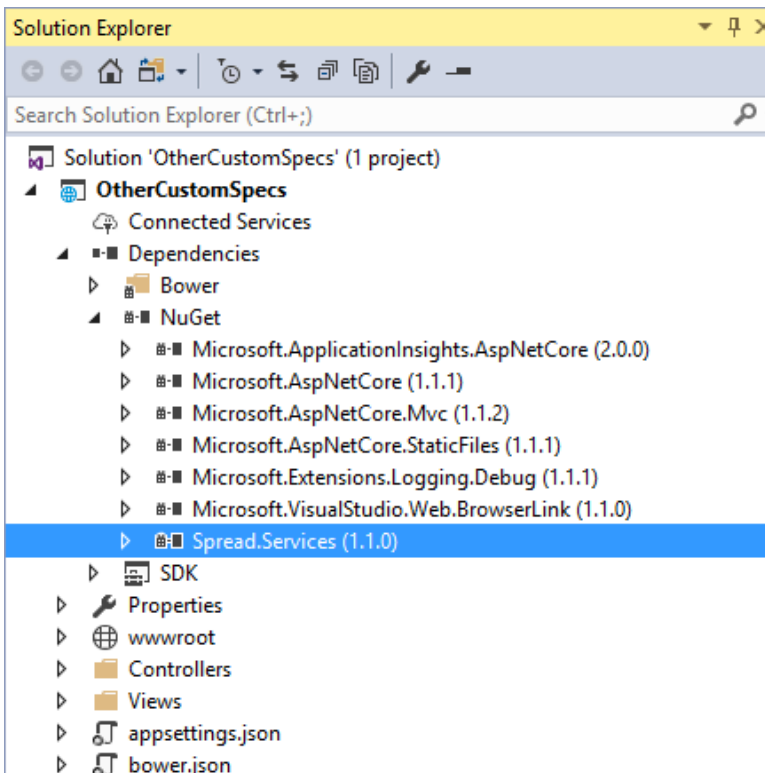
To manually create NuGet package source

In order to manually create Nuget feed source, you need to complete the following steps to add the Nuget feed URL to your Nuget settings in Visual Studio. Before you proceed with this step, make sure you first download the Spread.Services zip file using the link http://spread.grapecity.com/downloads/ and extract it.

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 ellipsis button next to the Source field to browse for the **Nupkg** folder (in the folder you extracted).
6. After you select the **Nupkg** folder, click the **Update** button and finally click OK.

To install Spread.Services package using command line interface

1. Open the CommandPrompt window on your Windows system.
2. Use the **cd** command to navigate to your project folder. For example, *cd C:\Users\Admin\Documents\Visual Studio*

2017\Projects\SpreadService.

3. Install Spread.Services NuGet package using the following command:
   dotnet add package Spread.Services

To add Spread.Services package reference

Spread.Services 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:

```
<ItemGroup>
    <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.0.0" />
    <PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2" />
    <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.1" />
    <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.1" />
    <PackageReference Include="Microsoft.VisualStudio.Web.BrowserLink" Version="1.1.0" />
    <PackageReference Include="Spread.Services" Version="1.1.0" />
</ItemGroup>
```

Spread.Services License Information

Spread.Services supports the following types of license:

- Evaluation License
- No-License
- Production License

Evaluation License

Spread.Services trial license is available for users to evaluate the product and see how it can help them with their comprehensive project requirements.

The evaluation version of the product has an expired date field that is determined when an evaluation key is generated. The evaluation version limitation is of 1 month. After users apply the evaluation license key, they can use the product before they reach the license expiry date. 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.

However, this version comes with a few limitations that restrict users to use it for unlimited period of time and perform the following crucial operations that are available only in the fully licensed version:

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.

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 a file, an Excel file with just a watermark sheet will be saved, the content of watermark will tell users that no license is found and provide our sales and contact information.

No License

No-License has a few limitations, that are highlighted below.

Maximum time of opening and saving excel files
Every time a user runs an application, he/she can open or save up-to 100 excel files using Spread.Services.

- If 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 license is not found.

- If user has saved 100 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 the user runs the program, so user 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 Spread.Services 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.

Watermark Sheet
When you save a 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.

Powered by GrapeCity Spread.Services NON-LICENSED EVALUATION VERSION. This version is not licensed to be used in a production environment. Visit http://spread.grapecity.com or email spread.sales@grapecity.com to purchase your license for distribution.

Apply License to Spread.Services

Applying license to **Spread.Services** involves completing the following steps to create and license a WorkBook:

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

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

1. In Visual Studio, select **File | New | Project** to create a new ASP.NET Core Web Application.
2. Under installed templates, select **Visual C# | Web | ASP.NET Core Web Application**, and then click **OK**.
3. In the **New ASP.NET Core Web Application(.NET Core)** dialog, select **Web Application**, and then click **OK**.
4. Add the Spread.Services 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 **Spread.Services** package, and click **Install**.
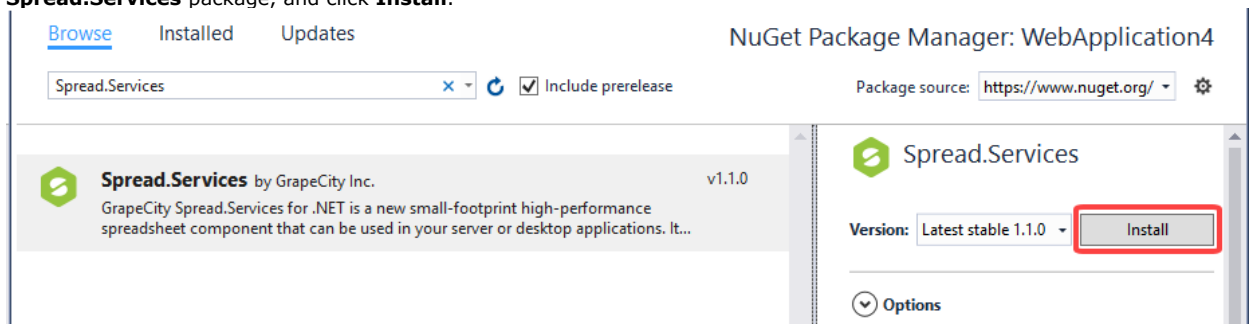
Step 2: Add a Controller

1. In the **Solution Explorer**, right click the folder **Controllers.**
2. From the context menu, select **Add | New Item.** The **Add New Item** dialog appears.
3. Complete the following steps in the **Add New Item** dialog:
   a. Expand the **Installed** tab towards left, and select **ASP.NET|MVC Controller Class**.
   b. Set name of the controller (For example: Spread_{Controller}).
   c. Click **Add.**
4. Add the product license key. For example, LicenseManager.SetLicense("Your License Key").
5. Add the following code to replace the Index() method.

