

## Table of Contents

Table of Contents	1-7
GcExcel Java Overview	8
Key Features	9-10
Getting Started	11-12
Quick Start	12-14
License Information	14-16
Technical Support	16
Contacting Sales	17
Redistribution	17
End User License Agreement	17
Features	18
Worksheet	18-19
Work with Worksheets	19-24
Range Operations	24
Access a Range	24-25
Access Areas in a Range	25-26
Access Cells, Rows and Columns in a Range	26-27
Cut or Copy Cell Ranges	27-31
Cut or Copy Shape, Slicer, Chart and Picture	31-33
Find and Replace Data	33-35
Get Row and Column Count	35
Hide Rows and Columns	35-36
Insert And Delete Cell Ranges	36-37
Insert and Delete Rows and Columns	37-39
Merge Cells	39
Set Values to a Range	39-40
Set Row Height and Column Width	40
Auto Fit Row Height and Column Width	40-41
Work with Used Range	41-42
Freeze Panes in a Worksheet	43-44
Freeze Trailing Panes in a Worksheet	44-45
Customize Worksheets	45-47

Worksheet Views	47-51
Cell Types	51-53
Quote Prefix	53-54
Tags	54-56
Rich Text	56-61
Workbook	61
Create Workbook	61
Open and Save Workbook	62-63
Protect Workbook	63-66
Cut or Copy Across Sheets	66
Enable or Disable Calculation Engine	66-67
Workbook Views	67-69
Comments	69-71
Hyperlinks	71-73
Sort	73-77
Filter	77-79
Group	79
Create Row or Column Group	79-81
Remove a Group	81-82
Summary Row	82
Outline Subtotals	82-84
Outline Column	84-88
Conditional Formatting	88-89
Cell Value Rule	89
Date Occurring Rule	89-90
Average Rule	90
Color Scale Rule	90-91
Data Bar Rule	91-92
Top Bottom Rule	92
Unique Rule	93
Icon Sets Rule	93-94
Expression Rule	94
Data Validations	94-95

Add Validations	95-97
Delete Validation	97-98
Modify Validation	98
Data Binding	98-103
Digital Signatures	103-115
Formulas	115-116
Formula Functions	116-134
Set Formula to Range	134-135
Set Table Formula	135-137
Set Array Formula	137-138
Precedents and Dependents	138-141
Custom Functions	141-147
Shapes And Pictures	147-151
Customize Shape Format and Shape Text	151-155
Hyperlink on Shape	155-157
Group or Ungroup Shapes	157-159
Shape Adjustment	159-160
Background Image	160-161
Size and Position of Image	161-162
Image Transparency	162
Control Position of Overlapping Shapes	162-163
Styles	163-165
Set Sheet Styling	165-168
Create and Set Custom Named Style	168-170
Theme	170-171
Chart	171-172
Create and Delete Chart	172-173
Configure Chart	173
Chart Title	173-175
Chart Area	175-176
Plot Area	176
Customize Chart Objects	176-177
Series	177-181

Configure Chart Series	181-187
Error Bars	187-191
Walls	191-192
Axis and Other Lines	192-195
Configure Chart Axis	195-198
Floor	198
Data Label	198-200
Legends	200-201
Chart Types	201-203
Area Chart	203-205
Bar Chart	205-207
Column Chart	207-209
Combo Chart	209-211
Line Chart	211-213
Pie Chart	213-214
Stock Chart	214-216
Surface Chart	216-217
XY (Scatter) chart	217-219
Radar Chart	219-221
Statistical Chart	221
Box Whisker	221-222
Histogram	222-223
Waterfall Chart	223-225
Pareto Chart	225-226
Specialized Chart	226
Sunburst	226-227
Treemap	227-228
Funnel	228-229
Chart Sheet	229-232
Table	232
Create and Delete Tables	232-233
Modify Tables	233-235
Table Sort	235-236

Table Filters	236
Add and Delete Table Columns and Rows	236-237
Table Style	237-238
Modify Table with Custom Style	238
Modify Table Layout	238-239
Pivot Table	239-240
Create Pivot Table	240-241
Pivot Table Settings	241-250
Pivot Table Style	250-256
Sparkline	256-259
Slicer	259-260
Add Slicer in Table	260-261
Add Slicer in Pivot Table	261-263
Slicer Style	263
Modify Slicer with Custom Style	264
Modify Table Layout for Slicer Style	264-265
Auto-Filter Table with Slicer	265-266
Configure Slicer Layout	266-267
Cut or Copy Slicer	267-269
Duplicate Slicer	269-270
Use Do Filter Operation	270-271
Print Settings	271-272
Configure Page Header and Footer	272-273
Configure Page Settings	273-274
Configure Page Breaks	274-276
Configure Paper Settings	276-277
Configure Print Area	277
Configure Columns to Repeat at Left and Right	277-278
Configure Rows to Repeat at Top and Bottom	278-279
configure-drafts	279-280
Configure Print Page Range	280-281
Configure Sheet Print Settings	281-282
Templates	283-286

Template Configuration	286-287
Template Fields	287-291
Template Properties	291-298
Cell Expansion	298
Cell Context	298-302
Conditional Formatting	302-303
Global Settings	303-307
Fixed Layout	307-310
PDF Form Builder	310-322
Charts	322-327
Tables	327-329
Sparklines	329-331
Data Source Binding	331-332
Create Excel Report using Template	332-338
File Operations	339
Import and Export .xlsx Document	339-341
Export to a PDF File	341-342
Configure Fonts and Set Style	342-343
Export Pivot Table Styles And Format	343-346
Export Shapes	346-347
Export Borders	347-349
Export Conditional Formatting	349-350
Control Pagination	350-351
Render Excel Range Inside PDF	351-354
Export Multiple Sheets To One Page	354-356
Keep Rows Together Over Page Breaks	356-357
Delete Blank Pages From Middle	357-358
Export Different Headers On Different Pages	358-359
Export Last Page Without Headers	359-360
Export Custom Page Information	360-361
Export Specific Pages to PDF	361-362
Save Multiple Workbooks to Single PDF	362-365
Export Worksheet to PDF	365-367

Export Fills	367
Export Picture	367-368
Export Sparkline	368-369
Export Table	369-370
Export Text	370-373
Export Vertical Text	373-374
Shrink To Fit With Text Wrap	374-375
Export Slicers	375-376
Export Signature Lines	376
Support Security Options	376-379
Support Document Properties	379
Adjust Column Width and Row Height	379-380
Support Sheet Background Image	380-381
Support Background Color Transparency	381-382
Export to an HTML File	382-385
Working With Page Setup	385-389
Import and Export CSV File	389-391
Import and Export CSV Files with Delimiters	391-393
Import and Export JSON Stream	393-397
Import and Export Macros	397-398
Import and Export OLE Objects	398-399
Convert to Image	399-401
Configure JDK 8 Date Time API	402-403
Release Notes	404
Release Notes for Version 3.2.0	404
Release Notes for Version 3.1.0	404-405
Release Notes for Version 3.0.0	405-406
Release Notes for Version 2.2.0	406-407
Release Notes for Version 2.1.0	407-408
Index	409-415

## GcExcel Java Overview

Thank you for choosing **GrapeCity Documents for Excel, Java Edition**.

GcExcel Java is a high-performance spreadsheet component that comes packaged with all the necessary features to help users handle complex spreadsheet challenges in an efficient way. This product can be used with Java Web Applications and Java Desktop Applications, as well as can be deployed on cloud platforms.

Developed for use in Java programming, this component provides developers with a comprehensive API to allow them to quickly create, manipulate, convert, and share Microsoft Excel-compatible spreadsheets that are accessible from nearly any application, platform or IDE. It targets varied platforms including Enterprise Web Applications, Linux, Unix and Windows; thus serving as the one-stop solution for all your spreadsheet requirements.

GcExcel Java is outstanding in a way that it possesses the ability to model its interface-based Java API on the document object model of Excel. This not only makes it easier for users to import worksheets, perform calculations on the data, run customized queries and generate custom outputs but also allows them to export complex spreadsheet scenarios and work with cross sheet references as and when desired.

### What GcExcel Java Offers You

- Facilitates server-side spreadsheet generation, manipulation, and serialization.
- Requires low memory footprint.
- Robust calculation engine.
- Produces output in varied formats including .xlsx, pdf and ssjson.
- Deploys on Desktop, Mobile, Web applications, Application Services, Azure functions, AWS Lambda

For API reference details, the following documentation is available:

- **Java API Documentation (on-line documentation)**

For a comprehensive list of available features, refer to:

- [Features](#)



## Key Features

With a set of class libraries, collections, interfaces, methods and fields that comes packaged with GcExcel Java; developers can build everything right from scratch while working with spreadsheets for increased productivity and improved data analysis.

GcExcel Java assists developers in creating powerful spreadsheets while working with Java desktop and web applications with the following key features:

- **Lightweight API Architecture for Enhanced Efficiency**

GcExcel Java possesses an extremely lightweight API architecture that allows users to save significant amount of time, storage space and development efforts while also maximizing the overall efficiency of generating, loading, editing, exporting and converting spreadsheets.

- **Flexible Themes and Components**

Using GcExcel Java, users can apply custom themes, configure components, summarise data, set custom styles, embed drawing objects, apply cell formatting and integrate a robust calculation engine while working with spreadsheets.

- **Seamless Excel Compatibility**

GcExcel Java is fully compatible with MS Excel. With this product, users can insert pivot tables, include comments, add charts, shapes, pictures, slicers, sparklines and tables, apply conditional formatting, data validation, filters and formulas, etc.

- **Extensive Support for Major Operating Systems**

You can deploy the Java desktop and web applications created using GcExcel Java on all major operating systems including Microsoft Windows, Linux and macOS.

- **Based on Excel Object Model**

With the interface-based Java API model, you can import data, calculate formulas, query, generate, and save worksheets with complex scenarios as per custom preferences.

- **No Dependency on MS Excel**

In order to work with GcExcel Java, users don't need to install MS Office Suite and access MS Excel on their systems.

- **Use Built-in Templates for Simple Forms**

With the help of built-in templates, you can quickly create simple forms like invoice etc. while working with spreadsheets.

- **Create Interactive Experience with SpreadJS Sheets**

GcExcel Java provides a completely interactive and user-friendly spreadsheet experience when used concurrently with SpreadJS.

- **Workbook and Worksheets**

You can create workbook and add worksheets while also performing the import and export operations. Further, you can activate worksheets, configure its display, delete it and protect it from modification or encrypt it with a password.

- **Formulas and Functions**

With extensive support for implementing formulas, adding custom functions and choosing from 450+ built-in

functions, users can execute complex spreadsheet calculations without any hassle.

- **Pivot and Excel Tables**

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

- **Export to PDF**

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

- **Deploy Apps with Excel Spreadsheets to the Cloud**

With GcExcel, you can apply cloud based deployments and deploy your applications on Azure, AWS Lambda and Google Cloud Java.

- **Shapes and Pictures**

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

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

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

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

## Getting Started

### System Requirements

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

- Java Development Kit (JDK) 6.0 or higher.
- Any Java IDE (Eclipse, IntelliJ etc.) , Gradle or Maven.

For OS versions supported in GcExcel Java, refer to [System Requirements](#).

### Setting up an application

GcExcel Java API reference is available through [Maven Central Repository](#), a directory that stores all the java archives (.jar files) and adds libraries, plugins and references to your project.

For installation of the product, refer to the following steps:

- **Step 1 - Download the GcExcel Java package**
- **Step 2 - Install the GcExcel Java package and add dependency for GcExcel library**

#### Step 1 - Download the GcExcel Java package

You can either download the GcExcel java package (gcexcel-3.2.0.jar) from [Maven Central repository](#) or download it locally on your machine from [GrapeCity website](#).

#### Step 2 - Install the GcExcel Java package and add dependency for GcExcel library

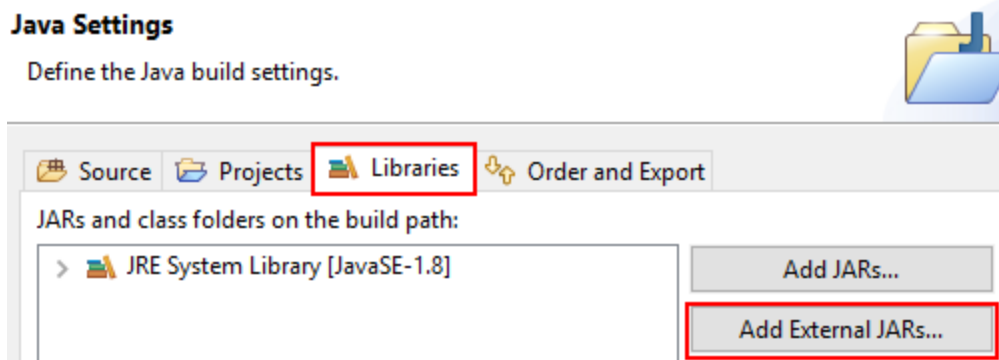
Complete the following steps to install the GcExcel Java package (gcexcel-3.2.0.jar) and add dependency for GcExcel Java library in your application.

**If you are creating an application :**

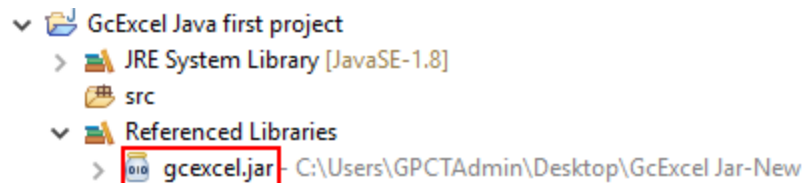
##### A. Using Java Integrated Development Environment (IDE) :

You can install any Java IDE - Eclipse or IntelliJ as per your choice. Shared below are the steps to create a Java application using Eclipse IDE:

1. Open the Eclipse IDE.
2. Create a new Java project.
3. In **Project name** field, enter the name of your project and click Next.
4. In **Java settings**, under **Libraries** tab, click **Add External JARs..**



5. Select the gcexcel-3.2.0.jar to add it to your project.
6. Click Finish.
7. The jar file will be added under the **Referenced Libraries** in your project.