SpreadController.cs

```
public IActionResult Index()
        {
                //Apply license before using the API, otherwise it will be considered as no
license.
                //Apply the license only once for an application
                LicenseManager.SetLicense("Your License Key")
                Workbook workbook = new Workbook();
                IWorksheet worksheet = workbook.Worksheets[0];
                worksheet.Range["A1:A3"].Value = 121;
                workbook.Save("Workbook.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: Spread).
3. Right click the folder Spread, and select **Add | New Item**. The **Add New Item** dialog appears.
4. Complete the following steps in the **Add New Item** dialog:
   a. Expand the **Installed** tab towards left, and select **ASP.NET|MVC View Page**.
   b. Set name of the view (for example: Index.cshtml).
   c. Click **Add**.
5. Add the following code in Index.cshtml file.

Index.cshtml

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

Step 4: Build and Run the Project

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

End User License Agreement

The GrapeCity licensing information, including the GrapeCity end-user license agreements, frequently asked licensing questions, and the GrapeCity licensing model, is available online at http://spread.grapecity.com/licensing-faqs/ and http://spread.grapecity.com/EULA/.

Handle Redistribution

In order to deploy Spread.Services, you need to make sure that you have at least one of the following frameworks

installed on your system:

- .NET Core 1.0
- .NET Framework 4.5
- Mono 4.0

In order to distribute the application, make sure you meet the installation criteria specified in the Product Requirements page in this documentation. Further, the users also needs 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	http://spread.grapecity.com/
E-mail	spread.sales@grapecity.com
Phone	(800) 858-2739 or (412) 681-4343 outside the U.S.A.
Fax	(412) 681-4384

Understand the Product

Spread.Services provides a comprehensive solution to handle all the challenges faced by users while working with bulk data in spreadsheets.

The following topics introduce this powerful product and explain how it can be used for setting up a complex spreadsheet with all the appropriate functional implementations and customizations:

- Product Overview
- Features Overview
- Namespaces Overview

Product Overview

GrapeCity Spread.Services for .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.

Spread.Sevices 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 Spread.Services 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, sorted and filtered tables, sorted and filtered pivot tables, charts, sparklines, conditional formats, and dashboard reports etc.

Features Overview

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

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

- Comments
- Chart
- Conditional Format
- Data Validation
- Formula
- Group

- [Hyperlinks](#)
- [Shape and Picture](#)
- [Style](#)
- [Sparkline](#)
- [Theme](#)
- [Table](#)
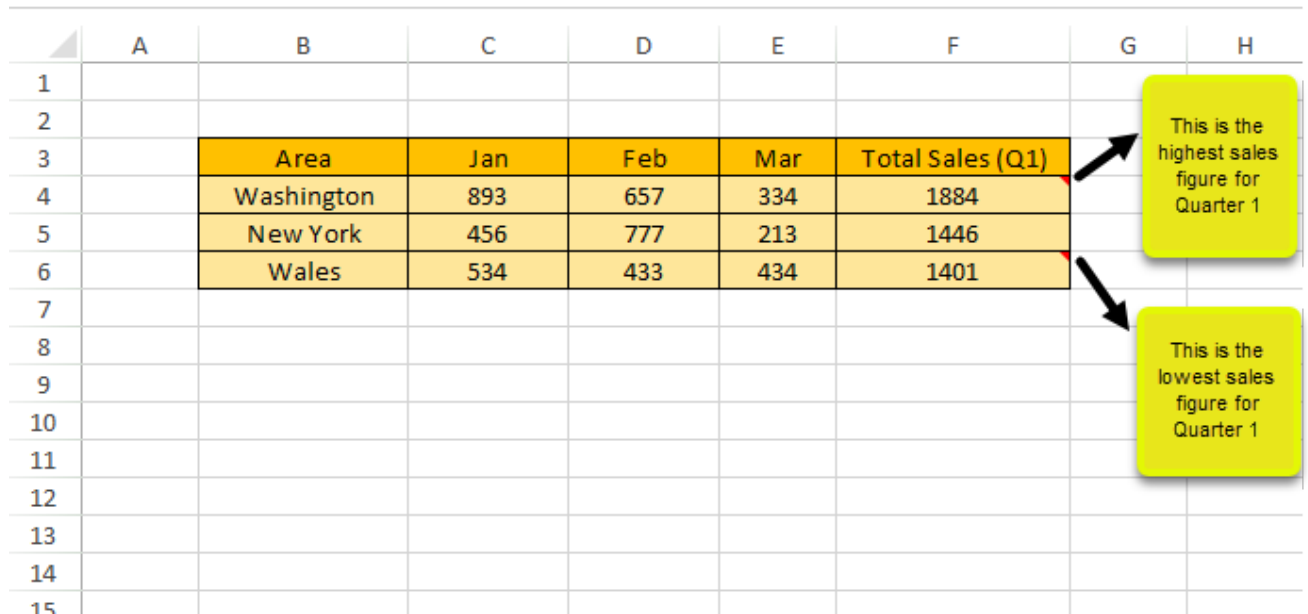
Comments

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

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

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

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

Chart

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

Annual Sales Records
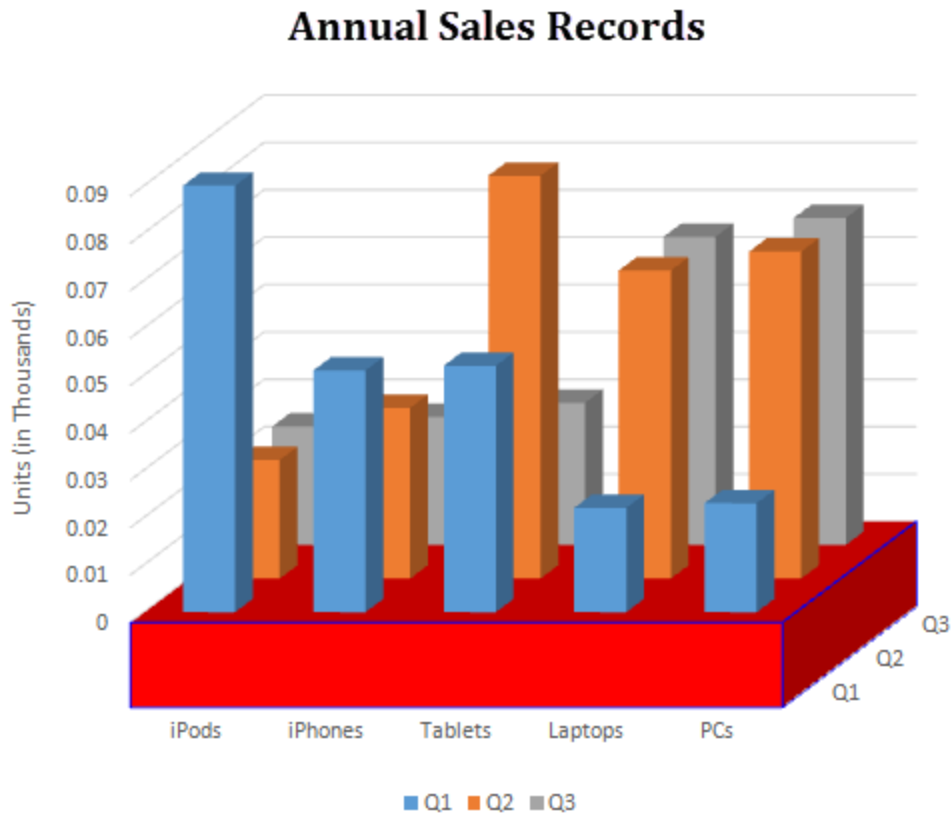
For more information on Spread.Services Chart, refer to Work with Chart.

Conditional Format

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

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

For more information, refer to Apply Conditional Formatting.

Data Validation

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

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

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

- Display error alert messages when invalid data is entered in a cell.

For more information on data validation, refer to Apply Data Validations.

Formula

Spread.Services provides you with the ability to create and use formulas - expressions to calculate the value of a cell quickly and accurately, in the spreadsheets. You can also use some built-in functions and operators to generate formulas and calculate values in cells.

For more information about formulas, refer to Manage Formulas in this documentation.

Group

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

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

To see how grouping can be used in Spread.Services to organize data in groups, refer to Apply Grouping.

Hyperlinks

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

For more information on Hyperlinks in Spread.Services, refer to Manage Hyperlinks.

Shape and Picture

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

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

Style

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

To know more about the cell styles in Spread, refer to Apply Style in this documentation.

Sparkline

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

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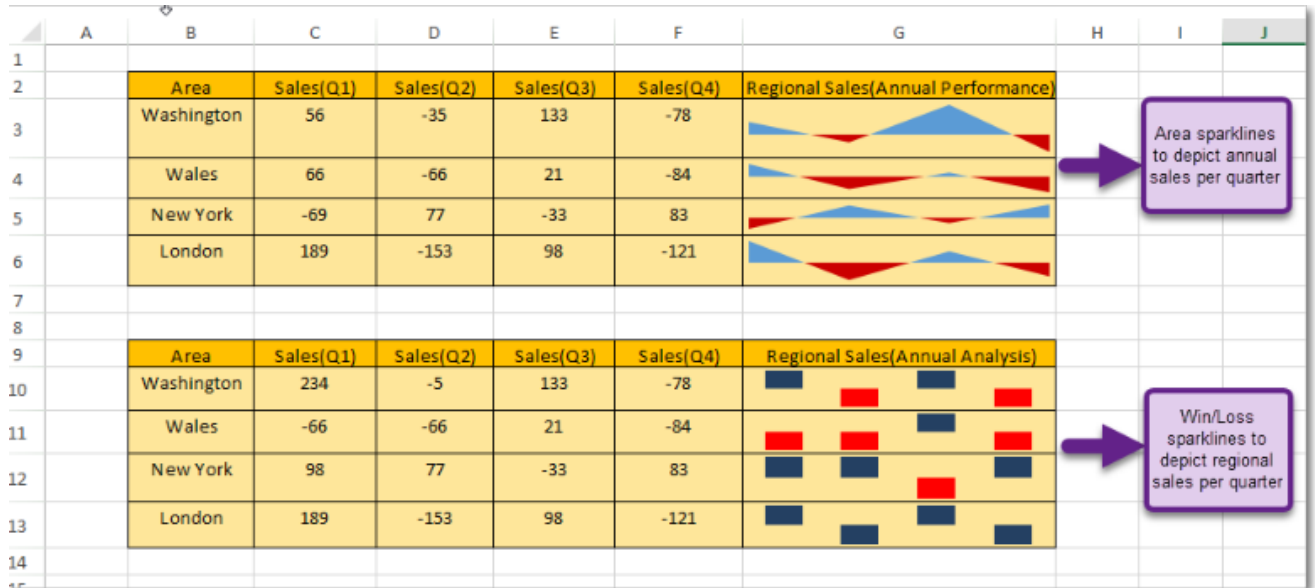
For more information on Sparklines, refer to Use Sparklines in this documentation.

Theme

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

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

For more information on Themes, please refer to Apply Theme in this documentation.

Table

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

For more information, refer to Use Table in this documentation.

Namespaces Overview

Spread.Services provides you with a list of following namespaces in the **Grapecity.Documents.Spread ('GrapeCity.Documents.Spread Assembly' in the on-line documentation)** assembly.

Namespace	Description
Grapecity.Documents.Spread ('GrapeCity.Documents.Spread Namespace' in the on-line documentation)	Provides the base classes, interfaces, enumerations, and structures for Spead.Services
Grapecity.Documents.Spread.Drawing ('GrapeCity.Documents.Spread.Drawing	Provides the interfaces and enumerations for Spead.Services drawing component.

Namespace' in the on-line documentation)

Customize User Interaction

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

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

- Manage Workbook
- Manage Worksheet
- Allow Sort
- Apply Filters
- Manage Hyperlinks
- Apply Grouping
- Apply Style
- Apply Comments

Manage Workbook

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

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

Managing a workbook involves the following tasks:

- Create Workbook
- Open, Save and Protect Workbook
- Cut or Copy Across Sheets
- Apply Theme
- Enable or Disable Calculation Engine

Create Workbook

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

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

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

C#

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

In order to add more worksheets to your workbook, refer to Work with Worksheets in this documentation.

Open, Save and Protect Workbook

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

This topic includes the following tasks:

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

Open a workbook

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

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

Refer to the following example code to open a workbook.

C#

```
// Open the existing Excel file
workbook.Open("createWorkbook.xlsx");
```

Save a workbook

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

Refer to the following example code to save your workbook.

C#

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

Protect a workbook

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

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

C#

```
// Save the Excel file and protect it using password.
workbook.Save("createWorkbook.xlsx", "123456");
```

Cut or Copy Across Sheets

In Spread.Services, 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#

```csharp
// 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#

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

Enable or Disable Calculation Engine

Spread.Services 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#

```csharp
//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#

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

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

Apply Theme

You can apply either a built-in theme or a custom theme to a workbook. The default theme of a workbook is the standard Office theme. In Spread.Services, the current theme of a workbook is represented by the **ITheme**

interface (on-line documentation).

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

Applying theme in a workbook involves the following tasks:

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

Apply built-in theme to the workbook

In order to enable you to maintain consistency in the appearance across all the worksheets in the workbook, Spread.Services 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#

```csharp
//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 **Add method (on-line documentation)** of **Themes class (on-line documentation)** to add a custom theme. For this, you first need to create an instance of the Themes class and initialize it. 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#

```csharp
Themes themes = new Themes();

//base theme is office theme when not give parameter.
ITheme theme = themes.Add("testTheme");
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;
```

Manage Worksheet

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

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

Managing a worksheet involves the following tasks:

- [Work with Worksheets](#)
- [Range Operations](#)

- [Freeze Panes in a Worksheet](#)
- [Work with Shape And Picture](#)
- [Customize Worksheets](#)

Work with Worksheets

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

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

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#

```csharp
// 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#

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

Activate a worksheet

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

Refer to the following example code to activate a worksheet.

C#

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

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

Access a worksheet

All the worksheets within a workbook are stored in **Worksheets** collection. In order to access a specific worksheet within a workbook, you can choose either of the two ways : using the **Index property (on-line documentation)**

or using the **Name property (on-line documentation)** of the **IWorksheet interface (on-line documentation)**.

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

C#

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

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

Protect a worksheet

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

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

C#

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

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

Delete Worksheet

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

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

C#

```csharp
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();
```

Range Operations

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

The operations handled using Range property are as follows:

- Insert And Delete Cell Ranges
- Cut or Copy Cell Ranges
- Customize Cell Ranges
- Merge Cells
- Get Range and Range Area
- Insert and Delete Rows and Columns
- Work with Used Range

Insert And Delete Cell Ranges

Spread.Services 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

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

Spread.Services 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#

```csharp
//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#

```csharp
//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

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

Spread.Services 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#

```csharp
//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#

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

Cut or Copy Cell Ranges

Spread.Services 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. To cut or copy data across multiple sheets, refer to Cut or Copy Across Sheets.

Copy cell range

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

Spread.Services 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 to copy the cell range in a workbook.