## B. Using the Gradle project:

Open the build.gradle and append the following script in the dependencies block:


```
compile("com.grapecity.documents:gcexcel:3.2.0")
```

## C. Using the Maven project:

Open the pom.xml and add below xml element in the dependencies node.

```
<dependency>
  <groupId>com.grapecity.documents</groupId>
  <artifactId>gcexcel</artifactId>
  <version>3.2.0</version>
</dependency>
```

The jar file will be added as a library in the project and your project can now reference all classes of GcExcel in the jar file.

 **Note:** The complete list of GcExcel Java dependencies can be downloaded from [here](#) ('GcExcelJava\_Dependencies.docx' in the on-line documentation).

## Quick Start

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

- **Step 1: Create a Java project and add dependency for GcExcel library**
- **Step 2: Add dependency for processing JSON**

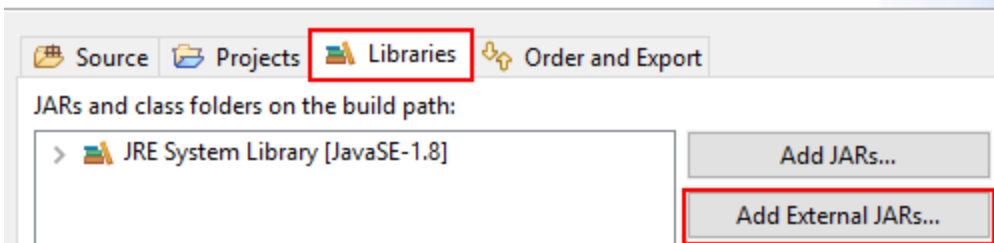
- **Step 3: Create a basic application with GcExcel Java**
- **Step 4: Build and Run the Project**

## Step 1: Create a Java Project and add dependency for GcExcel library

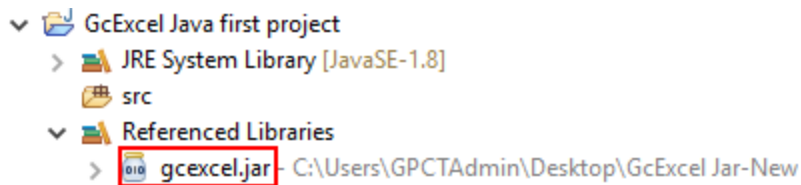
1. In Eclipse IDE, select **File | New | Java Project** to create a new Java project.
2. In **Project name** field, enter the name of your project and click Next.
3. In **Java settings**, under **Libraries** tab, click **Add External JARs..**

### Java Settings

Define the Java build settings.



4. Select the gcexcel-3.2.0.jar to add it to your project.
5. Click Finish.
6. The jar file will be added under the **Referenced Libraries** in your project.



## Step 2 - Add dependency for processing JSON

Download the [JSON Processing Default Provider](#) and add reference to javax.json as dependency library in your Java project.

If you're using javax.json-1.1 and above, make sure you add the following two files in your java project in order to license it successfully -:

javax.json-1.1.x.jar

javax.json.api-1.1.x.jar

## Step 3 - Create a basic application with GcExcel Java

To create a basic application with GcExcel Java, refer to the following tasks:

### 1. Add Namespaces

In Main.java, import the following namespaces -

```
import com.grapecity.documents.excel.*;
import com.grapecity.documents.excel.drawing.*;
```

## 2. Create a new workbook, access the default worksheet and configure settings

Create a new workbook, access the default worksheet. You can also configure settings like the default row height and column width of the worksheet. An example code is shown below:

```
Workbook workbook = new Workbook();  
IWorksheet worksheet = workbook.getWorksheets().get(0);  
worksheet.setStandardHeight(20);  
worksheet.setStandardWidth(50);
```

## 3. Save the workbook

After customizing the worksheet, you can save the workbook with the desired name and provide the required path. An example code is shown below:

```
workbook.save("GcExcelFeatures.xlsx");
```

## Step 4 - Build and Run the Project

When you finish, build and run your project. You will notice that the GcExcelFeatures.xlsx file is created at the specified location on your system.

## License Information

### Types of Licenses

GcExcel Java supports the following types of license:

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

#### Unlicensed

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

#### Maximum time of opening and saving Excel files

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

- 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 excel files, and trying to save the 101th file, an Excel file with just a watermark sheet will be saved. The content of watermark tells users that no license is found.

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

## Maximum Operating Time

While executing an application program, the duration of operating GcExcel Java 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.getWorksheets().add()	Returns null.

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

## Watermark Sheet

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

When saving a PDF file, a PDF file with a watermark on the top of each exported page will be added. The content of the watermark will tell users which license is applied and will provide our sales and contact information.

The following watermark will be displayed:

*"Unlicensed copy of GrapeCity Documents for Excel, Java Edition. Contact [us.sales@grapecity.com](mailto:us.sales@grapecity.com) to get your 30-day evaluation key to remove this text and other limitations".*

## Evaluation License

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

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

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

- **Cannot create new instance**

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

- **Open and Save Excel Files**

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

- **Save PDF Files**

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

- **API Limitations**

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

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

- **Watermark**

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

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

*"Expired Evaluation copy of GrapeCity Documents for Excel, Java Edition. Contact [us.sales@grapecity.com](mailto:us.sales@grapecity.com) to purchase license".*

## Licensed

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

### Watermark Sheet

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

## Apply License To GcExcel

To apply evaluation/production license in GcExcel Java, the long string key needs to be copied to the code in one of the following two ways.

- **To license all the workbooks in a project**, add the license by using SetLicenseKey method. This will license all the workbooks.

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

- **To license an instance of the workbook**, add the license key when an instance of workbook is created. Add the following code:

```
Main.java
var workbook = new Workbook("Your License Key");
```

## Technical Support

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

- Product Forum: <https://www.grapecity.com/forums>
- Email: [us.sales@grapecity.com](mailto:us.sales@grapecity.com)



## Contacting Sales

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

World Wide Web site	<a href="https://www.grapecity.com/">https://www.grapecity.com/</a>
E-mail	<a href="mailto:us.sales@grapecity.com">us.sales@grapecity.com</a>
Phone	(800) 858-2739 or (412) 681-4343 outside the U.S.A.
Fax	(412) 681-4384

## Redistribution

In order to deploy GcExcel Java, you need to ensure that you have the following installed on your system:

- Java Development Kit (JDK) 6.0 or higher.

In order to distribute the application, make sure you meet the installation criteria specified in the [System Requirements](#) in this documentation. Further, the users also 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	<a href="https://www.grapecity.com/">https://www.grapecity.com/</a>
E-mail	<a href="mailto:us.sales@grapecity.com">us.sales@grapecity.com</a>
Phone	(800) 858-2739 or (412) 681-4343 outside the U.S.A.
Fax	(412) 681-4384

## End User License Agreement

The GrapeCity licensing information, including the GrapeCity end-user license agreement, frequently asked licensing questions, and the GrapeCity licensing model, is available online. For detailed information on licensing, refer to [GrapeCity Licensing](#). For GrapeCity end-user license agreement, refer to [End-User License Agreement For GrapeCity Software](#).

## Features

This section comprises the features available in GcExcel.

### Worksheet

Work with cells, range and basic worksheet operations.

### Workbook

Work with basic workbook operations.

### Comments

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

### Hyperlinks

Add, configure or delete hyperlinks.

### Sort

Apply sorting and its various types.

### Filter

Apply filtering and its several types.

### Group

Apply grouping over data in rows or columns.

### Conditional Formatting

Apply conditional formatting in a cell or range of cells.

### Data Validations

Add, modify and delete data validations.

### Data Binding

Bind data to sheet, cell or table columns.

### Formulas

Use formulas to carry complex calculations.

### Custom Functions

Create custom functions to implement custom arithmetic logic.

### Shapes and Pictures

Work with shapes and pictures in a worksheet.

### Styles

Set styles to format cell appearance.

### Theme

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

### Chart

Work with charts to display data graphically.

### Table

Use tables to organize large amount of data efficiently.

### Pivot Table

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

### Sparkline

Use sparklines to insert graphical illustration of trends in data.

### Slicer

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

### Print Settings

Configure print settings to manage printing options.

## Worksheet

A worksheet refers to 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 (representing numeric characters like 1,2,3 etc.) and columns (representing alphabetical letters like A,B,C etc.). For instance, in a worksheet, B3 represents the cell in column B and row 3.

In GcExcel Java, the **IWorksheets** ('IWorksheets Interface' in the on-line documentation) interface stores the collection of all the sheets in the workbook and the **IWorksheet** ('IWorksheet Interface' in the on-line documentation) interface represents a particular worksheet.

You can use the methods of the IWorksheets interface and IWorksheet interface to execute important tasks in a spreadsheet including insertion of a new worksheet in the workbook, deletion of a worksheet from the collection, assigning an active sheet, executing range operations and so much more.

Managing a worksheet involves the following tasks:

- [Work with Worksheets](#)
- [Range Operations](#)
- [Freeze Panes in a Worksheet](#)
- [Customize Worksheets](#)
- [Worksheet Views](#)
- [Cell Types](#)
- [Quote Prefix](#)
- [Tags](#)
- [Rich Text](#)

## Work with Worksheets

While managing worksheets, you can execute the following operations to accomplish essential spreadsheet tasks.

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

### Access the default worksheet

By default, an empty worksheet with the name **Sheet1** is automatically added in the workbook when a new workbook is created. For every workbook, only one default worksheet is added.

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

Java

```
// Fetch the default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
```

### Add multiple worksheets

You can add one more worksheets before or after a specific sheet in the workbook.

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

## Java

```
// Add a worksheet to the workbook.
IWorksheet worksheet1 = workbook.getWorksheets().add();

// Add a new worksheet before worksheet1 and reset its name
IWorksheet worksheet2 = workbook.getWorksheets().addBefore(worksheet1);
worksheet2.setName("MySheet2");

// Add a sheet after worksheet2
workbook.getWorksheets().addAfter(workbook.getWorksheets().get(1));
```

## Activate a worksheet

In a workbook with multiple worksheets, you may want to set the current sheet or any particular worksheet as workbook's active sheet. This can be done using the **activate** ('activate Method' in the on-line documentation) method of the **IWorksheet** ('IWorksheet Interface' in the on-line documentation) interface.

Refer to the following example code in order to activate a worksheet in a workbook.

## Java

```
IWorksheet worksheet4 = workbook.getWorksheets().add();

// Activate the newly created sheet
worksheet4.activate();
```

## Access a worksheet

A workbook stores all the worksheets in the **Worksheets** collection.

In order to access a particular worksheet within a workbook, refer to the following example code.

## Java

```
// Accessing a worksheet using sheet index.
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Accessing a worksheet using sheet name as "Sheet1".
IWorksheet worksheet1 = workbook.getWorksheets().get("Sheet1");
```

## Protect a worksheet

In order to ensure security and integrity of the data in the workbook, GcExcel Java enables users to protect worksheets via converting it into a read-only sheet. A worksheet can be prevented from modification either by using a password or without it.

## Protect worksheet from modification without password

The **IProtectionSettings** ('IProtectionSettings Interface' in the on-line documentation) interface provides the

methods to explicitly configure the protection settings in a worksheet. In case you want to remove protection, you can unprotect your worksheet by setting the protection field to false.

To protect or unprotect a worksheet in GcExcel Java, refer to the following example code.

Java

```
// Protect Worksheet
worksheet.setProtection(true);
worksheet.getProtectionSettings().setAllowInsertingColumns(true);

// Unprotect worksheet
IWorksheet worksheet1 = workbook.getWorksheets().add();
worksheet1.setProtection(false);
```

## Protect worksheet from modification using password

A worksheet can be made password protected to restrict modification by using the **Protect** ('**protect Method**' in the **on-line documentation**) method of **IWorksheet** interface. The password is a case sensitive string which can be passed as a parameter to the **Protect** method.

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

Java

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Data
Object data = new Object[][] { { "Name", "City", "Sex", "Weight", "Height", "Age" },
    { "Bob", "NewYork", "male", 80, 180, 56 }, { "Betty", "NewYork", "female", 72,
168, 45 },
    { "Gary", "NewYork", "male", 71, 179, 50 }, { "Hunk", "Washington", "male", 80,
171, 59 },
    { "Cherry", "Washington", "female", 58, 161, 34 }, { "Coco", "Virginia",
"female", 58, 181, 45 },
    { "Lance", "Chicago", "female", 49, 160, 57 }, { "Eva", "Washington", "female",
71, 180, 81 } };

// Set data
worksheet.getRange("A1:G9").setValue(data);
//Protects the workbook with password so that other users cannot view hidden worksheets,
add, move, delete, hide, or rename worksheets.
worksheet.protect("Ygs_87@ytr");
// Save workbook to xlsx
workbook.save("ProtectWorksheet.xlsx", SaveFileFormat.Xlsx);
```

A password protected worksheet can be unprotected by using the **Unprotect** method of **IWorksheet** interface. The correct password (password set in **Protect** method) needs to be passed as a parameter to the **Unprotect** method. In case,

the password is omitted or an incorrect password is passed, an exception message "Invalid Password" is thrown. Refer to the following example code to unprotect a worksheet from modification using password.

## Java

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Data
Object data = new Object[][] { { "Name", "City", "Sex", "Weight", "Height", "Age" },
    { "Bob", "NewYork", "male", 80, 180, 56 }, { "Betty", "NewYork", "female", 72,
168, 45 },
    { "Gary", "NewYork", "male", 71, 179, 50 }, { "Hunk", "Washington", "male", 80,
171, 59 },
    { "Cherry", "Washington", "female", 58, 161, 34 }, { "Coco", "Virginia",
"female", 58, 181, 45 },
    { "Lance", "Chicago", "female", 49, 160, 57 }, { "Eva", "Washington", "female",
71, 180, 81 } };
// Set data
worksheet.getRange("A1:G9").setValue(data);
worksheet.protect("Ygs_87@ytr");
//Removes the above protection from the workbook.
worksheet.unprotect("Ygs_87@ytr");
// Save workbook to xlsx
workbook.save("UnProtectWorksheet.xlsx", SaveFileFormat.Xlsx);
```

## Delete Worksheet

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

To delete a specific sheet from the workbook, refer to the following example code.

## Java

```
IWorksheet worksheet5 = workbook.getWorksheets().add();

// Workbook must contain at least one visible worksheet, if delete the one visible
worksheet, it will throw exception.
worksheet5.delete();
```

## Copy and Move Worksheet

You can copy the current spreadsheet on which you're working as well as copy a worksheet between workbooks and then move them to a specific location as per your custom requirements and preferences. This can be done by using the **copy()** ('copy Method' in the on-line documentation) method, the **copyAfter()** ('copyAfter Method' in the on-line documentation) method, the **copyBefore()** ('copyBefore Method' in the on-line documentation) method, the **move()** ('move Method' in the on-line documentation) method, the **moveBefore()** ('moveBefore Method' in the on-line documentation) method and the **moveAfter()** ('moveAfter Method' in the on-line documentation) method of

the **IWorksheet** ('IWorksheet Interface' in the on-line documentation) interface. Using these methods, the worksheet can easily be copied and relocated by placing it within the same workbook or another workbook as and when you want.

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

## Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
Object data = new Object[][] {
{ "Name", "City", "Birthday", "Sex", "Weight", "Height", "Age" },
{ "Bob", "newyork", new GregorianCalendar(1968, 6, 8), "male", 80, 180, 56 },
{ "Betty", "newyork", new GregorianCalendar(1972, 7, 3), "female", 72, 168, 45 },
{ "Gary", "NewYork", new GregorianCalendar(1964, 3, 2), "male", 71, 179, 50 },
{ "Hunk", "Washington", new GregorianCalendar(1972, 8, 8), "male", 80, 171, 59 },
{ "Cherry", "Washington", new GregorianCalendar(1986, 2, 2), "female", 58, 161, 34 },
{ "Coco", "Virginia", new GregorianCalendar(1982, 12, 12), "female", 58, 181, 45 },
{ "Lance", "Chicago", new GregorianCalendar(1962, 3, 12), "female", 49, 160, 57 },
{ "Eva", "Washington", new GregorianCalendar(1993, 2, 5), "female", 71, 180, 81 } };

// Set data
worksheet.getRange("A1:G9").setValue(data);

// Copy the active sheet to the end of current workbook
IWorksheet copy_worksheet = worksheet.copy();
copy_worksheet.setName("Copy of " + worksheet.getName());

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

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

## Java

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

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

// Fetch the active worksheet
IWorksheet worksheet = source_workbook.getActiveSheet();
Object data = new Object[][] {
{ "Name", "City", "Birthday", "Sex", "Weight", "Height", "Age" },
{ "Bob", "newyork", new GregorianCalendar(1968, 6, 8), "male", 80, 180, 56 },
{ "Betty", "newyork", new GregorianCalendar(1972, 7, 3), "female", 72, 168, 45 },
{ "Gary", "NewYork", new GregorianCalendar(1964, 3, 2), "male", 71, 179, 50 },
```

```
{ "Hunk", "Washington", new GregorianCalendar(1972, 8, 8), "male", 80, 171, 59 },
{ "Cherry", "Washington", new GregorianCalendar(1986, 2, 2), "female", 58, 161, 34 },
{ "Coco", "Virginia", new GregorianCalendar(1982, 12, 12), "female", 58, 181, 45 },
{ "Lance", "Chicago", new GregorianCalendar(1962, 3, 12), "female", 49, 160, 57 },
{ "Eva", "Washington", new GregorianCalendar(1993, 2, 5), "female", 71, 180, 81 } };
```

```
// Set data
worksheet.getRange("A1:G9").setValue(data);

/* Copy content of active sheet from source_workbook to the current workbook
   before the first sheet */
IWorksheet copy_worksheet = worksheet.copyBefore(workbook.getWorksheets().get(0));
copy_worksheet.setName("Copy of Sheet1");
copy_worksheet.activate();

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

## Range Operations

A cell or a collection of cells in a worksheet is called Range and the various operations executed on the cells in rows and columns is called Range Operations.

In GcExcel Java, the **getRange ('getRange Method' in the on-line documentation)** method in the **IWorksheet ('IWorksheet Interface' in the on-line documentation)** interface enables users to perform range operations.

Using GcExcel Java, users can handle the following range operations:

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

## Access a Range

An array of cells defined in a worksheet is called range.

Using GcExcel Java, you can define a range and access the rows and columns in order to perform essential tasks like cell



formatting, insert, merge and delete operations etc.

In order to access a range using different methods, refer to the following example code.

## Java

```
// Create a new workbook
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Using index to access cell A1.
worksheet.getRange(0, 0).getInterior().setColor(Color.GetLightGreen());

// Using index to access cell range A1:B2
worksheet.getRange(0, 0, 2, 2).setValue(5);

// Using string to access range.
worksheet.getRange("A2").getInterior().setColor(Color.GetLightYellow());
worksheet.getRange("C3:D4").getInterior().setColor(Color.GetTomato());
worksheet.getRange("A5:B7, C3, H5:N6").setValue(2);

// Using index to access rows
worksheet.getRows().get(2).getInterior().setColor(Color.GetLightSalmon());

// Using string to access rows
worksheet.getRange("4:4").getInterior().setColor(Color.GetLightSkyBlue());

// Using index to access columns
worksheet.getColumns().get(2).getInterior().setColor(Color.GetLightSalmon());

// Using string to access columns
worksheet.getRange("D:D").getInterior().setColor(Color.GetLightSkyBlue());

// Using Cells to access range.
worksheet.getCells().get(5).getInterior().setColor(Color.GetLightBlue());
worksheet.getCells().get(5, 5).getInterior().setColor(Color.GetLightYellow());

// Access all rows in worksheet
String allRows = worksheet.getRows().toString();

// Access all columns in worksheet
String allColumns = worksheet.getColumns().toString();

// Access the entire sheet range
String entireSheet = worksheet.getCells().toString();
```

## Access Areas in a Range

In a large worksheet with non-contiguous selections, you can access specific areas in a multiple-area range by using the **getArea** ('getArea Method' in the on-line documentation) method of the **IAreas** ('IAreas Interface' in the on-line documentation) interface and **getAreas** ('getAreas Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface.

The methods of the **IAreas** interface and the **IRange** interface also represent the area count (number of areas) of the multiple-area range and all the selected ranges in the multiple area range.

In order to access areas in a range, refer to the following example code.

Java

```
IRange range = worksheet.getRange("A5:B7, C3, H5:N6");

// Access the first area - area1 is A5:B7.
IRange area1 = worksheet.getRange("A5:B7, C3, H5:N6").getAreas().getArea(0);

// Set interior color for the first area
area1.getInterior().setColor(Color.GetPink());

// Access the second area - area2 is C3.
IRange area2 = worksheet.getRange("A5:B7, C3, H5:N6").getAreas().getArea(1);

// Set interior color for the second area
area2.getInterior().setColor(Color.GetLightGreen());

// Access the third area - area3 is H5:N6.
IRange area3 = worksheet.getRange("A5:B7, C3, H5:N6").getAreas().getArea(2);

// Set interior color for the third area
area3.getInterior().setColor(Color.GetLightBlue());
```

## Access Cells, Rows and Columns in a Range

You can access cells, rows and columns in a range by using the **getCells** ('getCells Method' in the on-line documentation) method, the **getRows** ('getRows Method' in the on-line documentation) method and the **getColumns** ('getColumns Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface.

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

Java

```
// Create a new workbook and fetch the worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().add();

// Access Range and set A1:B5, C2:E4's interior color
worksheet.getRange("A1:B5, C2:E4").getInterior().setColor(Color.GetGreen());
```

```
// Access Rows and set row 1's font size
worksheet.getRows().get(0).getFont().setSize(20);

// Access Columns and delete column C
worksheet.getColumns().get(2).delete();

// Access Cells and set A1's value
worksheet.getCells().get(0).setValue(1);
```

## Cut or Copy Cell Ranges

GcExcel Java enables users 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](#).

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

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

### Copy Cell Range

You can copy a cell or a range of cells in the worksheet by calling the **copy ('copy Method' in the on-line documentation)** method of the **IRange ('IRange Interface' in the on-line documentation)** interface. In order to copy a single cell or a range of cells, specify the cell range to be copied, for example **B3:D12**.

GcExcel Java provides the following different ways to use the copy method.

Example	Description
<code>copy(sheet.getRange["E5"])</code>	This method copies data from cell range <b>B3:D12</b> and pastes the data to cell <b>E5</b> onwards.
<code>copy(sheet.getRange["E5:G14"])</code>	This method copies data from cell range <b>B3:D12</b> and pastes the data in cell range <b>E5:G14</b> . In case the range of cells copied does not fit into the destination cell range, the data is lost.

In order to copy the cell range in a workbook, refer to the following example code:

```
Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Set data of PC
worksheet.getRange("A2").setValue("PC");

Object data = new Object[] { "Device", "Quantity", "Unit Price" };
```

```
worksheet.getRange("A4:C4").setValue(data);

Object otherData = new Object[][] {
{ "T540p", 12, 9850 }, { "T570", 5, 7460 }, { "Y460", 6, 5400 },
{ "Y460F", 8, 6240 }, };
worksheet.getRange("A5:C10").setValue(otherData);

// Set style
worksheet.getRange("A2").setRowHeight(30);
worksheet.getRange("A2").getFont().setSize(20);
worksheet.getRange("A2").getFont().setBold(true);
worksheet.getRange("A4:C4").getFont().setBold(true);
worksheet.getRange("A4:C4").getFont().setColor
(com.grapacity.documents.excel.Color.GetWhite());
worksheet.getRange("A4:C4").getInterior().setColor
(com.grapacity.documents.excel.Color.GetLightBlue());
worksheet.getRange("A5:C10").getBorders()
.get(BordersIndex.InsideHorizontal)
.setColor(com.grapacity.documents.excel.Color.GetOrange());
worksheet.getRange("A5:C10").getBorders()
.get(BordersIndex.InsideHorizontal)
.setLineStyle(BorderLineStyle.DashDot);

// Copy only style and row height from cells A2:C10
worksheet.getRange("H1").setValue("Copy style and row height from previous cells.");
worksheet.getRange("H1").getFont().setColor(com.grapacity.documents.excel.Color.GetRed());
worksheet.getRange("H1").getFont().setBold(true);
worksheet.getRange("A2:C10").copy(worksheet.getRange("H2"),
EnumSet.of(PasteType.Formats));

// Set data of mobile devices
worksheet.getRange("H2").setValue("Mobile");

// Object data = new Object[] {"Device", "Quantity", "Unit Price" };
worksheet.getRange("H4:J4").setValue(data);

Object otherDataRange = new Object[][] {
{ "T540p", 12, 9850 }, { "T570", 5, 7460 }, { "Y460", 6, 5400 },
{ "Y460F", 8, 6240 }, };
worksheet.getRange("H5:J10").setValue(otherDataRange);

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

// Copy only style of Cell A2:C10 to new sheet
worksheet.getRange("A2:C10")
.copy(worksheet2.getRange("A2"), EnumSet.of(PasteType.Formats));
worksheet2.getRange("A3").setValue("Copy style from sheet1.");
worksheet2.getRange("A3").getFont()
```

```
.setColor(com.grapacity.documents.excel.Color.GetRed());
worksheet2.getRange("A3").getFont().setBold(true);

// Saving workbook to xlsx
workbook.save("PasteOptionsEnhancements.xlsx", SaveFileFormat.Xlsx);
```

### Working With Paste Options

Users can choose from several paste options while copying the data from the cell range. The **PasteType ('PasteType Enumeration' in the on-line documentation)** enumeration can be used to work with multiple paste options as described in the table shared below.

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

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

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

```
Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Set data of PC
worksheet.getRange("A2").setValue("PC");

Object data = new Object[] { "Device", "Quantity", "Unit Price" };
worksheet.getRange("A4:C4").setValue(data);

Object otherData = new Object[][] {
{ "T540p", 12, 9850 }, { "T570", 5, 7460 }, { "Y460", 6, 5400 },
{ "Y460F", 8, 6240 }, };
}
```

```
worksheet.getRange("A5:C10").setValue(otherData);

// Set style
worksheet.getRange("A2").setRowHeight(30);
worksheet.getRange("A2").getFont().setSize(20);
worksheet.getRange("A2").getFont().setBold(true);
worksheet.getRange("A4:C4").getFont().setBold(true);
worksheet.getRange("A4:C4").getFont().setColor(
    com.grapecity.documents.excel.Color.GetWhite());
worksheet.getRange("A4:C4").getInterior().setColor(
    com.grapecity.documents.excel.Color.GetLightBlue());
worksheet.getRange("A5:C10").getBorders()
    .get(BordersIndex.InsideHorizontal)
    .setColor(com.grapecity.documents.excel.Color.GetOrange());
worksheet.getRange("A5:C10").getBorders()
    .get(BordersIndex.InsideHorizontal)
    .setLineStyle(BorderLineStyle.DashDot);

// Copy only style and row height from cells A2:C10
worksheet.getRange("H1").setValue("Copy style and row height from previous cells.");
worksheet.getRange("H1").getFont().setColor(com.grapecity.documents.excel.Color.GetRed());
worksheet.getRange("H1").getFont().setBold(true);
worksheet.getRange("A2:C10").copy(worksheet.getRange("H2"),
    EnumSet.of(PasteType.Formats));

// Set data of mobile devices
worksheet.getRange("H2").setValue("Mobile");

// Object data = new Object[] {"Device", "Quantity", "Unit Price" };
worksheet.getRange("H4:J4").setValue(data);

Object otherDataRange = new Object[][] {
    { "T540p", 12, 9850 }, { "T570", 5, 7460 }, { "Y460", 6, 5400 },
    { "Y460F", 8, 6240 }, };
worksheet.getRange("H5:J10").setValue(otherDataRange);

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

// Copy only style of Cell A2:C10 to new sheet
worksheet.getRange("A2:C10")
    .copy(worksheet2.getRange("A2"), EnumSet.of(PasteType.Formats));
worksheet2.getRange("A3").setValue("Copy style from sheet1.");
worksheet2.getRange("A3").getFont()
    .setColor(com.grapecity.documents.excel.Color.GetRed());
worksheet2.getRange("A3").getFont().setBold(true);

// Saving workbook to.xlsx
workbook.save("PasteOptionsEnhancements.xlsx", SaveFileFormat.Xlsx);
```

### Cut Cell Range

You can cut a cell or a range of cells in a worksheet by calling the **cut** ('**cut Method**' in the on-line documentation) method of the **IRange** ('**IRange Interface**' in the on-line documentation) interface. To cut a cell or a range of cells, specify the cell range that you want to move, for example **B3:D12**.