C#

```csharp
// 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

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

Spread.Services 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#

```csharp
// 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"]);
```

Merge Cells

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

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

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

C#

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

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

C#

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

Get Range and Range Area

Range or a range area refers to an array of cells that have been defined in a spreadsheet. While working with worksheets in a workbook, users can define multiple ranges and then further access those range areas separately to perform certain tasks like formatting of cells, merging of cells, insertion or deletion of cells within a range and other useful operations.

Refer to the following example code to see how you can define a range.

C#

```
// Defining range
var range = worksheet.Range["A5:B7,C3,H5:N6"];
```

Refer to the following example code to see how you can access the range area with a range.

C#

```
//area1 is A5:B7.
var area1 = range.Areas[0];
area1.Value = "5";
//area2 is C3.
var area2 = range.Areas[1];
area2.Value = "8";
//area3 is H5:N6.
var area3 = range.Areas[2];
area3.Value = "10";
```

Insert and Delete Rows and Columns

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

Insert rows and columns

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

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

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

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

C#

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

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

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

C#

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

Delete row and column

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

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

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

C#

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

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

C#

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

Work with Used Range

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

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

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

Work with worksheet's used range

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

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

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

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

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

Work with feature related used range

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

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

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

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

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

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

Freeze Panes in a Worksheet

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

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

Freeze Panes

You can freeze panes in a worksheet using the **FreezePanes() method ('FreezePanes Method' in the on-line documentation)** of the **IWorksheet interface (on-line documentation)**. This method will freeze the split panes according to the incoming row index and column index parameters.

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

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

C#
```csharp
// Adding worksheets to the workbook
```

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

Unfreeze Panes

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

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

C#

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

Work with Shape And Picture

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

This topic includes the following tasks:

1. Create different shape types
   - **Connector**
   - **Shape**
   - **Picture**

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

Connector

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

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

C#

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

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

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

Shape

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

Picture

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

Refer to the following example code when working with picture in Spread.Services:

C#

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

Shape Format

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

Solid Fill

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

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

C#

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

Gradient Fill

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

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

C#

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

Pattern Fill

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

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

C#

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

Picture Fill

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

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

C#

```csharp
// Add shape of picture type
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 20, 20, 100, 100);
string path = @"Images\flower.jpg";
FileStream stream = System.IO.File.Open(path, FileMode.Open);
shape.Fill.UserPicture(stream, ImageType.JPG);
stream.Dispose();
// Recolor the picture
shape.PictureFormat.ColorType = PictureColorType.Grayscale;
// Set picture's brightness and contrast ratio.
shape.PictureFormat.Brightness = 0.6;
shape.PictureFormat.Contrast = 0.3;
// Set height, width, x-axis offset and y-axis offset of the specified picture.
shape.PictureFormat.Crop.PictureOffsetX = 10;
shape.PictureFormat.Crop.PictureOffsetY = -5;
shape.PictureFormat.Crop.PictureWidth = 120;
shape.PictureFormat.Crop.PictureHeight = 80;
```

Texture Fill

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

documentation), **TextureOffsetX property (on-line documentation)**, **TextureOffsetY property (on-line documentation)** and **TextureVerticalScale property (on-line documentation)** to configure the layout of the texture.

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

C#

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

Line

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

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

C#

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

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

3D Formatting

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

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

C#

```
// To config shape's text and text style.
IShape shape = worksheet.Shapes.AddShape(AutoShapeType.Rectangle, 40, 40, 100, 100);
```

```
shape.TextFrame.TextRange.Font.Color.RGB = GrapeCity.Documents.Spread.Color.FromRGB(0,
255, 0);
shape.TextFrame.TextRange.Font.Bold = true;
shape.TextFrame.TextRange.Font.Italic = true;
shape.TextFrame.TextRange.Font.Size = 20;
shape.TextFrame.TextRange.Font.Strikethrough = true;

shape.TextFrame.TextRange.Paragraphs.Add("This is a rectangle shape.");
shape.TextFrame.TextRange.Paragraphs.Add("My name is Spread.");
shape.TextFrame.TextRange.Paragraphs[1].Runs.Add("Hello World!");

shape.TextFrame.TextRange.Paragraphs[1].Runs[0].Font.Strikethrough = false;
shape.TextFrame.TextRange.Paragraphs[1].Runs[0].Font.Size = 35;
```

Customize Worksheets

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

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

- **Configure display**
- **Set the tab color**
- **Set visibility**
- **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 Spread.Services.

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 = "Services";
worksheet.Range["C2"].Value = 4;
worksheet.Range["D1"].Value = "Spread";
worksheet.Range["D2"].Value = "ABCD";

// Set the specified sheet to be displayed from left to right.
worksheet.SheetView.DisplayRightToLeft = true;
```

Set the tab color

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

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

C#

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

Set visibility

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

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

C#

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

Define standard width and height

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

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

C#

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

Allow Sort

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

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

Following are the types of sorting available in Spread.Services.

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#

```csharp
//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#

```csharp
//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, Spread.Services 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#

```csharp
//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#

```csharp
// 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.FromRGB(255, 0, 0);
( worksheet.Range["A3"].Interior.Gradient as ILinearGradient).ColorStops[1].Color =
Color.FromRGB(146, 208, 80);
( worksheet.Range["A3"].Interior.Gradient as ILinearGradient).Degree = 90;

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

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

( worksheet.Range["A6"].Interior.Gradient as ILinearGradient).ColorStops[0].Color =
Color.FromRGB(255, 0, 255);
( worksheet.Range["A6"].Interior.Gradient as ILinearGradient).ColorStops[1].Color =
Color.FromRGB(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#

```csharp
// 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.FromRGB(0, 128, 0);
 worksheet.Range["B2"].Font.Color = Color.FromRGB(128, 0, 0);
 worksheet.Range["B3"].Font.Color = Color.FromRGB(0, 0, 128);
 worksheet.Range["B4"].Font.Color = Color.FromRGB(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#

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

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

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

Apply Filters

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

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

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

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

Apply number filters

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

C#

```csharp
// 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#

```csharp
//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#

```csharp
//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

```csharp
//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.FromRGB(255, 255, 0),
AutoFilterOperator.CellColor);
```

Apply filters by no fill

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

C#

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

Apply filters by icon

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

C#

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

Apply filters by no icon

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

C#

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

Manage Hyperlinks

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

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

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

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

Add hyperlinks

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

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

C#
```csharp
// Add a hyperlink link to external file
worksheet.Range["A1:B2"].Hyperlinks.Add(worksheet.Range["A1"],

@"C:\Documents\Spread.Core(TFS)\Spread.Core\SpreadCoreProject\Hyperlink\SampleFile.xlsx",
                          null,
                          "link to SampleFile.xlsx file.",
                          "SampleFile.xlsx");
```

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

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

Configure Hyperlinks

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

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

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

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

Delete Hyperlinks

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

Refer to the following example code to delete hyperlinks.

C#

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

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

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

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

Apply Grouping

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

Each group in Spread.Services is distinguished with a group header row with collapse and expand icons next to it that can be used for displaying or hiding information as and when required. You can set the **Show Detail property ('ShowDetail Property' in the on-line documentation)** of the **IRange Interface (on-line documentation)** to boolean true to expand a group to display rows and columns that have been hidden and false to collapse the expanded rows or columns.

Applying grouping in a spreadsheet involves the following tasks:

- Create Row or Column Group
- Remove a Group
- Set Summary Row

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

Create Row or Column Group

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

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

Apply row grouping

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

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

C#

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

Apply column grouping

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

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

C#

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

Set outline level for rows and columns

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

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

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

C#

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

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

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

Remove a Group

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

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

Ungroup rows and columns

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

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

C#

```
// Row Ungrouping
//1:5 rows' outline level will be 1.
worksheet.Range["1:5"].Ungroup();

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

Clear outline

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

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

C#

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

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

Collapse a group

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

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

C#

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

Set Summary Row

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

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

Refer to the following example code to set summary row.

C#

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

//Summary row will be row 4.
```

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

Apply Style

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

Applying style in a worksheet involves following tasks.

- Set Sheet Styling
- Create and Delete Named Style
- Get or Set Named Style

Some of the built-in styles in Spread.Services 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 (on-line documentation)** of the **IRange interface (on-line documentation)**. A cell interior can be of three types, namely, solid fill, pattern fill and gradient fill.

Solid fill

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

Refer to the following example code to set solid fill.

C#

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

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

Pattern fill

You can integrate pattern fill in cells using the **Pattern** property of the IInterior interface to one of the valid pattern types.

Pattern fill also consists of two parts - background Color and foreground Color.

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

Refer to the following example code to set pattern fill.

C#

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

Gradient Fill

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

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

Linear gradient fill

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

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

C#

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

```
Color.FromRGB(255, 0, 0);
(worksheet.Range["C1"].Interior.Gradient as ILinearGradient).ColorStops[1].Color =
Color.FromRGB(255, 255, 0);

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

Rectangular gradient fill

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

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

C#

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

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

Set font

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

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

C#

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

Set border

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

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

C#

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

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

Set number format

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

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

C#

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

Set alignment

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

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

C#

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

Set protection

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

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

C#

```csharp
//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 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 named styles as and when required. In Spread.Services, **Styles** refers to the named style collection that stores both the built-in and custom named styles.

Create custom named style

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

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

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

C#

```csharp
//Add custom name style.
```

```csharp
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.FromRGB(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.Spread.Pattern.Down;
style.Interior.PatternColor = Color.FromRGB(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.
```

Set custom named style

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

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

C#

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

Apply Comments

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

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

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

- **Delete comment**

Add comment to a cell

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

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

C#

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

Set comment layout

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

Refer to the following example code to set comment layout.

C#

```csharp
//Configure comment layout

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

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

Show/Hide comment

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

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

C#

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

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

Author comments

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

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

C#

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

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

Set rich text for comment

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

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

C#

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

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

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

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

commentC3.Visible = true;

var text = commentC3.Text;
```

Delete comment

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

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

C#

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

Manage Formulas

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

Shared below is the descending order of operations for Spread.Services 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 Spread.Services, managing formulas involves the following tasks.

- Set Formula to Range
- Set Table Formula
- Set Array Formula

Set Formula to Range

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

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

C#

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

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

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

Reference style

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

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

C#

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

Set Table Formula

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

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

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

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

Reference operators

In Spread.Services, 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.

Spread.Services 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#

```csharp
// 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 Spread.Services, you can use **FormulaArray property (on-line documentation)** of the **IRange interface (on-line documentation)** to set array formula for a range. In case, you want to find out whether a range has array formula or not, you can use the **HasArray property (on-line documentation)** of the IRange interface. In order to get an entire array if specified range is part of an array, you can use **CurrentArray property (on-line documentation)**.

Refer to the following example code to set array formula and get entire array:

C#

```csharp
// Setting cell value using arrays
worksheet.Range["E4:J5"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6}
};
```

```csharp
worksheet.Range["I6:J8"].Value = new object[,]
{
    {2, 2},
    {3, 3},
    {4, 4}
};
// To set array formula for range.
//O      P       Q
//2      4       #N/A
//12     15      #N/A
//#N/A   #N/A    #N/A
worksheet.Range["O9:Q11"].FormulaArray = "=E4:G5*I6:J8";

//O9's current array is "O9:Q11". Current array gets the entire array.
var currentarray = worksheet.Range["O9"].CurrentArray.ToString();
```

Manage Data in Spread Component

The tasks associated with handling data in spread component includes the following:

- Use Chart
- Use Sparkline
- Use Table
- Use Pivot Table
- Use Slicer
- Import and Export .xlsx Document
- Import and Export JSON Stream

Use Chart

You can use charts in spreadsheets to graphically interpret data and visualize large volumes of information quickly and efficiently.

Working with charts involves the following tasks:

- Create and Delete Chart
- Configure Chart
- Customize Chart Objects

Create and Delete Chart

Spread.Services allows users to create and delete chart in spreadsheets as per their requirements.

You can create and delete chart using the properties and methods of the **IShapes Interface (on-line documentation)** and the **IChart interface (on-line documentation)**

Create Chart

You can create chart in a worksheet by using the **AddChart method (on-line documentation)** of the IShapes interface.

Refer to the following example code to create a chart.

C#
```csharp
    //Add Chart
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
```

```
    {
        {null, "Revenue", "Profit", "Sales"},
        {"North", 10, 25, 25},
        {"East", 51, 36, 27},
        {"South", 52, 85, 30},
        {"West", 22, 65, 65}
    };
```

Delete Chart

You can delete an existing chart by using **Delete method (on-line documentation)** of the IChart interface.

Refer to the following example code to delete a chart from your spreadsheet.

C#

```csharp
// Delete Chart
shape.Chart.Delete();
```

Configure Chart

In Spread.Services, 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 Spread.Services, you can use the properties of the **IChart Interface (on-line documentation)** to set up the chart title as per your choice. When working with chart title, you can perform the following tasks:

- **Set formula for chart title**
- **Set format for chart title and font style**

Set formula for chart title

Refer to the following example code to set formula for chart title.

C#

```csharp
//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#

```csharp
//Set chart title's format and font style.
shape.Chart.HasTitle = true;
//shape.Chart.ChartTitle.Text = "aaaaa";
shape.Chart.ChartTitle.Font.Bold = true;
shape.Chart.ChartTitle.Format.Fill.Color.RGB = Color.Red;
shape.Chart.ChartTitle.Format.Line.Color.RGB = Color.Blue;
```

Chart Area

In Spread.Services, you can use the properties of the **IChartArea interface (on-line documentation)** to set up the chart area as per your preferences.