GcExcel Java provides the following different ways to use the cut method.

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

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

Java
<pre>IRange range1 = worksheet2.getRange("E5");  // Cut the data of the range of cell worksheet.getRange("B3:D12").cut(range1);  // OR IRange range1 = worksheet2.getRange("E5;G14"); worksheet.getRange("B3:D12").cut(range1);</pre>

## Cut or Copy Shape, Slicer, Chart and Picture

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

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

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

In order to cut or copy shape, slicer, chart and picture in GcExcel Java, refer to the following example code.

Java
<pre>Workbook workbook = new Workbook(); IWorksheet worksheet = workbook.getWorksheets().get(0);  // Create a shape in worksheet, shape's range is Range("A7:B7") IShape shape = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 40, 40, 100,</pre>

```

100);
shape.getTextFrame().getTextRange().getFont().getColor().setRGB(Color.FromArgb(0, 255,
0));

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

// Cross sheet copy operation - copy a new shape to worksheet2's Range["C1:F7"]
IWorksheet worksheet2 = workbook.getWorksheets().add();
worksheet.getRange("A1:D10").copy(worksheet2.getRange("C1"));
worksheet.getRange("A1:D10").copy(worksheet2.getRange("C1:G9"));

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

// Cross sheet cut operation - cut a new shape to worksheet2's Range["C1:F7"]
IWorksheet worksheet3 = workbook.getWorksheets().add();
worksheet.getRange("A1:D10").cut(worksheet3.getRange("C1"));
worksheet2.getRange("A1:D10").cut(worksheet3.getRange("C1:G9"));

```

To duplicate a shape to the current worksheet, you can use the methods of the **IShape ('IShape Interface' in the on-line documentation)** interface.

In order to duplicate an existing shape, slicer, chart and picture, refer to the following example code.

#### Java

```

// Create Shape
IShape shape1 = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 50, 50, 200,
200);

// Create Chart, chart's range is Range["G1:M21"]
IShape chart = chartworksheet.getShapes().addChart(ChartType.ColumnClustered, 300, 10,
300, 300);
chartworksheet.getRange("A1:D6").setValue(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}
});
chart.getChart().getSeriesCollection().add(chartworksheet.getRange("A1:D6"),
RowCol.Columns, true, true);

// Create slicer cache for table.
ISlicerCache cache = workbook1.getSlicerCaches().add(table, "Category",

```



```
"categoryCache");

// Create slicer
ISlicer slicer = cache.getSlicers().add(workbook1.getWorksheets().get("Sheet1"),
"catel", "Category", 30, 550, 100, 200);

// Create Picture
IShape picture = worksheet.getShapes().addPicture("C:/Pictures", 1, 1, 100, 100);

// Duplicate Shape
IShape newShape = shape1.duplicate();

// Duplicate Chart
IShape newchart = chart.duplicate();

// Duplicate Slicer
IShape slicerShape = slicer.getShape().duplicate();

// Duplicate Picture
IShape newPicture = picture.duplicate();
```

## Find and Replace Data

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

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

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

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

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

### Java

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

// Fetch default worksheet
```

```
IWorksheet worksheet = workbook.getWorksheets().get(0);

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

// Set data
worksheet.getRange("A1:G9").setValue(data);
worksheet.getRange("I10:P19").setValue(data);
worksheet.getRange("A21:G29").setValue(data);

String what = "newyork";
String replacement = "NewYork";
ReplaceOptions ro = new ReplaceOptions();
ro.setMatchCase(true);

// Specify range to search in
IRange searchRange = worksheet.getRange("A1:G9,I10:P19");

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

// Saving workbook to xlsx
workbook.save("FindAndReplaceData.xlsx", SaveFileFormat.Xlsx);
```

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

#### Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Set formulas
worksheet.getRange("A1:H5").setFormula("SUM(6,10)");

FindOptions fo = new FindOptions();
fo.setLookIn(FindLookIn.Formulas);
```

```
IRange range = null;

// Specify range to search in formulas
IRange searchRange = worksheet.getRange("A1:B4");
do
{
    range = searchRange.find("SUM", range, fo);
    if (range != null)
    {
        // Using Replace method to replace formula in searched range
        range.setFormulaArray(range.getFormula().replace("SUM", "PRODUCT"));
    }
} while (range != null);

// Saving workbook to xlsx
workbook.save("FindAndReplaceFormulas.xlsx", SaveFileFormat.Xlsx);
```

## Get Row and Column Count

When you have a worksheet with bulk data, it becomes cumbersome to manually fetch the number of rows and columns.

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

The fields and methods of the **IRange ('IRange Interface' in the on-line documentation)** interface represent the cell count of all the areas in a range.

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

Java

```
IRange range = worksheet.getRange("A5:B7, C3, H5:I6");

// Cell count is 11. All areas cell count
int cellcount = range.getCount();
System.out.println(cellcount);

// Cell count is 11. All areas cell count
int cellcount1 = range.getCells().getCount();
System.out.println(cellcount1);

// Row count is 3. First area's row count
int rowcount = range.getRows().getCount();
System.out.println(rowcount);

// Column count is 2. First area's column count
int columncount = range.getColumns().getCount();
```

## Hide Rows and Columns

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

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

```
Java

worksheet.getRange("E1").setValue(1);

// Hide row 2:6 using the setHidden method.
worksheet.getRange("2:6").setHidden(true);

// Hide column A:D using the setHidden method.
worksheet.getRange("A:D").setHidden(true);
```

## Insert And Delete Cell Ranges

GcExcel Java enables users to insert and delete a cell or a range of cells while working with spreadsheets. This facilitates the customization of worksheets based on specific requirements.

### Insert cell range

GcExcel Java 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 the **IRange** ('IRange Interface' in the on-line documentation) interface. 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.

You can choose from the following options while inserting a cell or a range of cells in a worksheet.

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 inserts the range of cells and shifts the existing range of cells to the right.

In order to insert a single cell and a cell range in the worksheet, refer to the following example code.

```
Java

// Insert the range of cell
worksheet.getRange("A3").insert();

// Insert the range of cells
worksheet.getRange("A3:C10").insert();
```

In order to insert cell range in a worksheet while specifying the desired shift direction for the existing cells, refer to the

following example code.

Java

```
// Insert the range of cells from desired direction
worksheet.getRange("A3:C10").insert(InsertShiftDirection.Down);
worksheet.getRange("A3:C10").insert(InsertShiftDirection.Right);
```

### Delete cell range

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

GcExcel Java provides the 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.

In order to delete a single cell or a cell range in a worksheet, refer to the following example code.

Java

```
// Delete the range of cell
worksheet.getRange("A3").delete();

// Delete the range of cells
worksheet.getRange("A3:C10").delete();
```

In order to delete a single cell or a range of cells in a worksheet while specifying the desired shift direction for the existing cells, refer to the following example code.

Java

```
// Delete the range of cells from desired direction
worksheet.getRange("A3:C10").delete>DeleteShiftDirection.Left);
worksheet.getRange("A3:C10").delete>DeleteShiftDirection.Up);
```

## Insert and Delete Rows and Columns

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

### Insert rows and columns

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

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

In order to insert rows in a worksheet, refer to the following example code.

Java

```
// Insert rows
worksheet.getRange("A3:A5").getEntireRow().insert();
// OR
worksheet.getRange("3:5").insert();
```

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

In order to insert columns in a worksheet, refer to the following example code.

Java

```
// Insert columns
worksheet.getRange("A3:A5").getEntireColumn().insert();
// OR
worksheet.getRange("3:5").insert();
```

## Delete row and column

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

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.

In order to delete rows from the worksheet, refer to the following example code.

Java

```
// Delete rows
worksheet.getRange("A3:A5").getEntireRow().delete();
// OR
worksheet.getRange("3:5").delete();
```

In order to delete columns from the worksheet, refer to the following example code.

Java

```
// Delete columns
worksheet.getRange("A3:A5").getEntireColumn().delete();
```

```
// OR  
worksheet.getRange("3:5").delete();
```

## Merge Cells

GcExcel Java enables users to execute the merge operation on several cells by combining them into a single cell using **merge ('merge Method' in the on-line documentation)** method of the **IRange ('IRange Interface' in the on-line documentation)** interface. 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.

If all the cells within the given range are empty, the formatting of top left cell in the cell range is applied to the merged cell, by default.

In order to merge the range of cells, refer to the following example code.

```
Java  
  
// Merge the cell range A1:C4 into one single cell  
worksheet.getRange("A1:C4").merge();
```

In order to merge only the rows of the specified range of cell into one, refer to the following example code.

```
Java  
  
// Merge the cell range H5:J6 into a single merged cell in one row.  
worksheet.getRange("H5:J6").merge(true);
```

## Set Values to a Range

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

In order to set custom values to cell ranges in the worksheet, refer to the following example code.

```
Java  
  
worksheet.getRange("A:F").setColumnWidth(15);  
  
Object data = new Object[][] { { "Name", "City", "Birthday", "Eye color", "Weight",  
    "Height" },  
    { "Richard", "New York", new GregorianCalendar(1968, 5, 8), "Blue", 67, 165 },  
    { "Nia", "New York", new GregorianCalendar(1972, 6, 3), "Brown", 62, 134 },  
    { "Jared", "New York", new GregorianCalendar(1964, 2, 2), "Hazel", 72, 180 },  
    { "Natalie", "Washington", new GregorianCalendar(1972, 7, 8), "Blue", 66, 163 },  
    { "Damon", "Washington", new GregorianCalendar(1986, 1, 2), "Hazel", 76, 176 },  
    { "Angela", "Washington", new GregorianCalendar(1993, 1, 15), "Brown", 68, 145 }  
};  
  
// set two-dimension array value to range A1:F7  
worksheet.getRange("A1:F7").setValue(data);
```

```
// return a two-dimension array when get range A1:B7's value.  
Object result = worksheet.getRange("A1:B7").getValue();
```

## Set Row Height and Column Width

You can customize the height of the rows and the width of the columns in a spreadsheet based on specific preferences.

The **setRowHeight** ('setRowHeight Method' in the on-line documentation) method and the **setColumnWidth** ('setColumnWidth Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface can be used to set custom row height (in points) and column width (in characters) for the individual rows and columns of a worksheet, respectively.

The **setRowHeightInPixel** ('setRowHeightInPixel Method' in the on-line documentation) method and the **setColumnWidthInPixel** ('setColumnWidthInPixel Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface can be used to set the custom row height and column width (in pixels) for the rows and columns of a worksheet, respectively.

In order to set the row height and column width in a worksheet, refer to the following example code.

Java

```
// Set row height for row 1:2.  
worksheet.getRange("1:2").setRowHeight(50);  
  
// Set column width for column C:D.  
worksheet.getRange("C:D").setColumnWidth(20);
```

## Auto Fit Row Height and Column Width

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

### Advantage of Using Auto Fit Feature

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

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

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

- This feature supports the auto adjustment of column width and row height of specific cell ranges only.
- Users can use the **autoFit()** ('autoFit Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface in order to auto fit row height and column width.
- If the type of the cell range used is a column (this can be determined using **IRange.getColumns** ('getColumns



**Method' in the on-line documentation)/IRange.getEntireColumn ('getEntireColumn Method' in the on-line documentation) etc.), then only the column width will be adjusted to best fit but the row height will not be changed.**

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

## Java

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


// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Auto fit column width of range 'A1'
worksheet.getRange("A1").setValue("Grapecity Documents for Excel");
worksheet.getRange("A1").getColumns().autoFit();

// Auto fit row height of range 'B2'
worksheet.getRange("B2").setValue("Grapecity Documents for Excel");
worksheet.getRange("B2").getFont().setSize(20);
worksheet.getRange("B2").getRows().autoFit();

// Auto fit column width and row height of range 'C3'
worksheet.getRange("C3").setValue("Grapecity Documents for Excel");
worksheet.getRange("C3").getFont().setSize(32);
worksheet.getRange("C3").autoFit();

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

 Note: The Auto fit feature has the following limitations :

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

## Work with Used Range

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

GcExcel Java enables users to work with the already used range of cells in a worksheet in the following ways:

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

## Work with worksheet's used range

While working with the worksheet's used range, the first step is to get the used range by using the **getUsedRange** ('**getUsedRange Method**' in the on-line documentation) method of the **IWorksheet** ('**IWorksheet Interface**' in the on-line documentation) interface. As a second step, you can use the methods of the **IRange** ('**IRange Interface**' in the on-line documentation) interface to further customize the used range as per your preferences.

In order to get used range and customize it, refer to the following example code.

Java

```
worksheet.getRange("H6:M7").setValue(1);
worksheet.getRange("J9:J10").merge();

// UsedRange is "H6:M10"
String usedrange = worksheet.getUsedRange().toString();

// Customize the used range
usedRange.getInterior().setColor(Color.GetLightBlue());
```

## Work with feature related used range

While working with the feature related used range, the first step is to get the feature related used range by using the **getUsedRange** ('**getUsedRange Method**' in the on-line documentation) method of the **IWorksheet** ('**IWorksheet Interface**' in the on-line documentation) interface. As a second step, you can customize the feature related used range using the methods of the **IRange** ('**IRange Interface**' in the on-line documentation) interface.

In order to get feature related used range and customize it, refer to the following example code.

Java

```
IComment commentA1 = worksheet.getRange("A1").addComment("Range A1's comment");
IComment commentA2 = worksheet.getRange("A2").addComment("Range A2's comment");

// Comment used range is "A1:D5", contains comment shape plot area
EnumSet<UsedRangeType> usedRangeTypes = EnumSet.of(UsedRangeType.Comment);
String commentsUsedRange = worksheet.getUsedRange(usedRangeTypes).toString();
System.out.println(commentsUsedRange);

worksheet.getRange("A1:B2").setValue(new Object[][] { { 1, 2 }, { "aaa", "bbb" } });
worksheet.getRange("A2:C3").getInterior().setColor(Color.GetGreen());

// Customize the feature related used range by applying style - used range is A2:C3.
IRange usedRange_style = worksheet.getUsedRange(EnumSet.of(UsedRangeType.Style));
usedRange_style.getInterior().setColor(Color.GetLightBlue());
System.out.println(usedRange_style);
```

After getting the used range of cells with the help of any of the above two methods, you can customize the used range as per your preferences. For example- you can set the row height and column width; change the row hidden and column hidden settings; execute essential tasks like group and merge operations; insert values, formulas and comments to the used range in your spreadsheet, as and when required.

## Freeze Panes in a Worksheet

GcExcel Java enables users to freeze panes in a worksheet. This feature allows users to keep some specific rows or columns visible while scrolling through the rest of the sheet. In a large worksheet with a lot of data, this functionality is helpful in enhancing readability and data manipulation of bulk information that spans across a number of rows or columns.

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

### Freeze Panes

You can freeze panes in a worksheet using the **freezePanes** ('freezePanes Method' in the on-line documentation) method of the **IWorksheet** ('IWorksheet Interface' in the on-line documentation) interface. This method freezes the split panes based on the specified 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 **getFreezeRow** ('getFreezeRow Method' in the on-line documentation) and **getFreezeColumn** ('getFreezeColumn Method' in the on-line documentation) methods respectively.

To see how panes in a worksheet can be frozen, refer to the following example code.

Java

```
// Adding worksheets to the workbook
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.freezePanes(worksheet.getRange("A5").getRow(),
worksheet.getRange("A5").getColumn());
IWorksheet worksheet2 = workbook.getWorksheets().add();

// Freeze Panes
worksheet2.freezePanes(worksheet.getRange("B10").getRow(),
worksheet.getRange("B10").getColumn());
```

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

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

Java

```
// Use sheet index to get worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Freeze pane
worksheet.freezePanes(10, 10);

// Set frozen line color as red
worksheet.setFrozenLineColor(Color.GetBlue());

// Save workbook to ssjson
String json = workbook.toJson();
```

## Unfreeze Panes

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

To see how frozen panes in a worksheet can be unfrozen, refer to the following example code.

Java

```
// Unfreeze all panes in worksheet2
worksheet2.unfreezePanes();
```

## Freeze Trailing Panes in a Worksheet

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

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

## Freeze Trailing Panes

The trailing panes in a worksheet can be frozen by using the **freezeTrailingPanes** method of **IWorksheet** interface which takes row and column positions as parameters. The number of frozen rows and column can also be retrieved by using the **getFreezeTrailingRow** and **getFreezeTrailingColumn** methods respectively.

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

Java

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

// Use sheet index to get worksheet.
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Freeze trailing pane
worksheet.freezeTrailingPanes(2, 3);

System.out.println("Number of trailing rows are: " + worksheet.getFreezeTrailingRow()
    + "\nNumber of trailing columns are: " + worksheet.getFreezeTrailingColumn());

// Save workbook to ssjson
String json = workbook.toJson();
try {
    BufferedWriter out = new BufferedWriter(
        new OutputStreamWriter(new
    FileOutputStream("FreezeTrailingRowsCols.ssjson"), "utf-8"));
    out.write(json);
```

```
        out.flush();
        out.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

### Unfreeze Trailing Panes

Similarly, the frozen trailing panes can be unfrozen by using the **unfreezeTrailingPanes** method of [IWorksheet](#) interface. Refer to the following example code to unfreeze trailing panes in a worksheet.

#### Java

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

// Use sheet index to get worksheet.
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Freeze trailing pane
worksheet.freezeTrailingPanes(2, 3);

// Unfreeze trailing pane
worksheet.unfreezeTrailingPanes();

// Save workbook to ssjson
String json = workbook.toJson();
try {
    BufferedWriter out = new BufferedWriter(
        new OutputStreamWriter(new
FileOutputStream("UnFreezeTrailingRowsCols.ssjson"), "utf-8"));
    out.write(json);
    out.flush();
    out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

## Customize Worksheets

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

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

- **Configure display**
- **Set the tab color**

- Set visibility
- Set background image
- Define standard height and width

## Configure display

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

In order to configure the display of your worksheet as per your preferences, refer to the following example code.

Java

```
// Configure Sheet Settings
Workbook workbook = new Workbook();

// Fetch the default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Assign the values to the cells
worksheet.getRange("B1").setValue("ABCD");
worksheet.getRange("B2").setValue(3);
worksheet.getRange("C1").setValue("GrapeCity Documents");
worksheet.getRange("C2").setValue(4);
worksheet.getRange("D1").setValue("GcExcel");
worksheet.getRange("D2").setValue("ABCD");
worksheet.getSheetView().setDisplayRightToLeft(true);
```

## Set the tab color

You can change the default tab color of your worksheet using the **setTabColor** ('**setTabColor Method** in the **on-line documentation**') method of the **IWorksheet** ('**IWorksheet Interface** in the **on-line documentation**') interface.

In order to set the tab color for your worksheet as per your preferences, refer to the following example code.

Java

```
// Set the tab color of the specified sheet as green.
worksheet.setTabColor(Color.GetGreen());
```

## Set visibility

You can show or hide your worksheet using the **setVisible** ('**setVisible Method** in the **on-line documentation**') method of the **IWorksheet** ('**IWorksheet Interface** in the **on-line documentation**') interface.

In order to set the visibility of your worksheet, refer to the following example code.

Java

```
// Adding new sheet and set the visibility of the sheet as Hidden.
IWorksheet worksheet1 = workbook.getWorksheets().add();
worksheet1.setVisible(Visibility.Hidden);
```

## Set background image

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

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

Java

```
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// To load an image from a specific file in input stream
InputStream inputStream = ClassLoader.getResourceAsStream("GrapeCityLogo.png");
try {
    byte[] bytes = new byte[inputStream.available()];
    // Read an image from input stream
    inputStream.read(bytes, 0, bytes.length);

    // Setting worksheet's BackgroundPicture
    worksheet.setBackgroundPicture(bytes);
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

## Define standard width and height

You can define the standard height and width of your worksheet using the **setStandardHeight** ('setStandardHeight Method' in the on-line documentation) and **setStandardWidth** ('setStandardWidth Method' in the on-line documentation) methods of the **IWorksheet** ('IWorksheet Interface' in the on-line documentation) interface, respectively.

In order to define the standard width and height as per your requirements, refer to the following example code.

Java

```
// Setting the height and width of the worksheet
worksheet.setStandardHeight(20);
worksheet.setStandardWidth(40);
```

## Worksheet Views

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

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

and **IWorksheetView** ('**IWorksheetView Interface** in the on-line documentation) interface.

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

Method	Description
<b>IWorksheet.splitPanes(int row, int column)</b> (' <b>splitPanes Method</b> ' in the on-line documentation)	This method can be used to lock the rows and columns in a worksheet in order to divide the worksheet into multiple areas that can be scrolled independently. Users need to provide the cell index as parameters in this method to specify the location where they want the split.
<b>IWorksheet.unsplitPanes(int row, int column)</b> (' <b>unsplitPanes Method</b> ' in the on-line documentation)	This method can be used to unsplit the split panes. Using this method is similar to using IWorksheet.SplitPanes(0,0).
<b>IWorksheet.getSplitRow</b> (' <b>getSplitRow Method</b> ' in the on-line documentation) / <b>IWorksheet.getSplitColumn</b> (' <b>getSplitColumn Method</b> ' in the on-line documentation)	This method gets the split distances (row count and column count) from top (in case of row) or left (in case of column).
<b>IWorksheet.getPanes</b> (' <b>getPanes Method</b> ' in the on-line documentation)	A range object that represents the frozen or split panes of the worksheet.
<b>IWorksheet.getActivePane</b> (' <b>getActivePane Method</b> ' in the on-line documentation)	This method can be used to get the active pane in a worksheet.
<b>IPane.activate()</b> (' <b>activate Method</b> ' in the on-line documentation)	This method activates the current pane.
<b>IPane.getIndex</b> (' <b>getIndex Method</b> ' in the on-line documentation)	This method can be used to get the index of the current pane in IWorksheet.Panes.
<b>IPane.setScrollColumn</b> (' <b>setScrollColumn Method</b> ' in the on-line documentation) / <b>IPane.setScrollRow</b> (' <b>setScrollRow Method</b> ' in the on-line documentation)	This method can be used to get or set the top left cell position of the current pane.
<b>IWorksheet.getSheetView</b> (' <b>getSheetView Method</b> ' in the on-line documentation)	This method can be used to get the view of the worksheet.
<b>IWorksheetView.setZoom</b> (' <b>setZoom Method</b> ' in the on-line documentation)	This method can be used to get and set a variant numeric value that represents the display size of the worksheet as a percentage where the 100 equals normal size, 200 equals double size, and so on.
<b>IWorksheetView.setGridlineColor</b> (' <b>setGridlineColor Method</b> ' in the on-line documentation)	This method can be used to get and set the gridline color.
<b>IWorksheetView.setScrollColumn</b> (' <b>setScrollColumn Method</b> ' in the on-line documentation)	This method can be used to get and set the number of the leftmost column in the worksheet.



<b>IWorksheetView.setScrollRow</b> ( <b>'setScrollRow Method'</b> in the on-line documentation)	This method can be used to get and set the number of the row that appears at the top of the worksheet.
<b>IWorksheetView.setDisplayRightToLeft</b> ( <b>'setDisplayRightToLeft Method'</b> in the on-line documentation)	This method can be used to get and set whether the specified worksheet is displayed from right to left instead of from left to right.
<b>IWorksheetView.setDisplayFormulas</b> ( <b>'setDisplayFormulas Method'</b> in the on-line documentation)	This method can be used to get and set whether the worksheet displays formulas.
<b>IWorksheetView.setDisplayGridlines</b> ( <b>'setDisplayGridlines Method'</b> in the on-line documentation)	This method can be used to get and set whether the gridlines are displayed.
IWorksheetView.setDisplayVerticalGridlines	This method can be used to get and set whether the vertical gridlines are displayed.
IWorksheetView.setDisplayHorizontalGridlines	This method can be used to get and set whether the horizontal gridlines are displayed.
<b>IWorksheetView.setDisplayHeadings</b> ( <b>'setDisplayHeadings Method'</b> in the on-line documentation)	This method can be used to get and set whether the headers are displayed.
<b>IWorksheetView.setDisplayOutline</b> ( <b>'setDisplayOutline Method'</b> in the on-line documentation)	This method can be used to get and set whether the outline symbols are displayed.
<b>IWorksheetView.setDisplayRuler</b> ( <b>'setDisplayRuler Method'</b> in the on-line documentation)	This method can be used to get and set whether a ruler is displayed for the specified worksheet.
<b>IWorksheetView.setDisplayWhitespace</b> ( <b>'setDisplayWhitespace Method'</b> in the on-line documentation)	This method can be used to get and set whether the whitespace is displayed.
<b>IWorksheetView.setDisplayZeros</b> ( <b>'setDisplayZeros Method'</b> in the on-line documentation)	This method can be used to get and set whether the zero values are displayed.

In order to set custom view for a worksheet using different methods of the IWorksheet interface, refer to the following example code.

Java

```
// Configure Sheet Settings
Workbook workbook = new Workbook();

// Fetch the default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
```

```
// Assign the values to the cells
worksheet.getRange("B1").setValue("ABCD");
worksheet.getRange("B2").setValue(3);
worksheet.getRange("C1").setValue("GrapeCity Documents");
worksheet.getRange("C2").setValue(4);
worksheet.getRange("D1").setValue("GcExcel");
worksheet.getRange("D2").setValue("ABCD");
worksheet.getSheetView().setDisplayRightToLeft(true);
```

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

Java

```
// Split worksheet to panes using splitPanes() method.
worksheet.splitPanes(worksheet.getRange("B3").getRow(),
worksheet.getRange("B3").getColumn());
```

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

Java

```
// Create a new workbook
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("A1:I88").setValue(10);

// Set to not show horizontal gridlines
worksheet.getSheetView().setDisplayHorizontalGridlines(false);

// Set to show vertical gridlines
worksheet.getSheetView().setDisplayVerticalGridlines(true);

// Save workbook to ssjson
String json = workbook.toJson();
try {
    BufferedWriter out = new BufferedWriter(
        new OutputStreamWriter(new
FileOutputStream("HorizontalVerticalGridlines.ssjson"), "utf-8"));
    out.write(json);
    out.flush();
    out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```




**Note:** If the value of **`setDisplayGridlines`** is set, **`setDisplayVerticalGridlines`** and **`setDisplayHorizontalGridlines`** are

also set to the same value.