This topic includes the following tasks:

- **Configure chart area style**
- **Set chart area format**

Configure chart area style

You can configure the chart area style by changing its font, format and other attributes using the **Font property (on-line documentation)**, **Format property (on-line documentation)** and **RoundedCorners property (on-line documentation)** of the IChartArea interface.

Refer to the following example code to configure chart area style in your worksheet.

C#

```csharp
//Configure chart area style
IShape shape = worksheet.Shapes.AddChart(ChartType.Column3D, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IChartArea chartarea = shape.Chart.ChartArea;
//Format.
chartarea.Format.Fill.Color.RGB = Color.Gray;
chartarea.Format.Line.Color.RGB = Color.Gold;
chartarea.Format.ThreeD.RotationX = 60;
chartarea.Format.ThreeD.RotationY = 20;
chartarea.Format.ThreeD.RotationZ = 100;
chartarea.Format.ThreeD.Z = 20;
chartarea.Format.ThreeD.Perspective = 20;
chartarea.Format.ThreeD.Depth = 5;
//Font
chartarea.Font.Bold = true;
chartarea.Font.Italic = true;
chartarea.Font.Color.RGB = Color.Red;
//Rounded corners.
chartarea.RoundedCorners = true;
```

Set chart area format

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

C#

```csharp
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}
```

```csharp
};
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 Spread.Services, you can use the properties of the **IPlotArea Interface (on-line documentation)** to set up the plot area in a chart as per your preferences.

Configure plot area format

You can configure the plot area format by changing its fill color, line color and other attributes using the **Format property (on-line documentation)** of the IPlotArea interface.

Refer to the following example code to configure plot area format for a chart inserted in your worksheet.

C#

```csharp
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 Spread.Services, 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 Spread.Services:

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.
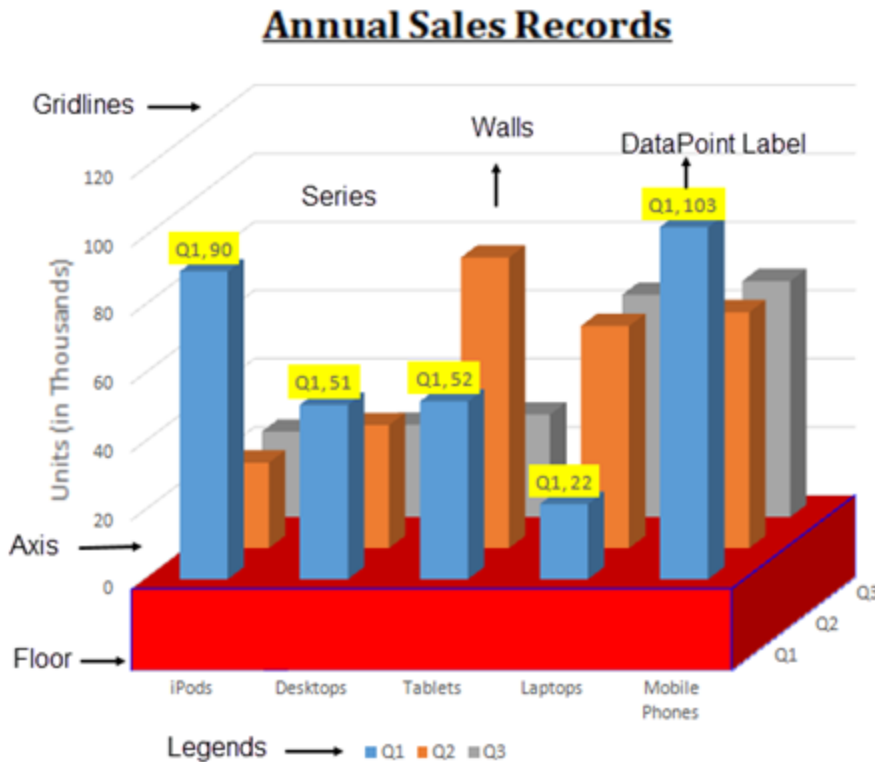
Series

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 Spread.Services, the SeriesCollection can be used to create chart series. The properties and methods of the **ISeries interface (on-line documentation)** and the **ISeriesCollection interface (on-line documentation)** allows users to add individual series, access it, delete it and perform other useful operations on it as per the requirements.

Refer to the following example code to add series in your chart.

C#

```csharp
// 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},
```

```csharp
        {"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 Spread.Services, you can configure chart series using the following in your spreadsheet:

- **DataPoint**
- **DataLabel**
- **Trendline**
- **ChartGroup**

- **DropLine,HiLoLine and SeriesLine**
- **Up-Down Bars**

DataPoint

The Points collection in Spread.Services is used to represent all the points in a specific series and the indexer notation of the **IPoints interface (on-line documentation)** to get a specific point in the series. Also, you can use the **DataLabel property (on-line documentation)** of the **IPoint interface (on-line documentation)** to get data label of a specific point.

Set the format of DataPoint

Refer to the following example code to set data point format for the chart inserted in your worksheet.

C#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
ISeries series2 = shape.Chart.SeriesCollection[1];
ISeries series3 = shape.Chart.SeriesCollection[2];

series1.Format.Fill.Color.RGB = Color.Blue;
series1.Points[2].Format.Fill.Color.RGB = Color.Green;
```

Configure secondary section for pie of a pie chart

You can use the **SecondaryPlot property (on-line documentation)** of the IPoint interface to set if the point lies in the secondary section of either a pie of pie chart or a bar of pie chart.

Refer to the following example code to configure secondary section for pie of a pie chart.

C#

```csharp
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 Spread.Services is used to represent the collection of all the data labels for the specified series.

You can use the **Font property (on-line documentation)** and **Format property (on-line documentation)** of the **IDataLabel interface (on-line documentation)** to set font style, fill, line and 3-D formatting for all the data labels of the specified series. You can also configure the layout of the data labels using other properties of the IDataLabel interface.

Set all data labels and specific data label format for series

Refer to the following example code to set series' all data labels and specific data label format.

C#

```csharp
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#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;
```

```
//customize data lables' text.
series1.DataLabels.ShowCategoryName = true;
series1.DataLabels.ShowSeriesName = true;
series1.DataLabels.ShowLegendKey = true;
```

Trendline

The Trendlines collection in Spread.Services is used to represent a collection of trend lines for a specific series. You can use the **Add Method (on-line documentation)** of the **ITrendlines interface (on-line documentation)** to create a new trendline for a specific series. Also, you can use the indexer notation of the ITrendlines interface to get a specific trend line.

Add trendline for series and configure its style

Refer to the following example code to add trendline for series and configure its style.

C#

```csharp
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#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.Trendlines.Add();
series1.Trendlines[0].Type = TrendlineType.Linear;
series1.Trendlines[0].Forward = 5;
series1.Trendlines[0].Backward = 0.5;
series1.Trendlines[0].Intercept = 2.5;
```

```
series1.Trendlines[0].DisplayEquation = true;
series1.Trendlines[0].DisplayRSquared = true;


series1.Trendlines.Add();
series1.Trendlines[1].Type = TrendlineType.Polynomial;
series1.Trendlines[1].Order = 3;
```

Chart Group

Chart Group contains common settings for one or more series. Typically, it is a group of specific featured series.

Set varied colors for column chart with one series

Refer to the following example code to set different colors for a column chart which has only one series.

C#

```csharp
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#

```csharp
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#

```csharp
//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#

```csharp
//Configure bubble chart's layout
IShape shape = worksheet.Shapes.AddChart(ChartType.Bubble, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

shape.Chart.XYGroups[0].BubbleScale = 150;
shape.Chart.XYGroups[0].SizeRepresents = SizeRepresents.SizeIsArea;
shape.Chart.XYGroups[0].ShowNegativeBubbles = true;
```

Configure the layout of the doughnut chart

Refer to the following example code to configure the layout of the doughnut chart as per your preferences.

C#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.Doughnut, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
```

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

ISeries series1 = shape.Chart.SeriesCollection[0];
series1.HasDataLabels = true;

shape.Chart.DoughnutGroups[0].FirstSliceAngle = 50;
shape.Chart.DoughnutGroups[0].DoughnutHoleSize = 20;
```

Dropline, HiLoline and SeriesLine

You can use the **HasDropLines property (on-line documentation)**, **HasHiLoLines property (on-line documentation)**, **HasSeriesLines property (on-line documentation)**, **DropLines property (on-line documentation)**,**HiLoLines property (on-line documentation)**, **SeriesLines property (on-line documentation)** of the **IChartGroup interface (on-line documentation)** to configure Dropline, HiLoline and Series lines in a chart.

Configure the drop lines of the line chart

Refer to the following example code to configure the drop lines of the line chart as per your preferences.

C#

```csharp
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#

```csharp
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#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnStacked, 200, 100, 300, 300);
```

```
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.ColumnGroups[0].HasSeriesLines = true;
shape.Chart.ColumnGroups[0].SeriesLines.Format.Line.Color.RGB = Color.Red;
```

Configure the connector lines for pie of a pie chart

Refer to the following example code to configure the connector lines for pie of a pie chart as per your preferences.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.PieOfPie, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.PieGroups[0].HasSeriesLines = true;
shape.Chart.PieGroups[0].SeriesLines.Format.Line.Color.RGB = Color.Red;
```

Up-Down Bars

You can use the **HasUpDownBars property (on-line documentation)**, **DownBars property (on-line documentation)** and **UpBars property (on-line documentation)** of the IChartGroup interface up-down bars in a chart to configure the style of the up bars and the down bars as per your preferences.

Configure the up-down bars for the line chart

Refer to the following example code to configure the up-down bars for the line chart as per your preferences.

C#

```
IShape shape = worksheet.Shapes.AddChart(ChartType.Line, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

shape.Chart.LineGroups[0].HasUpDownBars = true;
shape.Chart.LineGroups[0].UpBars.Format.Fill.Color.RGB = Color.Green;
shape.Chart.LineGroups[0].DownBars.Format.Fill.Color.RGB = Color.Red;
```

Walls

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

Spread.Services enables users to set up a chart as per their custom preferences by defining the thickness, fill color, line color and format of the back wall as well as the side wall, using the properties of the **IWall Interface (on-line documentation)** and the **IChart Interface (on-line documentation)**.

Refer to the following example code to configure the walls of the chart inserted in a worksheet.

C#

```csharp
//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 Spread.Services, an axis in a chart can have a title, major tick mark, minor tick mark, tick mark labels, major gridlines and minor gridlines.

There are three types of axes in charts:

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

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

You can use the properties of the **IAxis Interface (on-line documentation)** to configure category axis, value axis and series axis in a chart.

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

C#

```csharp
//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 Spread.Services, you can configure chart axis using the following elements in your spreadsheet:

- **Axis title**
- **Gridlines**
- **Display unit label**
- **Tick labels**

Axis title

While configuring chart axis, you can set the style for the axis title as per your preferences by using the **AxisTitle property (on-line documentation)** of the **IAxis interface (on-line documentation)**.

Refer to the following example code to configure axis title's layout.