## Cell Types

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

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

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

### Button Cell Type

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

Java

```
private static void ButtonCellTypes() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Creating ButtonCellType
    ButtonCellType button = new ButtonCellType();
    button.setText("Click Me...!!");
    button.setButtonBackColor("LightBlue");
    button.setMarginLeft(10);
    worksheet.getRange("A1:B2").setCellType(button);

    // Saving workbook to Pdf
    workbook.save("151-ButtonCellTypes.pdf", SaveFileFormat.Pdf);
}
```

### CheckBox Cell Type

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

Java

```
private static void CheckBoxCellTypes() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
```

```
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Creating CheckBoxCellType
CheckBoxCellType checkBox = new CheckBoxCellType();
checkBox.setCaption("Caption");
checkBox.setTextTrue("True");
checkBox.setTextFalse("False");
checkBox.setIsThreeState(false);
worksheet.getRange("A1:C3").setCellType(checkBox);

worksheet.getRange("A1").setValue(true);
worksheet.getRange("B2").setValue(true);

// Saving workbook to Pdf
workbook.save("152-CheckBoxCellTypes.pdf", SaveFileFormat.Pdf);
```

### ComboBox Cell Type

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

#### Java

```
private static void ComboCellTypes() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

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

    ComboBoxCellItem comboItem = new ComboBoxCellItem();
    comboItem.setValue("US");
    comboItem.setText("United States");
    comboBox.getItems().add(comboItem);

    comboItem = new ComboBoxCellItem();
    comboItem.setValue("CN");
    comboItem.setText("China");
    comboBox.getItems().add(comboItem);

    comboItem = new ComboBoxCellItem();
    comboItem.setValue("JP");
    comboItem.setText("Japan");
    comboBox.getItems().add(comboItem);

    worksheet.getRange("A1:B2").setCellType(comboBox);
}
```

```
worksheet.getRange("A1").setValue("CN");

// Saving workbook to Pdf
workbook.save("153-ComboCellTypes.pdf", SaveFileFormat.Pdf);
```

## Hyperlink Cell Type

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

Java

```
private static void HyperlinkCellTypes() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Creating HyperLinkCellType
    HyperLinkCellType hyperlinkCell = new HyperLinkCellType();
    hyperlinkCell.setText("GrapeCity Website");
    hyperlinkCell.setLinkColor("Blue");
    hyperlinkCell.setLinkToolTip("GrapeCity Website");
    hyperlinkCell.setVisitedLinkColor("Green");
    hyperlinkCell.setTarget(HyperLinkTargetType.Blank);

    worksheet.getRange("A1").setCellType(hyperlinkCell);
    worksheet.getRange("A1").setValue("https://www.grapecity.com/");

    // Saving workbook to Pdf
    workbook.save("154-HyperlinkCellTypes.pdf", SaveFileFormat.Pdf);
}
```

## Quote Prefix

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

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

Java

```
worksheet.getRange("C4").setValue("00001234");
worksheet.getRange("C5").setValue("Dec-1");

worksheet.getRange("F4").setValue("'006438098");
worksheet.getRange("F5").setValue("'Jan-4");
```

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

The screenshot shows an Excel spreadsheet with columns A through E and rows 1 through 6. The formula bar at the top shows the value 'Dec-1'. In cell C5, the value 'Dec-1' is displayed. The cell C4 contains the value '00001234'.

	A	B	C	D	E
1					
2					
3					
4			00001234		
5			Dec-1		
6					

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

The screenshot shows an Excel spreadsheet with columns D through H and rows 1 through 6. The formula bar at the top shows the value '"Jan-4"'. In cell F5, the value 'Jan-4' is displayed. The cell F4 contains the value '"006438098"'.

	D	E	F	G	H
1					
2					
3					
4			"006438098"		
5			Jan-4		
6					

## Tags

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

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

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

### Using Code

Refer to the following code to use tags:

Java

```
private static void Tags() {
    // Initialize workbook
    Workbook workbook = new Workbook();
```

```
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Add Tag for worksheet
worksheet.setTag("This is a Tag for sheet.");
// Add Tag for Cell C1
worksheet.getRange("C1").setTag("This is a Tag for Cell C1");
// Add Tag for Row 4
worksheet.getRange("A4").getEntireRow().setTag("This is a Tag for Row 4");
// Add Tag for Column F
worksheet.getRange("F5").getEntireColumn().setTag("This is a Tag for Column F");
// Add tag for Range A1:B2
worksheet.getRange("A1:B2").setTag("This is a Tag for A1:B2");

// Exporting workbook to JSON stream
String jsonstr = workbook.toJson();

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

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

// Get Tag of Range A1:B2
Object tag = workbook2.getWorksheets().get(0).getRange("A1:B2").getTag();

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

Refer to the following code to use custom tags:

#### Java

```
private static void CustomTag() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Set custom json serializer
    Workbook.setTagJsonSerializer(new MyJsonSerializer());

    // Set Tag of "A1" as custom type "Student"
    worksheet.getRange("A1").setTag(new Student("Robin", 7));

    // Exporting workbook to JSON stream
    String json = workbook.toJson();
```

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

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

// Get Tag as JObject
Object tag = workbook2.getWorksheets().get(0).getRange("A1").getTag();

// Convert JObject to "Student"
Student student = new Gson().fromJson((JsonElement) tag, Student.class);

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

public static class Student {
    public String name;
    public int age;

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

public static class MyJsonSerializer implements IJsonSerializer {
    Gson gson = new Gson();

    @Override
    public String serialize(Object value) {
        return gson.toJson(value);
    }

    @Override
    public Object deserialize(String json) {
        return gson.fromJson(json, JsonElement.class);
    }
}
```

## Rich Text

GcExcel.NET provides support for applying rich text formatting in the cells of the worksheet. By default, when textual information is entered in a cell, the alphabets are displayed without any formatting style. Rich text feature allows you to



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

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

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

	A	B	C
1	GrapeCity Documents for Excel		
2			
3			

Users can use any of the following ways in order to set the rich text in the cells of a worksheet -

- Using the **IRichText Interface**.
- Using the **IRange.characters()**
- Using the **IRange.characters()** to Configure Font Across Several Runs.
- Using the **ITextRun.insertAfter()** and **ITextRun.insertBefore()**.

## Using the IRichText Interface.

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

## Using Code

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

Java

```
// Setting column "A" width
worksheet.getRange("A1").setColumnWidth(70);

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

// Fetch the IRichText object associated with the cell range
IRichText richText = worksheet.getRange("A1").getRichText();

// Add string "GrapeCity " to IRichText object and apply formatting
ITextRun run1 = richText.add("GrapeCity ");
run1.getFont().setColor(Color.GetRed());
```

```
run1.getFont().setBold(true);
run1.getFont().setSize(20);

// Append string "Documents" to IRichText object and apply formatting
ITextRun run2 = richText.add("Documents");
run2.getFont().setThemeFont(ThemeFont.Major);
run2.getFont().setThemeColor(ThemeColor.Accent1);
run2.getFont().setSize(30);
run2.getFont().setUnderline(UnderlineType.Single);

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

// Append string "Excel" to IRichText object and apply formatting
ITextRun run3 = richText.add("Excel");
run3.getFont().setName("Arial Black");
run3.getFont().setColor(Color.GetLightGreen());
run3.getFont().setSize(36);
run3.getFont().setItalic(true);
```

### Using the IRange.characters()

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

### Using Code

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

#### Java

```
// Setting column "A" width
worksheet.getRange("A1").setColumnWidth(70);

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

// Setting Cell Text
worksheet.getRange("A1").setValue("GrapeCity Documents for Excel");

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

// Formatting string "Grapecity"
ITextRun run1 = worksheet.getRange("A1").characters(0, 9);
run1.getFont().setColor(Color.GetRed());
run1.getFont().setBold(true);
run1.getFont().setSize(20);
```

```
// Formatting string "Documents"
ITextRun run2 = worksheet.getRange("A1").characters(10, 9);
run2.getFont().setThemeFont(ThemeFont.Major);
run2.getFont().setThemeColor(ThemeColor.Accent1);
run2.getFont().setSize(30);
run2.getFont().setUnderline(UnderlineType.Single);

// Formatting string "Excel"
ITextRun run3 = worksheet.getRange("A1").characters(24, 5);
run3.getFont().setName("Arial Black");
run3.getFont().setColor(Color.GetLightGreen());
run3.getFont().setSize(36);
run3.getFont().setItalic(true);
```

### Using the IRange.characters() to Configure Font Across Several Runs

You can also insert rich text in the cells of a worksheet by using the **characters()** ('**characters Method**' in the **on-line documentation**) method of the **IRange** ('**IRange Interface**' in the **on-line documentation**) interface. Using this method, you can configure the font across several runs and then consolidate them into a single entity.

### Using Code

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

#### Java

```
// Setting column "A" width
worksheet.getRange("A1").setColumnWidth(75);

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

// Fetch the IRichText object associated with the cell range
IRichText richText = worksheet.getRange("A1").getRichText();

// Add string "GrapeCity " to IRichText object and apply formatting
ITextRun run1 = richText.add("GrapeCity ");
run1.getFont().setColor(Color.GetRed());
run1.getFont().setBold(true);
run1.getFont().setSize(20);

// Append string "Documents" to IRichText object and apply formatting
ITextRun run2 = richText.add("Documents");
run2.getFont().setThemeFont(ThemeFont.Major);
run2.getFont().setThemeColor(ThemeColor.Accent1);
run2.getFont().setSize(30);
run2.getFont().setUnderline(UnderlineType.Single);
```

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

// Append string "Excel" to IRichText object and apply formatting
ITextRun run3 = richText.add("Excel");
run3.getFont().setName("Arial Black");
run3.getFont().setColor(Color.GetLightGreen());
run3.getFont().setSize(36);
run3.getFont().setItalic(true);

// Create composite run
// Extract character range composed of "City" word from run1 and " for" word and apply
// formatting
ITextRun compositeRun = worksheet.getRange("A1").characters(5, 18);
compositeRun.getFont().setBold(true);
compositeRun.getFont().setItalic(true);
compositeRun.getFont().setThemeColor(ThemeColor.Accent1);
```

### Using the ITextRun.insertAfter() and ITextRun.insertBefore()

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

### Using Code

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

#### Java

```
// Setting column "A" width
worksheet.getRange("A1").setColumnWidth(70);

// Use ITextRun.insertAfter() and insertBefore() to add rich text

// Fetch the IRichText object associated with the cell range
IRichText richText = worksheet.getRange("A1").getRichText();

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

// Use InsertBefore() to add string "Documents" to run1 and apply formatting
ITextRun run2 = run1.insertBefore("Documents");
run2.getFont().setThemeFont(ThemeFont.Major);
```

```
run2.getFont().setThemeColor(ThemeColor.Accent1);
run2.getFont().setSize(30);
run2.getFont().setUnderline(UnderlineType.Single);

// Use InsertBefore() to add string "GrapeCity " to run2 and apply formatting
ITextRun run3 = run2.insertBefore("GrapeCity ");
run3.getFont().setColor(Color.GetRed());
run3.getFont().setBold(true);
run3.getFont().setSize(20);

// Use InsertAfter() to add string "Excel" to run1 and apply formatting
ITextRun run4 = run1.insertAfter("Excel");
run4.getFont().setName("Arial Black");
run4.getFont().setColor(Color.GetLightGreen());
run4.getFont().setSize(36);
run4.getFont().setItalic(true);
```

## Workbook

GcExcel Java provides all the essential packages with required methods and fields to facilitate users in creating a workbook and performing complex operations on the data while making use of several workbook events that can be triggered by the user through code.

You can manage workbook in the following ways:

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

## Create Workbook

You can create a new instance of a workbook using the **Workbook** ('**Workbook Class**' in the on-line documentation) class.

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

Refer to the following example code to create a workbook using GcExcel Java.

Java

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

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

## Open and Save Workbook

After a workbook is created, you can open the workbook to incorporate modifications, save the changes back to the workbook and protect it with a password in order to ensure security.

This topic includes the following tasks:

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

### Open a workbook

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

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

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

Refer to the following example code in order to open a workbook.

#### Java

```
// Opening a workbook
workbook.open("OpenWorkbook.xlsx");

// Opening a workbook with Import options

// Import only data from .xlsx document
XlsxOpenOptions options = new XlsxOpenOptions();
options.setImportFlags(EnumSet.of(ImportFlags.Data));
workbook.open("OpenWorkbookWithOptions.xlsx", options);

// Don't recalculate after opened.
options.setDoNotRecalculateAfterOpened(true);
```



**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.

While opening a password protected excel file (version 2013 or later) in GcExcel Java, the illegal key size exception will be thrown. This happens because Java compiler supports 128 bit key and doesn't support the 256 bit key (Excel

2013 or later versions use 256 bit key) by default. In such a scenario, users can [apply the JCE unlimited strength policy files](#) in order to resolve the issue.

## 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.

Java

```
// Saving an excel file (.xlsx document)
workbook.save("SaveExcel.xlsx");

// Saving an excel file with saveFileFormat
workbook.save("SaveExcelWithFormat.xlsx", SaveFileFormat.Xlsx);

// Saving an excel file while setting password
XlsxSaveOptions options = new XlsxSaveOptions();
options.setPassword("123456");
workbook.save("SaveExcelWithPassword.xlsx", options);
```

## Protect Workbook

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

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

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

### Protect Workbook using Password

GcExcel enables users to protect a workbook by encrypting it with a password. This is important when you have a business-critical workbook containing sensitive data that cannot be shared with everyone. You can secure a workbook using the **setPassword** ('**setPassword Method**' in the **on-line documentation**) method of **XlsxSaveOptions** ('**XlsxOpenOptions Class**' in the **on-line documentation**) class.

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

Java

```
// Saving an excel file while setting password
XlsxSaveOptions options = new XlsxSaveOptions();
options.setPassword("123456");
workbook.save("SaveExcelWithPassword.xlsx", options);
```

## Protect Workbook from Modification

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

The **Workbook ('Workbook Class' in the on-line documentation)** class provides two overloaded **protect ('protect() Method' in the on-line documentation)** methods, one of which takes password as a parameter. Both the methods have two optional parameters, **structure** and **windows**, which provide different types of modification protection when set.

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

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

Java

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Data
Object data = new Object[][] { { "Name", "City", "Sex", "Weight", "Height", "Age" },
    { "Bob", "NewYork", "male", 80, 180, 56 }, { "Betty", "NewYork", "female", 72,
168, 45 },
    { "Gary", "NewYork", "male", 71, 179, 50 }, { "Hunk", "Washington", "male", 80,
171, 59 },
    { "Cherry", "Washington", "female", 58, 161, 34 }, { "Coco", "Virginia",
"female", 58, 181, 45 },
    { "Lance", "Chicago", "female", 49, 160, 57 }, { "Eva", "Washington", "female",
71, 180, 81 } };

// Set data
worksheet.getRange("A1:G9").setValue(data);
//Protects the workbook with password so that other users cannot view hidden worksheets,
add, move, delete, hide, or rename worksheets.
workbook.protect("Ygs_87@ytr");
// Save workbook to xlsx
workbook.save("ProtectWorkbook.xlsx", SaveFileFormat.Xlsx);
```

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

Java

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Protects the workbook so that other users cannot view hidden worksheets, add,
// move, delete, hide, or rename worksheets.
workbook.protect();
```



```
// Saving workbook to.xlsx
workbook.save("1-ProtectWorkbook.xlsx", SaveFileFormat.Xlsx);
```

## Unprotect Workbook from Modification

A protected workbook can be unprotected to make modifications using the **Unprotect ('unprotect Method' in the on-line documentation)** method of the **Workbook ('Workbook Class' in the on-line documentation)** class, which removes the protection from a workbook.

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

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

C#

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Data
Object data = new Object[][] { { "Name", "City", "Sex", "Weight", "Height", "Age" },
    { "Bob", "NewYork", "male", 80, 180, 56 }, { "Betty", "NewYork", "female", 72,
168, 45 },
    { "Gary", "NewYork", "male", 71, 179, 50 }, { "Hunk", "Washington", "male", 80,
171, 59 },
    { "Cherry", "Washington", "female", 58, 161, 34 }, { "Coco", "Virginia",
"female", 58, 181, 45 },
    { "Lance", "Chicago", "female", 49, 160, 57 }, { "Eva", "Washington", "female",
71, 180, 81 } };
// Set data
worksheet.getRange("A1:G9").setValue(data);
workbook.protect("Ygs_87@ytr");
//Removes the above protection from the workbook
workbook.unprotect("Ygs_87@ytr");
// Save workbook to.xlsx
workbook.save("UnProtectWorkbook.xlsx", SaveFileFormat.Xlsx);
```

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

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

Java

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
workbook.protect();
// Removes the above protection from the workbook.
```

```
workbook.unprotect();  
// Saving workbook to xlsx  
workbook.save("2-UnprotectWorkbook.xlsx", SaveFileFormat.Xlsx);
```

## Cut or Copy Across Sheets

In GcExcel Java, you can cut or copy data across a range of cells or several worksheets without any hassle.

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.

Java

```
Object[][] data = new Object[][] { { 1 }, { 3 }, { 5 }, { 7 }, { 9 } };  
worksheet.getRange("A1:A5").setValue(data);  
  
// Copy across sheets  
IRange range = worksheet2.getRange("B7");  
worksheet.getRange("A5").copy(range);
```

### Cut across sheets

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

Java

```
IRange range1 = worksheet2.getRange("B3");  
  
// Cut across sheets  
worksheet.getRange("A3").cut(range1);
```

## Enable or Disable Calculation Engine

GcExcel Java provides you with comprehensive computing features via a built-in calculation engine that is capable of performing even the most complex operations on the data in the spreadsheets. This calculation engine can be integrated with spreadsheets to achieve the desired results with complete accuracy and within fraction of seconds.

Some of the advantages of using calculation engine are shared below:

### 1. Bulk Data Analysis:

Involves significantly less programming effort to handle complex spreadsheet calculations and display accurate results for effective analysis of tons of data.

## 2. Ease of Use:

Easy-to-configure calculation engine with the ability to quickly fetch data from cells within the spreadsheets and perform accurate calculations on the data.

## 3. Saves Time and Efforts:

Pre-defined functions and built-in methods to save implementation time and minimize programming efforts.

### Enable calculation engine

In order to enable the calculation engine, refer to the following example code.

Java

```
// Enable the calculation engine
worksheet.getRange("A1").setValue(1);
worksheet.getRange("A2").setFormula("=A1");
workbook.setEnableCalculation(true);

// Calculates formula while getting values. A2's value is 1
Object value1 = worksheet.getRange("A2").getValue();
```

### Disable calculation engine

In order to disable calculation engine, refer to the following example code.

Java

```
// Disable the calculation engine
workbook.setEnableCalculation(false);
worksheet.getRange("A1").setValue(1);
worksheet.getRange("A2").setFormula("=A1");

// A2's value is zero
Object value = worksheet.getRange("A2").getValue();
```

## Workbook Views

GcExcel Java enables users to customize the workbook appearance based on specific preferences.

You can use the **getBookView** ('**getBookView Method** in the on-line documentation) method of the **IWorkbook** ('**IWorkbook Interface** in the on-line documentation) interface in order to set the workbook display as per your choice.

The **IWorkbookView** ('**IWorkbookView Interface** in the on-line documentation) interface provides the following methods to allow users to further customize essential display settings in the workbook:

Methods	Descriptions
<b>getDisplayHorizontalScrollBar</b> ('getDisplayHorizontalScrollBar Method' in the on-line documentation) <b>setDisplayHorizontalScrollBar</b> ('setDisplayHorizontalScrollBar Method' in the on-line documentation)	Used to get or set the display of the horizontal scrollbar.
<b>getDisplayVerticalScrollBar</b> ('getDisplayVerticalScrollBar Method' in the on-line documentation) <b>setDisplayVerticalScrollBar</b> ('setDisplayVerticalScrollBar Method' in the on-line documentation)	Used to get or set the display of the vertical scrollbar.
<b>getDisplayWorkbookTabs</b> ('getDisplayWorkbookTabs Method' in the on-line documentation) <b>setDisplayWorkbookTabs</b> ('setDisplayWorkbookTabs Method' in the on-line documentation)	Used to get or set the display of the workbook tabs.
<b>getTabRatio</b> ('getTabRatio Method' in the on-line documentation) <b>setTabRatio</b> ('setTabRatio Method' in the on-line documentation)	Used to get or set the ratio of the width of the tab area (of the workbook) to the width of the horizontal scroll bar (of the worksheet). The value of TabRatio can be any number between 0 and 1. By default, if the TabRatio is not set, the value is 0.6.

In order to set the workbook view and customize other display settings, refer to the following code snippet.

Java

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

// Configure Workbook View Settings
IWorkbookView bookView = workbook.getBookView();
bookView.setDisplayVerticalScrollBar(false);
bookView.setDisplayWorkbookTabs = true;
bookView.setTabRatio(0.5);
```

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

## Comments

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

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

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

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

This is the highest sales figure for Quarter 1

This is the lowest sales figure for Quarter 1

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

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

### Add comment to a cell

In GcExcel Java, a cell comment instance is represented by the **IComment** ('IComment Interface' in the on-line documentation) interface. You can insert comment to a cell or a range of cells using the **addComment** ('addComment Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface. You can also set the text of the comment using the **setText** ('setText Method' in the on-line documentation) method of the **IComment** ('IComment Interface' in the on-line documentation) interface.

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

```
Java
// Add new comments
IComment commentC3 = worksheet.getRange("C3").addComment("Range C3's comment.");
IComment commentC4 = worksheet.getRange("C4").addComment("Range C4's comment.");
IComment commentC5 = worksheet.getRange("C5").addComment("Range C5's comment.");

// Change the text of a comment.
commentC3.setText("New Comment for Range C3.");
```

### Set comment layout

You can configure the layout of the comment added to an individual cell or a range of cells using **getShape** ('**getShape Method** in the on-line documentation') method of the **IComment** ('**IComment Interface** in the on-line documentation') interface.

Refer to the following example code to set comment layout.

Java

```
IComment commentC3 = worksheet.getRange("C3").addComment("Range C3's comment.");  
// Configure comment layout  
commentC3.getShape().getLine().getColor().setRGB(Color.GetGreen());  
commentC3.getShape().getLine().setWeight(7);  
commentC3.getShape().getLine().setStyle(LineStyle.ThickThin);  
commentC3.getShape().getLine().setDashStyle(LineDashStyle.Solid);  
commentC3.getShape().getFill().getColor().setRGB(Color.GetGray());  
commentC3.getShape().setWidth(100);  
commentC3.getShape().setHeight(200);  
commentC3.getShape().getTextFrame().getTextRange().getFont().setBold(true);  
commentC3.setVisible(true);
```

## Show/Hide comment

You can choose to keep comments hidden or visible by using the **setVisible** ('**setVisible Method** in the on-line documentation') method of the **IComment** ('**IComment Interface** in the on-line documentation') interface.

In order to show/hide comment added to a cell, refer to the following example code.

Java

```
// Add comment.  
IComment commentC3 = worksheet.getRange("C3").addComment("Range C3's comment.");  
// Show comment  
commentC3.setVisible(true);  
// Hide comment  
commentC3.setVisible(false);
```

## Author comments

You can represent the author of the comment by using the **getAuthor** ('**getAuthor Method** in the on-line documentation') method of the **IComment** ('**IComment Interface** in the on-line documentation') interface. Also, you can use this method to change the author of an existing comment.

In order to set comment author for a cell, refer to the following example code.

Java

```
// Set comment author  
workbook.setAuthor("joneshan");  
IWorksheet worksheet = workbook.getWorksheets().get(0);  
worksheet.getRange("C3").addComment("Range C3's comment.");  
// C3's comment author is "joneshan"  
String authorC3 = worksheet.getRange("C3").getComment().getAuthor();  
System.out.println(authorC3);
```

## Set rich text for comment

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

In order to set rich text for the comment, refer to the following example code.

Java

```
// Add a new comment  
IComment commentC3 = worksheet.getRange("C3").addComment("Range C3's comment.");  
  
// Configure the paragraph style using Rich Text and Text Frame.
```

```
commentC3.getShape().getTextFrame().getTextRange().getParagraphs().get(0).getFont().setBold(true);
commentC3.getShape().getTextFrame().getTextRange().getParagraphs().get(0).getRuns().add("ccc", 0);
commentC3.getShape().getTextFrame().getTextRange().getParagraphs().get(0).getRuns().add("eee", 1);
commentC3.getShape().getTextFrame().getTextRange().getParagraphs().get(0).getRuns().add("ddd");
commentC3.getShape().getTextFrame().getTextRange().getParagraphs().get(0).getRuns().get(2).getFont()
    .setItalic(true);
commentC3.getShape().getTextFrame().getTextRange().getParagraphs().get(0).getRuns().get(2).getFont().getColor()
    .setRGB(Color.GetGreen());
```

### Delete comment

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

In order to delete comment from a cell, refer to the following example code.

```
Java

// Add comments
worksheet.getRange("C3").addComment("Range C3's comment.");
worksheet.getRange("D4").addComment("Range D4's comment.");
worksheet.getRange("D5").addComment("Range D5's comment.");

// Delete a single cell comment
worksheet.getRange("D5").getComment().delete();

// Clear multiple comments from a range of cells
worksheet.getRange("C3:D4").clearComments();
```

## Hyperlinks

With GcExcel Java, you can insert and manage hyperlinks in the cells of the worksheets without any hassle.

Basically, a hyperlink in a cell refers to the hypertext link that is entered into a cell in order to assign data reference that point to another externally located file or a section within the document. Users can insert multiple hyperlinks in a worksheet or a cell range depending on the requirements.

In GcExcel Java, you can work with hyperlinks in the following ways:

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

### Add hyperlinks

Hyperlinks can be added via linking to an external file, linking to a webpage, linking to an email address and linking to a cell or a range of cells within the worksheet. You can insert hyperlinks using the **add** ('add Method' in the on-line documentation) method of the **IHyperlinks** ('IHyperlinks Interface' in the on-line documentation) interface.

In order to add hyperlinks to an external file, to a webpage, to a range within the worksheet and to an email address; refer to the following example code.

```
Java

// Add hyperlink to an external file
worksheet.getRange("A1:B2").getHyperlinks().add(worksheet.getRange("A1"),
    "C:\\Documents\\GcExcel\\GrapeCityDocumentsExcel\\Project\\Hyperlink\\SampleFile.xlsx",
```

```

null,
"link to SampleFile.xlsx file.",
"SampleFile.xlsx");

```

Java

```

// Add hyperlink to a webpage
worksheet.getRange("A1:B2").getHyperlinks().add(worksheet.getRange("A1"),
    "http://www.grapecity.com/", null, "open GrapeCity website.", "GrapeCity");

```

Java

```

// Add hyperlink to a range in this document
worksheet.getRange("A1:B2").getHyperlinks().add(worksheet.getRange("A1"),
    null, "Sheet1!$C$3:$E$4", "Go To sheet1 C3:E4", "");

```

Java

```

// Add hyperlink to a range in this document
worksheet.getRange("A1:B2").getHyperlinks().add(worksheet.getRange("A1"),
    "mailto:us.sales@grapecity.com",
    null,
    "Send an email to sales department",
    "Send To US Sales");

```

## Configure Hyperlinks

You can configure hyperlinks using the methods of the **IHyperlink** ('IHyperlink Interface' in the on-line documentation) interface.

1. The methods of the IHyperlink interface enables users to configure the hyperlink references. The table shared below illustrates some examples:

Link To	Address	SubAddress
External File	Example: "C:\Documents\GcExcel\GrapeCityDocumentsExcel\Project\Hyperlink\SampleFile.xlsx"	null
Webpage	Example: " <a href="https://www.grapecity.com/">https://www.grapecity.com/</a> "	null
A range in this document	Example: null	"Sheet1!\$C\$3:\$E\$4"
Email Address	Example: <a href="mailto:us.sales@grapecity.com">mailto:us.sales@grapecity.com</a>	null

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

## Delete Hyperlinks

The hyperlinks added in the cells can be deleted from the worksheet or in a specific cell range using the delete method. In order to remove hyperlinks, refer to the following example code.