C#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);
IAxis value_axis = shape.Chart.Axes.Item(AxisType.Value);
category_axis.HasTitle = true;
category_axis.AxisTitle.Format.Fill.Color.RGB = Color.Pink;
category_axis.AxisTitle.Text = "aaaaaaaaaa";
category_axis.AxisTitle.Font.Size = 20;
category_axis.AxisTitle.Font.Color.RGB = Color.Green;
category_axis.AxisTitle.Font.Strikethrough = true;
```

Gridlines

While configuring the axis of a chart, you can also set the style of major and minor gridlines as per your choice using the **HasMajorGridlines property (on-line documentation)**, **HasMinorGridlines property (on-line documentation)**, **MajorGridlines property (on-line documentation)** and **MinorGridlines property (on-line documentation)** of the IAxis interface.

Refer to the following example code to set major and minor gridlines' style.

C#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300, 300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IAxis value_axis = shape.Chart.Axes.Item(AxisType.Value);
IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);
category_axis.HasMajorGridlines = true;
category_axis.HasMinorGridlines = true;
category_axis.MajorGridlines.Format.Line.Color.RGB = Color.Red;
category_axis.MajorGridlines.Format.Line.Weight = 3;
category_axis.MinorGridlines.Format.Line.Color.RGB = Color.Green;
category_axis.MinorGridlines.Format.Line.Weight = 1;
category_axis.MinorGridlines.Format.Line.Style = LineStyle.ThickThin;
```

Display unit label

While configuring the chart axis in your worksheet, you can also set the display unit for the axis and configure its label style using the **DisplayUnit property (on-line documentation)**, **DisplayUnitLabel property (on-line documentation)** and **HasDisplayUnitLabel property (on-line documentation)** of the IAxis interface.

Refer to the following example code to set display unit for the axis and configure its label style.

C#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

IAxis category_axis = shape.Chart.Axes.Item(AxisType.Category);
IAxis value_axis = shape.Chart.Axes.Item(AxisType.Value);
value_axis.DisplayUnit = DisplayUnit.Hundreds;
value_axis.HasDisplayUnitLabel = true;
value_axis.DisplayUnitLabel.Font.Color.RGB = Color.Green;
value_axis.DisplayUnitLabel.Font.Italic = true;
value_axis.DisplayUnitLabel.Format.Fill.Color.RGB = Color.Pink;
value_axis.DisplayUnitLabel.Format.Line.Color.RGB = Color.Red;
```

Tick labels

While configuring the axis of a chart, you can also set the position and layout of the tick-mark labels as per your choice using the **TickLabelPosition property (on-line documentation)**, **TickLabels property (on-line documentation)**, **TickLabelSpacing property (on-line documentation)**, **TickLabelSpacingIsAuto property (on-line documentation)** and **TickMarkSpacing property (on-line documentation)** of the IAxis interface.

Refer to the following example code to configure the tick mark label's position and layout.

C#

```csharp
IShape shape = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 200, 100, 300,
300);
worksheet.Range["A1:D6"].Value = new object[,]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
};
shape.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true, true);

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

//tick-mark labels' fill will be green according to axis's format.
category_axis.Format.Fill.Color.RGB = Color.Green;

category_axis.TickLabelPosition = TickLabelPosition.NextToAxis;
category_axis.TickLabelSpacing = 2;
category_axis.TickLabels.Font.Color.RGB = Color.Red;
```

```
category_axis.TickLabels.Font.Italic = true;
category_axis.TickLabels.NumberFormat = "#,##0.00";
category_axis.TickLabels.Offset = 100;
```

Floor

Floor represents the floor of a three-dimensional chart. The area of a 3-D chart can be formatted using floor as the charting object.

In Spread.Services, you can use the properties and methods of the **IFloor interface (on-line documentation)** to set the line and fill format of the floor along with its thickness.

Refer to the following example code to configure the format of floor in a chart.

C#

```csharp
//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

Spread.Services allows you to insert data labels in a chart to ensure the information depicted in it can be easily interpreted and visualized. You can add data labels in a chart using the properties and methods of the **IPoint interface (on-line documentation)** and the **ISeries interface (on-line documentation)**.

Refer to the following example code to set data labels in a chart and customize the data label text.

C#

```csharp
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.
```

```csharp
series1.DataLabels.Format.Fill.Color.RGB = Color.Green;
series1.DataLabels.Format.Line.Color.RGB = Color.Red;
series1.DataLabels.Format.Line.Weight = 3;
//set series1's specific data label's format.
series1.DataLabels[2].Format.Fill.Color.RGB = Color.Yellow;
series1.Points[2].DataLabel.Format.Line.Color.RGB = Color.Blue;
series1.Points[2].DataLabel.Format.Line.Weight = 5;

//Customize data label's text.
IShape shape2 = worksheet.Shapes.AddChart(ChartType.ColumnClustered, 550, 50, 300,
300);
shape2.Chart.SeriesCollection.Add(worksheet.Range["A1:D6"], RowCol.Columns, true,
true);
ISeries series2 = shape2.Chart.SeriesCollection[0];
series2.HasDataLabels = true;
//customize data lables' text.
series2.DataLabels.ShowCategoryName = true;
series2.DataLabels.ShowSeriesName = true;
series2.DataLabels.ShowLegendKey = true;
```

Legends

In order to enable users to quickly interpret and understand the charted data, Legends (visual charting elements) automatically appear in spreadsheets when you finish creating a chart.

Legends are also known as keys and are associated with the graphic data plotted on the chart. Usually, they are located at the right side of the chart. From a wider perspective, they facilitate end users to determine series and series points representing distinct data groups in a spreadsheet.

Typically, legends depict series names by listing and identifying the data points that belong to a particular series. Corresponding to the data, each legend entry appearing on the worksheet can be shown with the help of a legend marker along with the legend text that identifies it.

In Spread.Services, you can even customize the legend text, configure the position and layout of the legend, reset the font style for the legend entries, delete legend and its entries as and when you want using the properties and methods of the **ILegend interface (on-line documentation)** and the **IChart interface (on-line documentation)**.

Refer to the following example code to configure some useful legend settings in your chart.

C#
```csharp
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();
```

Use Sparkline

You can use sparklines in cells to insert graphical illustration of trends in data. Sparklines are particularly useful for analytical dashboards, presentations, business reports etc. The sparkline displays the most recent value as the rightmost data point and compares it with earlier values on a scale, allowing you to view general changes in data over time.

Using sparklines includes the following tasks.

- **Add a group of new sparklines**
- **Clear sparkline**
- **Clear sparkline groups**
- **Create a group of existing sparklines**
- **Add group of new sparklines with Date Axis**
- **Configure layout of sparkline**

Add a group of new sparklines

You can add a group of new sparklines for each row or column of data in your worksheet by first specifying the data range and then using the **Add method (on-line documentation)** of the **ISparklineGroups interface (on-line documentation)**.

Refer to the following example code to add a group of new sparklines.

C#
```
//Create workbook and access its first worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
    {
        {1, 2, 3},
```

```
            {4, 5, 6},
            {7, 8, 9},
            {10, 11, 12}
        };
    // Add a group of new sparklines
    worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
```

Clear sparkline

You can remove a sparkline from your worksheet by first specifying the data range and then using the **Clear method (on-line documentation)** of the ISparklineGroups interface.

Refer to the following example code to clear sparkline.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
    {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9},
        {10, 11, 12}
    };

worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
// Defining data in the range
worksheet.Range["F1:H4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
// Add a group of new sparklines
worksheet.Range["J1:J4"].SparklineGroups.Add(SparkType.Line, "F1:H4");

//Clear D2 and J1 cell's sparkline.
worksheet.Range["D2, J1"].SparklineGroups.Clear();
```

Clear sparkline groups

You can remove a group of sparklines (added for a row or column) from your worksheet by specifying the data range and then using the **ClearGroups method (on-line documentation)** of the ISparklineGroups interface.

Refer to the following example code to clear sparkline groups.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
// Add a group of new sparklines
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
// Defining data in the range
worksheet.Range["F1:H4"].Value = new object[,]
  {
    {1, 2, 3},
    {4, 5, 6},
```

```
      {7, 8, 9},
      {10, 11, 12}
  };
// Add a group of new sparklines
worksheet.Range["J1:J4"].SparklineGroups.Add(SparkType.Line, "F1:H4");

//Clear sparkline groups.
worksheet.Range["D2, J1"].SparklineGroups.ClearGroups();
```

Create a group of existing sparklines

You can create a group of existing sparklines by specifying the data range and then using the **Group() method ('Group Method' in the on-line documentation)** of the ISparklineGroups interface.

Refer to the following example code to create a group of existing sparklines.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
      {
          {1, 2, 3},
          {4, 5, 6},
          {7, 8, 9},
          {10, 11, 12}
      };
// Add a group of new sparklines
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
// Defining data in the range
worksheet.Range["F1:H4"].Value = new object[,]
          {
              {1, 2, 3},
              {4, 5, 6},
              {7, 8, 9},
              {10, 11, 12}
          };
// Add a group of new sparklines
worksheet.Range["J1:J4"].SparklineGroups.Add(SparkType.Column, "F1:H4");

//Create a new group, according to Range["J2"]'s sparkline group setting.
worksheet.Range["A1:J4"].SparklineGroups.Group(worksheet.Range["J2"]);
```

Add group of new sparklines with Date Axis

You can add a group of new sparklines with date axis by first specifying the data range and then using the **DateRange property (on-line documentation)** of the **ISparklineGroup interface (on-line documentation)**.

Refer to the following example code to add group of new sparkline with date axis.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
  {
      {1, 2, 3},
      {4, 5, 6},
      {7, 8, 9},
      {10, 11, 12}
  };
// Add a group of new sparklines
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
worksheet.Range["A7:C7"].Value = new object[] { new DateTime(2011, 12, 16), new
DateTime(2011, 12, 17), new DateTime(2011, 12, 18) };
```

```
//Set horizontal axis's Date range.
worksheet.Range["D1"].SparklineGroups[0].DateRange = "A7:C7";

worksheet.Range["D1"].SparklineGroups[0].Axes.Horizontal.Axis.Visible = true;
worksheet.Range["D1"].SparklineGroups[0].Axes.Horizontal.Axis.Color.Color =
Color.Green;
worksheet.Range["D1"].SparklineGroups[0].Axes.Vertical.MinScaleType =
SparkScale.SparkScaleCustom;
worksheet.Range["D1"].SparklineGroups[0].Axes.Vertical.MaxScaleType =
SparkScale.SparkScaleCustom;
worksheet.Range["D1"].SparklineGroups[0].Axes.Vertical.CustomMinScaleValue = -2;
worksheet.Range["D1"].SparklineGroups[0].Axes.Vertical.CustomMaxScaleValue = 8;
```

Configure layout of sparkline

You can configure the layout of the sparkline by using the properties of the ISparklineGroup interface.

Refer to the following example code to configure the layout of the sparkline.

C#

```
// Defining data in the range
worksheet.Range["A1:C4"].Value = new object[,]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
// Adding sparkline
worksheet.Range["D1:D4"].SparklineGroups.Add(SparkType.Line, "A1:C4");
// Configuring the layout
var sparklinegroup = worksheet.Range["D1"].SparklineGroups[0];
sparklinegroup.LineWeight = 2.5;
sparklinegroup.Points.Markers.Color.Color = Color.Red;
sparklinegroup.Points.Markers.Visible = true;
sparklinegroup.SeriesColor.Color = Color.Purple;
```

Use Table

Tabular data is easy to read, interpret, visualize and manage.

Spread.Services supports the use of tables in worksheets by enabling users to perform different tasks on a table that help them in handling large chunks of data quickly and efficiently.

In Spread.Services, you can use table in the following ways:

- Create and Delete Tables
- Modify Tables
- Apply Table Sort
- Set Table Filters
- Add and Delete Table Columns and Rows
- Apply Table Style

Create and Delete Tables

In Spread.Services, you can create and delete tables in spreadsheets using the **Add method (on-line documentation)** of the **ITables interface (on-line documentation)** and the **Delete Method (on-line documentation)** of the **ITable Interface (on-line documentation)**, or simply transform a cell range into a

table by specifying the existing data lying in a worksheet.

Refer to the following example code to create and delete tables in a worksheet.

C#

```csharp
//Create workbook and access its first worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
//Add first table
ITable table1 = worksheet.Tables.Add(worksheet.Range["A1:E5"], true);
//Add second table
ITable table2 = worksheet.Tables.Add(worksheet.Range["F1:G5"], true);
//Delete second Table
worksheet.Tables[1].Delete();
```

Modify Tables

While working with tables in Spread.Services, you can configure it as per your spreadsheet requirements by modifying the table using the properties and methods of the **ITable interface (on-line documentation)**.

- **Modify table range**
- **Modify table areas**
- **Modify totals row of table column**

Modify table range

Spread.Services allows you to modify the table range of your worksheet using the **Resize method (on-line documentation)** of the ITable interface.

Refer to the following example code to modify table range.

C#

```csharp
//Modify table range
table.Resize(worksheet.Range["B1:E4"]);
```

Modify table areas

You can modify the value of specific table areas by accessing its header range, data range and total range using the **HeaderRange property (on-line documentation)**, **DataRange property (on-line documentation)** and **TotalsRange property (on-line documentation)** of the ITable interface.

Refer to the following example code to modify table areas in your worksheet.

C#

```csharp
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;
```

```csharp
//Table second column name set to "Age".
worksheet.Tables[0].HeaderRange[0, 1].Value = "Age";

//"Age" Column's second row's value set to 23.
worksheet.Tables[0].DataRange[1, 1].Value = 23;

//"Age" column's total row function set to average.
worksheet.Tables[0].TotalsRange[0, 1].Formula = "=SUBTOTAL(101,[Age])";
```

Modify totals row of table column

When you need to make changes to the total row's calculation function of a specific table column, you can use the **TotalsCalculation property (on-line documentation)** of the **ITableColumn interface (on-line documentation)**.

Refer to the following example code to modify column total row's calculation function.

C#
```csharp
worksheet.Tables.Add(worksheet.Range["A1:C5"], true);
worksheet.Tables[0].ShowTotals = true;

//Populate table values
worksheet.Range["A2"].Value = 3;
worksheet.Range["A3"].Value = 4;
worksheet.Range["A4"].Value = 2;
worksheet.Range["A5"].Value = 1;
worksheet.Range["B1"].Value = 13;
worksheet.Range["B2"].Value = 32;
worksheet.Range["B3"].Value = 41;
worksheet.Range["B4"].Value = 12;
worksheet.Range["B5"].Value = 16;
worksheet.Range["C1"].Value = 1;
worksheet.Range["C2"].Value = 3;
worksheet.Range["C3"].Value = 4;
worksheet.Range["C4"].Value = 15;
worksheet.Range["C5"].Value = 18;

//First table column's total row calculation fuction will be "=SUBTOTAL(101,[Column1])"
worksheet.Tables[0].Columns[1].TotalsCalculation = TotalsCalculation.Count;
```

Apply Table Sort

Spread.Servcies provides an option to apply sorting on a specific table in the worksheet. To accomplish this, you can use the **Sort property (on-line documentation)** of the **ITable interface (on-line documentation)**. The **Apply ('Apply Method' in the on-line documentation)** method is used to apply the selected sort state and display the results.

Refer to the following example code to apply table sorting in a worksheet.

C#
```csharp
// Assigning Value to the range
 worksheet.Range["A2"].Value = 3;
 worksheet.Range["A3"].Value = 4;
 worksheet.Range["A4"].Value = 2;
 worksheet.Range["A5"].Value = 1;

 worksheet.Range["B2"].Value = 1;
 worksheet.Range["B3"].Value = 2;
 worksheet.Range["B4"].Value = 3;
 worksheet.Range["B5"].Value = 4;
```

```
    worksheet.Range["F2"].Value = "aaa";
    worksheet.Range["F3"].Value = "bbb";
    worksheet.Range["F4"].Value = "ccc";
    worksheet.Range["F5"].Value = "ddd";

    worksheet.Range["B2:B5"].FormatConditions.AddIconSetCondition();

//Sort by column A firstly, then by column B.
ValueSortField key1 = new ValueSortField(worksheet.Range["A1:A2"],
SortOrder.Ascending);
IconSortField key2 = new IconSortField(worksheet.Range["B1:B2"],
workbook.IconSets[IconSetType.Icon3Arrows][1], SortOrder.Descending);

table.Sort.SortFields.Add(key1);
table.Sort.SortFields.Add(key2);
table.Sort.Apply();
```

Set Table Filters

When you have a lot of data to handle, you can create as many tables on a spreadsheet as you want and apply separate filters on columns of each of the table to manage information in an effective manner.

Spread.Services provides users with the ability to set table filters while setting up worksheets for ensuring improved data analysis.

When applying filters on tables in worksheets created, you need to first get the table range and then use the **AutoFilter method (on-line documentation)** of the **IRange interface (on-line documentation)** to filter the table.

Refer to the following example code to set table filters in a worksheet.

C#

```
//Add Table
ITable table = worksheet.Tables.Add(worksheet.Range["A1:E5"], true);

//Populate table values
worksheet.Range["A2"].Value = 3;
worksheet.Range["A3"].Value = 4;
worksheet.Range["A4"].Value = 2;
worksheet.Range["A5"].Value = 1;

//Apply table filter
worksheet.Tables[0].Range.AutoFilter(0, ">2");
```

Add and Delete Table Columns and Rows

You can add and delete columns and rows of a table using the methods and properties of the following interfaces:

- **ITableColumns Interface (on-line documentation)** - Represents the table columns collection.
- **ITableRows Interface (on-line documentation)** - Represents the table rows collection.
- **ITableColumn Interface (on-line documentation)** - Represents an individual table column.
- **ITableRow Interface (on-line documentation)** - Represents an individual table row.

Add and delete table columns

To add and delete table columns, you can use the **Add method (on-line documentation)** of the ITableColumns interface and the **Delete method (on-line documentation)** of the ITableColumn interface respectively.

Refer to the following example code in order to add and delete table columns.

C#
```csharp
//Create first table
ITable table1 = worksheet.Tables.Add(worksheet.Range["D3:I6"], true);

//Create second table
ITable table2 = worksheet.Tables.Add(worksheet.Range["A1:C6"], true);

//Insert a table column before first column in first table
table1.Columns.Add(0);

//Insert a table column before first column in second table
table2.Columns.Add(0);

//Delete the first table column from the first table.
worksheet.Tables[0].Columns[0].Delete();
```

Add and delete table rows

To add and delete table rows, you can use the **Add method (on-line documentation)** of the ITableRows interface and the **Delete method (on-line documentation)** of the ITableRow interface respectively.

Refer to the following example code in order to add and delete table rows.

C#
```csharp
//insert a new row at the end of the first table.
table1.Rows.Add();

//insert a new row at the end of the second table.
table2.Rows.Add();

//Delete the second row in the second table.
table2.Rows[1].Delete();
```

Apply Table Style

In Spread.Services, you can create custom table style elements and apply them to your worksheet using the **ITableStyle Interface (on-line documentation)**. Also, you can format a table using any of the predefined table styles provided by Spread.services.

Typically, each workbook possesses an **ITableStyle collection ('ITableStyleCollection Interface' in the on-line documentation)** that is used to store both built-in and custom table styles. If you want to insert a custom table style, you use the **Add method (on-line documentation)** of the **ITables interface (on-line documentation)**, which returns the IStyle object representing the corresponding table style instance.

C#
```csharp
//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#

```csharp
//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.FromRGB(24, 232, 192);

//Set FirstColumnStripe element style.
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].Font.Bold = true;
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].Font.Color =
Color.FromRGB(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.FromRGB(255, 255, 0);
style.TableStyleElements[TableStyleElementType.FirstColumnStripe].StripeSize = 2;

//Set SecondColumnStripe element style.
style.TableStyleElements[TableStyleElementType.SecondColumnStripe].Font.Color =
Color.FromRGB(255, 0, 255);
style.TableStyleElements[TableStyleElementType.SecondColumnStripe].Borders.LineStyle =
BorderLineStyle.DashDot;
style.TableStyleElements[TableStyleElementType.SecondColumnStripe].Borders.Color =
Color.FromRGB(42, 105, 162);
style.TableStyleElements[TableStyleElementType.SecondColumnStripe].Interior.Color =
Color.FromRGB(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 Spread.services, 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#

```csharp
ITable table = worksheet.Tables.Add(worksheet.Range["A1:B2"]);

//Show table header row.
table.ShowHeaders = true;

//To make "first row stripe" and "second row stripe" table style element's style
effective.
table.ShowTableStyleRowStripes = false;

//Hide auto filter drop down button.
table.ShowAutoFilterDropDown = false;

//To make "first column" table style element's style effective.
table.ShowTableStyleFirstColumn = true;

//Show table total row.
table.ShowTotals = true;

//To make "last column" table style element's style effective.
table.ShowTableStyleLastColumn = true;

//To make "first column stripe" and "second column stripe" table style element's style
effective.
table.ShowTableStyleColumnStripes = true;

//Unfilter table column filters, and hide auto filter drop down button.
table.ShowAutoFilter = false;
```

Use Pivot Table

Spread.Services 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

Create Pivot Table

Spread.Services allows you to create pivot tables in a spreadsheet. But, before generating a pivot table, you first need to create the pivot cache using the PivotCaches collection to stores all the pivot caches in the workbook.

After you accomplish this, you need to call the **Create method (on-line documentation)** of the **IPivotCaches interface (on-line documentation)** to create a new pivot cache. After creating pivot cache, the next step is to create the new pivot table using **CreatePivotTable ('CreatePivotTable Method' in the on-line documentation)** method of the **IPivotCache interface (on-line documentation)**.

Refer to the following example code to create pivot table in a worksheet.

C#

```csharp
//Source data for PivotCache
object[,] sourceData = new object[,] {
{ "Order ID", "Product",  "Category",   "Amount", "Date",
"Country" },
    { 1,          "Carrots",  "Vegetables",  4270,     new DateTime(2012, 1, 6),
"United States" },
    { 2,          "Broccoli", "Vegetables",  8239,     new DateTime(2012, 1, 7),
"United Kingdom" },
    { 3,          "Banana",   "Fruit",       617,      new DateTime(2012, 1, 8),
"United States" },
    { 4,          "Banana",   "Fruit",       8384,     new DateTime(2012, 1, 10),
"Canada" },
    { 5,          "Beans",    "Vegetables",  2626,     new DateTime(2012, 1, 10),
"Germany" },
    { 6,          "Orange",   "Fruit",       3610,     new DateTime(2012, 1, 11),
"United States" },
    { 7,          "Broccoli", "Vegetables",  9062,     new DateTime(2012, 1, 11),
"Australia" },
    { 8,          "Banana",   "Fruit",       6906,     new DateTime(2012, 1, 16), "New
Zealand" },
    { 9,          "Apple",    "Fruit",       2417,     new DateTime(2012, 1, 16),
"France" },
    { 10,         "Apple",    "Fruit",       7431,     new DateTime(2012, 1, 16),
"Canada" },
    { 11,         "Banana",   "Fruit",       8250,     new DateTime(2012, 1, 16),
"Germany" },
    { 12,         "Broccoli", "Vegetables",  7012,     new DateTime(2012, 1, 18),
"United States" },
    { 13,         "Carrots",  "Vegetables",  1903,     new DateTime(2012, 1, 20),
"Germany" },
    { 14,         "Broccoli", "Vegetables",  2824,     new DateTime(2012, 1, 22),
"Canada" },
    { 15,         "Apple",    "Fruit",       6946,     new DateTime(2012, 1, 24),
"France" },
};

            //Initialize the WorkBook and fetch the default WorkSheet
            Workbook workbook = new Workbook();
            IWorksheet worksheet = workbook.Worksheets[0];
            // Assigning data to the range
            worksheet.Range["A1:F16"].Value = sourceData;
            // Creating pivot
            var pivotcache = workbook.PivotCaches.Create(worksheet.Range["A1:F16"]);
            var pivottable = worksheet.PivotTables.Add(pivotcache,
worksheet.Range["L7"], "pivottable1");
```

Pivot Table Settings

You can modify the setting of the pivot table created in a spreadsheet by performing the following tasks:

- **Configure pivot table fields**
- **Add field function**
- **Filter pivot table**

- **Managing pivot field level**
- **Refresh pivot table**

Configure pivot table fields

You can configure the fields of your pivot table using the properties and methods of the **IPivotCaches interface (on-line documentation)** and **IPivotTables interface (on-line documentation)**.

Refer to the following example code to configure the pivot table fields in a worksheet.

C#

```csharp
// 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#

```csharp
//Set field amount function
field_Amount.Function = ConsolidationFunction.Average;
```

Filter pivot table

Refer to the following example code to filter a pivot table.

C#

```csharp
var field_product = pivottable.PivotFields[1];
field_product.Orientation = PivotFieldOrientation.RowField;

var field_Amount = pivottable.PivotFields[3];
field_Amount.Orientation = PivotFieldOrientation.DataField;

var field_Country = pivottable.PivotFields[5];
field_Country.Orientation = PivotFieldOrientation.PageField;

//row field filter.
field_product.PivotItems["Apple"].Visible = false;
field_product.PivotItems["Beans"].Visible = false;
field_product.PivotItems["Orange"].Visible = false;

//page filter.
field_Country.PivotItems["United States"].Visible = false;
field_Country.PivotItems["Canada"].Visible = false;
```

Managing pivot field level

Refer to the following example code to manage the field level of a pivot table.

C#

```csharp
//product in level 1.
```

```
var field_product = pivottable.PivotFields["Product"];
field_product.Orientation = PivotFieldOrientation.RowField;

//category in level 2.
var field_category = pivottable.PivotFields["Category"];
field_category.Orientation = PivotFieldOrientation.RowField;

var field_Amount = pivottable.PivotFields[3];
field_Amount.Orientation = PivotFieldOrientation.DataField;

//category will in level 1 and product in level 2.
field_product.Position = 1;
field_category.Position = 0;
```

Refresh pivot table

Refer to the following example code to refresh a pivot table.

C#
```
var field_product = pivottable.PivotFields["Product"];
field_product.Orientation = PivotFieldOrientation.RowField;

var field_Amount = pivottable.PivotFields[3];
field_Amount.Orientation = PivotFieldOrientation.DataField;

//change pivot cache's source data.
worksheet.Range["D8"].Value = 3000;

//sync cache's data to pivot table.
worksheet.PivotTables[0].Refresh();
```

Use Slicer

Spread.Services allows users to add slicer in spreadsheets in order to enable them to perform quick filtration of the data in tables and pivot tables.

Using slicer in a worksheet involves the following tasks:

- Add Slicer in Table
- Add Slicer in Pivot Table
- Use Do Filter Operation
- Apply Slicer Style

Add Slicer in Table

In Spread.Services, you can use slicer in a table by accessing the properties and methods of the **ISlicer interface (on-line documentation)**, **ISlicerCache interface (on-line documentation)**, and **ISlicerCaches interface (on-line documentation)**.

To add slicer in your table, you need to first invoke the **Add method (on-line documentation)** of the ISlicerCaches interface to create a new slicer cache for your table.

Refer to the following example code to add slicer in table.

C#
```
// Defining source data
object[,] sourceData = new object[,] {
    { "Order ID", "Product",  "Category",   "Amount", "Date",
"Country" },
```

```
        { 1,           "Carrots",  "Vegetables",  4270,     new DateTime(2012, 1, 6),
"United States" },
        { 2,           "Broccoli", "Vegetables",  8239,     new DateTime(2012, 1, 7),
"United Kingdom" },
        { 3,           "Banana",   "Fruit",       617,      new DateTime(2012, 1, 8),
"United States" },
        { 4,           "Banana",   "Fruit",       8384,     new DateTime(2012, 1, 10),
"Canada" },
        { 5,           "Beans",    "Vegetables",  2626,     new DateTime(2012, 1, 10),
"Germany" },
        { 6,           "Orange",   "Fruit",       3610,     new DateTime(2012, 1, 11),
"United States" },
        { 7,           "Broccoli", "Vegetables",  9062,     new DateTime(2012, 1, 11),
"Australia" },
        { 8,           "Banana",   "Fruit",       6906,     new DateTime(2012, 1, 16), "New
Zealand" },
        { 9,           "Apple",    "Fruit",       2417,     new DateTime(2012, 1, 16),
"France" },
        { 10,          "Apple",    "Fruit",       7431,     new DateTime(2012, 1, 16),
"Canada" },
        { 11,          "Banana",   "Fruit",       8250,     new DateTime(2012, 1, 16),
"Germany" },
        { 12,          "Broccoli", "Vegetables",  7012,     new DateTime(2012, 1, 18),
"United States" },
        { 13,          "Carrots",  "Vegetables",  1903,     new DateTime(2012, 1, 20),
"Germany" },
        { 14,          "Broccoli", "Vegetables",  2824,     new DateTime(2012, 1, 22),
"Canada" },
        { 15,          "Apple",    "Fruit",       6946,     new DateTime(2012, 1, 24),
"France" },
    };
// Initialize the workbook and fetch the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.Worksheets[0];
// Adding data to the table
worksheet.Range["A1:F16"].Value = sourceData;
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F16"], true);
ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");
// Add slicer for table
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "Category",
200, 200, 100, 200);
ISlicer slicer2 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate2", "Category",
100, 100, 100, 200);
```

Add Slicer in Pivot Table

In Spread.Services, you can use slicer to organize data in pivot table and multi pivot table by accessing the properties and methods of the **IPivotCache Interface (on-line documentation),IPivotCaches Interface (on-line documentation),IPivotField Interface (on-line documentation),IPivotFields Interface (on-line documentation),IPivotTable Interface (on-line documentation),IPivotTables Interface (on-line documentation)IPivotItem Interface (on-line documentation)**.

To add slicer in a pivot table, you need to first invoke the **Add method (on-line documentation)** of the ISlicerCaches interface to create a new slicer cache for your pivot table.

Refer to the following example code to add slicer in a pivot table.

C#

```
// Defining source data
object[,] sourceData = new object[,] {
    { "Order ID", "Product",  "Category",    "Amount", "Date",
```

```
        "Country" },
            { 1,          "Carrots",  "Vegetables",  4270,     new DateTime(2012, 1, 6),
        "United States" },
            { 2,          "Broccoli", "Vegetables",  8239,     new DateTime(2012, 1, 7),
        "United Kingdom" },
            { 3,          "Banana",   "Fruit",       617,      new DateTime(2012, 1, 8),
        "United States" },
            { 4,          "Banana",   "Fruit",       8384,     new DateTime(2012, 1, 10),
        "Canada" },
            { 5,          "Beans",    "Vegetables",  2626,     new DateTime(2012, 1, 10),
        "Germany" },
            { 6,          "Orange",   "Fruit",       3610,     new DateTime(2012, 1, 11),
        "United States" },
            { 7,          "Broccoli", "Vegetables",  9062,     new DateTime(2012, 1, 11),
        "Australia" },
            { 8,          "Banana",   "Fruit",       6906,     new DateTime(2012, 1, 16), "New
        Zealand" },
            { 9,          "Apple",    "Fruit",       2417,     new DateTime(2012, 1, 16),
        "France" },
            { 10,         "Apple",    "Fruit",       7431,     new DateTime(2012, 1, 16),
        "Canada" },
            { 11,         "Banana",   "Fruit",       8250,     new DateTime(2012, 1, 16),
        "Germany" },
            { 12,         "Broccoli", "Vegetables",  7012,     new DateTime(2012, 1, 18),
        "United States" },
            { 13,         "Carrots",  "Vegetables",  1903,     new DateTime(2012, 1, 20),
        "Germany" },
            { 14,         "Broccoli", "Vegetables",  2824,     new DateTime(2012, 1, 22),
        "Canada" },
            { 15,         "Apple",    "Fruit",       6946,     new DateTime(2012, 1, 24),
        "France" },
        };
        // Initialize the workbook and fetch the default worksheet
        Workbook workbook = new Workbook();
        IWorksheet worksheet = workbook.Worksheets[0];
        // Adding data to the pivot table
        worksheet.Range["A1:F16"].Value = sourceData;

        IPivotCache 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#

```csharp
ISlicerCache cache = workbook.SlicerCaches.Add(pivottable1, "Product");
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "p1", "Product", 20,
20, 100, 200);
cache.PivotTables.AddPivotTable(pivottable2);
```

Use Do Filter Operation

You can set slicer filters to analyse bulk information in a spreadsheet quickly and efficiently.

Use slicer do-filter operation

Refer to the following example code to use slicer to perform do-filter operation.

C#

```csharp
// 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#

```csharp
// Adding data to the table
worksheet.Range["A1:F16"].Value = sourceData;
ITable table = worksheet.Tables.Add(worksheet.Range["A1:F16"], true);
ISlicerCache cache = workbook.SlicerCaches.Add(table, "Category", "categoryCache");
// Add slicer for table
ISlicer slicer1 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate1", "Category",
200, 200, 100, 200);
ISlicer slicer2 = cache.Slicers.Add(workbook.Worksheets["Sheet1"], "cate2", "Category",
100, 100, 100, 200);

//do filter operation.
slicer1.SlicerCache.SlicerItems["Vegetables"].Selected = false;

//clear filter.
slicer1.SlicerCache.ClearAllFilters();
```

Apply Slicer Style

When you create a slicer, it is mandatory to create a slicer cache first and then use the slicer cache created base on the column of the table or the pivot table.

The SlicerCaches collection in Spread.Services holds all the slicer caches in the workbook.

Set slicer to built-in style

You can set your slicer to built-in style by using the **Style property (on-line documentation)** of the **ISlicer interface (on-line documentation)**.

Refer to the following example code to set slicer to built-in style.

C#

```csharp
//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 Spread.Services, 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#

```csharp
//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 (on-line documentation)** and **DisplayHeader property (on-line documentation)** of the **ISlicer interface (on-line documentation)**.

Refer to the following example code to modify table layout for slicer style.

```csharp
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;
```

Import and Export .xlsx Document

This section summarizes how Spread.Services handles the spreadsheet documents(.xlsx files).

When you create a workbook using Spread.Services and save it, you automatically export it to an external location or folder. When bringing an Excel file into Spread.Services (importing) and when saving Spread.Services files to an Excel format (exporting), most of the data can be imported or exported successfully. The intention of the import and export capability is to handle as much of the data and formatting of a spreadsheet as possible.

Refer to the following example code to import and export .xlsx document.

```csharp
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");
// Importing Xlsx document
workbook.Open("Source.xlsx");
```

Import and Export JSON Stream

The sole purpose of facilitating users in importing and exporting to and from json stream is to enable them to exchange and organize object data as and when required. This reference summarizes how Spread.Services handles the import and export of json stream using .Net core.

You can export a workbook to a json string/stream using the **ToJson method (on-line documentation)** of the **IWorkbook interface (on-line documentation)**. You can also import a json string or stream to your workbook using the **FromJson method (on-line documentation)** of the IWorkbook interface.

Refer to the following example code to import and export json stream.

```csharp
C#
//ToJson&FromJson can be used in combination with spread.sheets product

//spread.services import an excel file.
//change the path to real source file path.
string source = "savingfile.xlsx";
workbook.Open(source);

//spread.services export to a json string.
```

```
var jsonstr = workbook.ToJson();
//use the json string to initialize spread.sheets product.
//spread.sheets will show the excel file contents.

//spread.sheets product export a json string.
//spread.services use the json string to initialize.
workbook.FromJson(jsonstr);
//spread.services export workbook to an excel file.
//change the path to real export file path.

string export = "export.xlsx";
workbook.Save(export);
```

Apply Conditional Formatting

You can apply conditional formatting in individual cells or a range of cells using rules or conditional operators. The set of conditional formatting rules for a range is represented with the **FormatConditions property (on-line documentation)** of the **IRange interface (on-line documentation)**.

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

- Add Cell Value Rule
- Add Date Occurring Rule
- Add Average Rule
- Add Color Scale Rule
- Add Data Bar Rule
- Add Top Bottom Rule
- Add Unique Rule
- Add Icon Sets Rule
- Add Expression Rule

If you want to delete the formatting rule applied to the cell range in a worksheet, you can do it by using the **Delete method (on-line documentation)** of **IFormatCondition interface (on-line documentation)**

Add Cell Value Rule

The cell value rule compares values entered in the cells with the condition specified in the conditional formatting rule. In order to add a cell value rule, you can use the **Formula1 property (on-line documentation)** and **Formula2 property (on-line documentation)** of the **IFormatCondition interface (on-line documentation)**. You can also use the **Operator property (on-line documentation)** of the IFormatCondition interface to set the operator that will perform the comparison operation, like "Between", "Less Than" etc.

Refer to the following example code to add cell value rule to a range of cells in a worksheet.

C#

```csharp
// Assigning value using object
worksheet.Range["A1:A5"].Value = new object[,]
{
    {1},{3},{5},{7},{9}
};
// Defining format rules.
IFormatCondition condition =
worksheet.Range["A1:A5"].FormatConditions.Add(FormatConditionType.CellValue,
FormatConditionOperator.Between, 1, 5) as IFormatCondition;
condition.NumberFormat = "0.000";
```

Add Date Occurring Rule

The date occurring rule in conditional formatting feature compares the values entered in date format in the cells or a range of cells. This rule can be added using the **DateOperator property (on-line documentation)** of the **IFormatCondition interface (on-line documentation)**.

Refer to the following example code to add date occurring rule to a range of cells in a worksheet.

C#

```csharp
// Adding Date occuring rules
IFormatCondition condition =
worksheet.Range["A1:A4"].FormatConditions.Add(FormatConditionType.TimePeriod) as
IFormatCondition;
condition.DateOperator = TimePeriods.Yesterday;
condition.Interior.Color = Color.FromRGB(128, 0, 128);

DateTime now = DateTime.Today;
worksheet.Range["A1"].Value = now.AddDays(-2);
worksheet.Range["A2"].Value = now.AddDays(-1);
worksheet.Range["A3"].Value = now;
worksheet.Range["A4"].Value = now.AddDays(1);
```

Add Average Rule

The average rule in conditional formatting can be added and deleted using the properties and methods of the **IAboveAverage interface (on-line documentation)**.

Refer to the following example code to add average rule to a range of cells in a worksheet.

C#

```csharp
// Adding average rule
worksheet.Range["A1"].Value = 1;
worksheet.Range["A2"].Value = 2;
worksheet.Range["A3"].Value = 3;
worksheet.Range["A4"].Value = 4;
worksheet.Range["A5"].Value = 60000000;

IAboveAverage averageCondition =
worksheet.Range["A1:A5"].FormatConditions.AddAboveAverage();
averageCondition.AboveBelow = AboveBelow.AboveAverage;
averageCondition.NumStdDev = 2;
averageCondition.NumberFormat = "0.00";
```

Add Color Scale Rule

The color scale rule uses a sliding color scale to format cells or a range of cells. For instance, if numeric cell value 1 is represented with color yellow and 50 with green, then 25 would be light green. This rule can be added using the properties and methods of the **IColorScale interface (on-line documentation)**.

Refer to the following example code to add color scale rule to a cell range in a worksheet.

C#

```csharp
// 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.FromRGB(255, 0, 0);

twoColorScaleRule.ColorScaleCriteria[1].Type = ConditionValueTypes.Number;
twoColorScaleRule.ColorScaleCriteria[1].Value = 5;
twoColorScaleRule.ColorScaleCriteria[1].FormatColor.Color = Color.FromRGB(0, 255, 0);
```

Add Data Bar Rule

The data bar rule in conditional formatting displays a bar in the cell on the basis of cell values entered in a range. This rule can be added using the properties and methods of the **IDataBar interface (on-line documentation)**.

Refer to the following example code to add data bar rule to a range of cells in a worksheet.

C#

```csharp
// 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.FromRGB(128, 0, 212);
dataBar.NegativeBarFormat.ColorType = DataBarNegativeColorType.Color;
dataBar.NegativeBarFormat.Color.Color = Color.FromRGB(128, 0, 240);
dataBar.ShowValue = false;
```

Add Top Bottom Rule

The top bottom rule checks whether the values in the top or bottom of a cell range match with the required values in the cell. In case the values don't match, the data is considered as invalid. This rule can be added using the properties and methods of the **ITop10 interface (on-line documentation)**.

The following options are available while adding top bottom rule in a worksheet:

- Top 10
- Top 10%
- Bottom 10
- Bottom 10%

- Above Average
- Below Average

Refer to the following example code to add top bottom rule in a worksheet.

C#

```csharp
// 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.FromRGB(128, 0, 128);
```

Add Unique Rule

The unique rule in conditional formatting is applied to check whether the value entered in a cell is a unique value in that particular range. This is possible only when the duplication option is set to false. To check for the duplicate values, the duplicate rule is applied separately.

Unique rule can be added using the properties and methods of the **IUniqueValues interface (on-line documentation)**.

Refer to the following example code to add unique rule in a worksheet.

C#

```csharp
// Adding Unique Rule
worksheet.Range["A1:A5"].Value = new object[,]
{
    {1},
    {2},
    {1},
    {3},
    {4}
};

IUniqueValues condition2 = worksheet.Range["A1:A5"].FormatConditions.AddUniqueValues();
condition2.DupeUnique = DupeUnique.Unique;
condition2.Font.Name = "Arial";
```

Add Icon Sets Rule

The icon sets rule in conditional formatting displays the icons on the basis of values entered in the cells. Each value represents a distinct icon that appears in a cell if it matches the icon sets rule applied on it. This rule can be added using the properties and methods of the **IIconSetCondition interface (on-line documentation)**.

Refer to the following example code to add icon sets rule in a worksheet.

C#

```csharp
// Adding IconSets rule
IIconSetCondition condition =
```

```
worksheet.Range["A1:A5"].FormatConditions.AddIconSetCondition();
condition.IconSet = workbook.IconSets[IconSetType.Icon3Symbols];
condition.IconCriteria[1].Operator = FormatConditionOperator.GreaterEqual;
condition.IconCriteria[1].Value = 50;
condition.IconCriteria[1].Type = ConditionValueTypes.Percent;
condition.IconCriteria[2].Operator = FormatConditionOperator.GreaterEqual;
condition.IconCriteria[2].Value = 70;
condition.IconCriteria[2].Type = ConditionValueTypes.Percent;

worksheet.Range["A1"].Value = 1;
worksheet.Range["A2"].Value = 2;
worksheet.Range["A3"].Value = 3;
worksheet.Range["A4"].Value = 4;
worksheet.Range["A5"].Value = 5;
```

Add Expression Rule

The expression rule in conditional formatting is used to set the expression rule's formula. This rule can be added using the properties and methods of the **IFormatCondition interface (on-line documentation)**.

Refer to the following example code to add expression rule in a worksheet.

C#

```
// Adding Expression Rule
worksheet.Range["A1:B5"].Value = new object[,]
{
    {1, 2},
    {0, 1},
    {0, 0},
    {0, 3},
    {4, 5}
};
IFormatCondition condition =
worksheet.Range["B1:B5"].FormatConditions.Add(FormatConditionType.Expression, 0, "=A1")
as IFormatCondition;
condition.Interior.Color = Color.FromRGB(255, 0, 0);
```

Apply Data Validations

You can use the data validation feature in Spread.Services to ensure users enter only the valid values into a cell while working in a spreadsheet.

For instance, let's say you have a worksheet where you want users to enter only whole numbers between 1 to 15. To accomplish this, you can create a data validation rule that restricts users to enter cell values other than a whole number between 1 to 15. You can even create custom dropdown lists to specify the possible values that can be entered in the cells or display messages or error alerts to validate the data and get notified if there is something wrong with the information entered in the worksheets.

Applying data validations in worksheets involves the following tasks.

- Add Validations
- Modify Validations
- Delete Validation

Add Validations

You can use the **Add method (on-line documentation)** of the **IValidation interface (on-line documentation)** to apply data validation to individual cells or a range of cells in a spreadsheet. A single cell can

have only one validation rule and if you try to apply validation on a cell that already possesses a validation rule, it will throw an exception.

Validation rule instance for a range is represented with the **Validation property (on-line documentation)** of the **IRange interface (on-line documentation)**. If you want to know whether a cell range already contains the validation rule, you can use the **HasValidation property (on-line documentation)** of the IRange interface. If all the cells in a range possess the same validation rule applied to them, it is represented with the **ValidationIsSame property (on-line documentation)** of the IRange interface.

Shared below is a list of data validations operations that can be implemented in Spread.Services.

- **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";
```

```csharp
//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#

```csharp
//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#

```csharp
//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#

```csharp
//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#

```csharp
//Add custom validation
worksheet.Range["A2"].Value = 1;
worksheet.Range["A3"].Value = 2;
worksheet.Range["C2"].Value = 1;
//when use custom validation, validationOperator and formula2 parameters will be
ignored even if you have given.
worksheet.Range["A2:A3"].Validation.Add(ValidationType.Custom,
ValidationAlertStyle.Information, formula1: "=C2");
```

Delete Validation

You can delete the applied validation rule using the **Delete method (on-line documentation)** of the **IValidation interface (on-line documentation)**.

Refer to the following example code to know how you can delete validation rule applied to a cell or a range of cells in a worksheet.

C#

```csharp
//Add validation
worksheet.Range["A1:A3"].Validation.Add(ValidationType.Whole,
ValidationAlertStyle.Stop, ValidationOperator.Between, 1, 8);
worksheet.Range["B1:B3"].Validation.Add(ValidationType.Whole,
ValidationAlertStyle.Stop, ValidationOperator.Between, 11, 18);

//Delete validation.
worksheet.Range["A1:A2"].Validation.Delete();
```

Modify Validation

You can change the validation rule for a range by using either of the two ways described below:

- Set properties of the **IValidation interface (on-line documentation)** (**Type property (on-line documentation)**, **Formula1 property (on-line documentation)**, **Formula2 property (on-line documentation)**, and many more).
- Use **Delete method (on-line documentation)** of the IValidation interface to first delete validation rule and then use the **Add method (on-line documentation)** to add the new rule.

Refer to the following example code to know how you can modify an existing validation rule applied to a cell or a range of cells in a worksheet.

C#

```csharp
//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();
```

Assembly Reference

The complete Spread.Services 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
Grapecity.Documents.Spread ('GrapeCity.Documents.Spread Assembly' in the on-line documentation)	Provides the namespaces for the Spread.Services component functionality for .Net Core.

For help with using the product, refer to the [Developer's Guide](#).

For a complete list of documentation, refer to the [Spread.Services Product Documentation](#).

Index