## Java

```
// Delete hyperlinks
worksheet.getRange("A1:B2").getHyperlinks().add(worksheet.getRange("A1:A2"), null,
"Sheet1!$C$3:$E$4", "Go To sheet1 C3:E4", "");
worksheet.getRange("H5").getHyperlinks().add(worksheet.getRange("A1"),
"http://www.grapecity.com");
worksheet.getRange("J6").getHyperlinks().add(worksheet.getRange("A1"),
"http://www.grapecity.com");

// Delete hyperlinks in range A1:A2
worksheet.getRange("A1:A2").getHyperlinks().delete();

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

## Sort

For efficient data organization and quickly locate relevant information, GcExcel Java enables users to execute sorting operation on the data in the worksheets.

The **sort ('sort Method' in the on-line documentation)** method can be used to perform sorting based on a cell range, range by value, color or icon in a spreadsheet. The **apply ('apply Method' in the on-line documentation)** method is used to apply the selected sort state and display the sorted results.



**Note:** Sort operation can also be performed on merged cells, provided the cells to be merged are of the same size.

Different types of sorting available in GcExcel Java are explained below:

### Sort by value

Sort by value refers to the sorting operation that arranges the data in a particular order on the basis of the cell values. The **setOrientation** method is used to specify the orientation category for sorting, i.e, columns or rows.

In order to execute sort by value operation, refer to the following example code.

## Java

```
// Assigning values to the range
worksheet.getRange("A1").setValue(2);
worksheet.getRange("A2").setValue(1);
worksheet.getRange("A3").setValue(1);
worksheet.getRange("A4").setValue(3);
worksheet.getRange("B1").setValue("g");
worksheet.getRange("B2").setValue("z");
worksheet.getRange("B3").setValue("z1");
worksheet.getRange("B4").setValue("a");

// Apply Sort by value using IRange.Sort() method
worksheet.getRange("A1:B4").sort(worksheet.getRange("A1:A4"), SortOrder.Ascending,
```

```
SortOrientation.Columns);
```

## Sort by value for multiple columns

Sort by value refers to the sorting operation that is performed on multiple columns via a single line of code. You can also specify the orientation of columns in either ascending order or descending order.

In order to sort by value for multiple columns, refer to the following example code.

Java

```
// Assigning values to the range
worksheet.getRange("A1").setValue(2);
worksheet.getRange("A2").setValue(1);
worksheet.getRange("A3").setValue(1);
worksheet.getRange("A4").setValue(1);
worksheet.getRange("B1").setValue("g");
worksheet.getRange("B2").setValue("z");
worksheet.getRange("B3").setValue("a");
worksheet.getRange("B4").setValue("b");

// Sort by value, multiple column sort uses IRange.Sort() method
worksheet.getRange("A1:B4").sort(SortOrientation.Columns, false,
new ValueSortField[] { new ValueSortField(worksheet.getRange("A1:A2"),
SortOrder.Ascending),
new ValueSortField(worksheet.getRange("B1:B2"), SortOrder.Descending) });
```

## Custom sort

Sorting is a common task, but not all data types are meant to follow the common ascending and descending sort order rule. For instance, months cannot be sorted in a meaningful way when sorted alphabetically. In such a scenario, GcExcel Java allows users to execute a custom sort.

In order to implement custom sorting, refer to the following example code.

Java

```
// Assigning values to the range
worksheet.getRange("A1").setValue("test");
worksheet.getRange("A2").setValue(3);
worksheet.getRange("A3").setValue(5);
worksheet.getRange("A4").setValue(1);
worksheet.getRange("A5").setValue(2);
worksheet.getRange("A6").setValue(4);

// Define a custom sort value string
ValueSortField sortkey = new ValueSortField(worksheet.getRange("A1:A2"), "1,2,3");
worksheet.getRange("A2:A6").sort(SortOrientation.Columns, false, sortkey);
```

## Sort by interior

Sort by interior refers to the sorting operation which is performed on the basis of the interior color, pattern color, gradient color and gradient angle. However, interior sort cannot be executed on the basis of cell color.

In order to perform sort by interior operation, refer to the following example code.

Java

```
// Create a new workbook and add data
Workbook workbook = new Workbook();
Object data = new Object[][]{
    {"Name", "City", "Birthday", "Eye color", "Weight", "Height"},
    {"Richard", "New York", new GregorianCalendar(1968, 5, 8), "Blue", 67, 165},
    {"Nia", "New York", new GregorianCalendar(1972, 6, 3), "Brown", 62, 134},
    {"Jared", "New York", new GregorianCalendar(1964, 2, 2), "Hazel", 72, 180},
    {"Natalie", "Washington", new GregorianCalendar(1972, 7, 8), "Blue", 66, 163},
    {"Damon", "Washington", new GregorianCalendar(1986, 1, 2), "Hazel", 76, 176},
    {"Angela", "Washington", new GregorianCalendar(1993, 1, 15), "Brown", 68, 145}
};
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("A1:F7").setValue(data);
worksheet.getRange("A:F").setColumnWidth(15);

// Set color to the range
worksheet.getRange("F2").getInterior().setColor(Color.GetLightPink());
worksheet.getRange("F3").getInterior().setColor(Color.GetLightGreen());
worksheet.getRange("F4").getInterior().setColor(Color.GetLightPink());
worksheet.getRange("F5").getInterior().setColor(Color.GetLightGreen());
worksheet.getRange("F6").getInterior().setColor(Color.GetLightBlue());
worksheet.getRange("F7").getInterior().setColor(Color.GetLightPink());

// "F4" will at the top.
worksheet.getSort().getSortFields().add(new
CellColorSortField(worksheet.getRange("F2:F7"),
worksheet.getRange("F4").getDisplayFormat().getInterior(), SortOrder.Ascending));
worksheet.getSort().setRange(worksheet.getRange("A2:F7"));
worksheet.getSort().setOrientation(SortOrientation.Columns);
// Apply sort
worksheet.getSort().apply();
```

## Sort by font color

Sort by font color refers to the sorting operation which is executed on the basis of the cell's display format and font color. However, sorting operation is not performed on the basis of cell color.

In order to execute sort by font color, refer to the following example code.

Java

```
// Assigning values to the range
```

```
worksheet.getRange("A1").setValue(2);
worksheet.getRange("A2").setValue(1);
worksheet.getRange("A3").setValue(1);
worksheet.getRange("A4").setValue(3);
worksheet.getRange("B1").setValue(2);
worksheet.getRange("B2").setValue(1);
worksheet.getRange("B3").setValue(1);
worksheet.getRange("B4").setValue(3);

// Assigning color to the range
worksheet.getRange("B1").getFont().setColor(Color.FromArgb(0, 128, 0));
worksheet.getRange("B2").getFont().setColor(Color.FromArgb(128, 0, 0));
worksheet.getRange("B3").getFont().setColor(Color.FromArgb(0, 0, 128));
worksheet.getRange("B4").getFont().setColor(Color.FromArgb(128, 128, 0));

// Defining sort by color
worksheet.getSort().getSortFields().add(new
FontColorSortField(worksheet.getRange("B1:B4"),
worksheet.getRange("B1").getDisplayFormat().getFont().getColor(),
SortOrder.Descending));
worksheet.getSort().setRange(worksheet.getRange("A1:B4"));
worksheet.getSort().setOrientation(SortOrientation.Columns);
worksheet.getSort().apply();
```

### Sort by Icon

Sort by icon refers to the sorting operation that is performed on the basis of the cell's conditional format icons.

In order to execute sort by icon operation, refer to the following example code.

#### Java

```
// Assigning values to the range
worksheet.getRange("A1").setValue(2);
worksheet.getRange("A2").setValue(1);
worksheet.getRange("A3").setValue(1);
worksheet.getRange("A4").setValue(3);
worksheet.getRange("B1").setValue(2);
worksheet.getRange("B2").setValue(1);
worksheet.getRange("B3").setValue(1);
worksheet.getRange("B4").setValue(3);

// Defining sort by icon
IIconSetCondition iconset =
worksheet.getRange("B1:B4").getFormatConditions().addIconSetCondition();
iconset.setIconSet(workbook.getIconSets().get(IconSetType.Icon3TrafficLights1));
worksheet.getSort().getSortFields().add(new IconSortField(worksheet.getRange("B1:B4"),
workbook.getIconSets().get(IconSetType.Icon3TrafficLights1).get(0),
SortOrder.Ascending));
```

```
worksheet.getSort().setRange(worksheet.getRange("A1:B4"));  
worksheet.getSort().setOrientation(SortOrientation.Columns);  
worksheet.getSort().apply();
```

## Filter

While working with spreadsheets, it often becomes challenging to analyse, manipulate and manage tons of data quickly and efficiently.

Applying filters not just helps you in viewing only the necessary information but also lets you hide the rest of the data. When you set a specific filter condition on a worksheet, only the most relevant records (rows) will display that match to a certain criteria in a particular column.

In GcExcel Java, filters can be applied to a selected range of data. For instance, you can apply number filter from range D3 to I6 to display only those rows where the columns contain numeric data.

There are several types of range filters that can be used while performing different 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

In order to apply number filters displaying data that meets the specified criteria set on a column containing numeric data, refer to the following example code.

```
Java  
  
// Apply number filter  
worksheet.getRange("D3:I6").autoFilter(0, "<>2");
```

### Apply multi select filters

In order to apply multi select filters to filter data based on cell values with multiple selections, refer to the following example code.

```
Java  
  
// Apply filter condition - multi select  
worksheet.getRange("A1:E5").autoFilter(0, new Object[] { "$2", "$4" },  
AutoFilterOperator.Values);
```

### Apply text filters

In order to apply text filters displaying rows with cell values that either match to the specified text or regular expression value on which the filter is applied, refer to the following example code.

Java

```
// Apply filter condition - begin with "a".
worksheet.getRange("D3:I9").autoFilter(1, "a*");
```

## Apply date filters

In order to apply date filter condition to a range of cells displaying only the specific results falling within the given dates, refer to the following example code.

Java

```
// Apply filter using Date criteria
String criteria1 = new GregorianCalendar(1972, 6, 3).getTime().toString();
String criteria2 = new GregorianCalendar(1993, 1, 15).getTime().toString();
// Filter date between 1972.7.3 and 1993.2.15
worksheet.getRange("A1:F7").autoFilter(2, ">=" + criteria1, AutoFilterOperator.And, "<="
+ criteria2);
```

## Apply dynamic date filters

In order to apply dynamic date filters displaying results that match the specified date criteria based on the current system date that automatically gets updated everyday, refer to the following example code.

Java

```
// Apply filter condition - filter by yesterday
worksheet.getRange("D7:F18").autoFilter(2, DynamicFilterType.Yesterday,
AutoFilterOperator.Dynamic);
```

## Apply filters by cell colors

In order to apply filters by cell colors on a column showing results with cells having different fill shades, refer to the following example code.

Java

```
// Apply filter by cell color
worksheet.getRange("A1:A6").autoFilter(0, Color.FromArgb(255, 255, 0),
AutoFilterOperator.CellColor);
```

## Apply filters by no fill

In order to apply filters by no fill on a column in order to filter results based on cells having no fill color, refer to the following example code.

Java

```
// Apply filter by no fill
worksheet.getRange("A1:A6").autoFilter(0, null, AutoFilterOperator.NoFill);
```

## Apply filters by icon

In order to apply filters by icon that filters results via possessing a specific icon in the cells, refer to the following example code.

```
Java

// Apply filter by icon
worksheet.getRange("A1:A10").autoFilter(0,
workbook.getIconSets().get(IconSetType.Icon5ArrowsGray).get(0), AutoFilterOperator.Icon);
```

## Apply filters by no icon

In order to apply filters by no icon that displays results where cells do not have an icon, refer to the following example code.

```
Java

// Apply filter by no icon
worksheet.getRange("A2:A10").autoFilter(0, null, AutoFilterOperator.NoIcon);
```


## Group

When you have extensive amount of data in spreadsheets, it becomes difficult to process relevant information for business management, analytics and data visualization.

GcExcel Java allows you to summarize bulk data in groups so that complex spreadsheets are easier to read, navigate, and manipulate. Each group in GcExcel Java is distinguished with a group header row next to it which can be used to display or hide information as and when required. In order to expand a group to display rows and columns that have been hidden, you can set the **setShowDetail** ('**setShowDetail Method**' in the **on-line documentation**) method of the **IRange** ('**IRange Interface**' in the **on-line documentation**) Interface to boolean true and in order to collapse the expanded rows or columns, you can set the **setShowDetail** method of the **IRange** Interface to boolean false.

Applying grouping in a spreadsheet involves the following tasks:

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

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

## Create Row or Column Group

With GcExcel Java, you can apply grouping on rows and columns of a spreadsheet by referring to the following tasks.

- 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** ('group Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface and specifying the rows you want to apply grouping on.

In order to apply row grouping in a worksheet, refer to the following example code.

Java

```
// 1:20 rows' outline level will be 2.
worksheet.getRange("1:20").group();
// 1:10 rows' outline level will be 3.
worksheet.getRange("1:10").group();
```

## Apply column grouping

You can apply column grouping by using the **group** ('group Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface and specifying the columns you want to apply grouping on.

In order to apply column grouping in a worksheet, refer to the following example code.

Java

```
// A:N columns' outline level will be 2.
worksheet.getRange("A:N").group();
// A:E columns' outline level will be 3.
worksheet.getRange("A:E").group();
```

## Set outline level for rows and columns

While performing the grouping operation 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 **setOutlineLevel** ('setOutlineLevel Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface. You can also choose to display specified levels of row or column groups using the methods of the **IOutline** ('IOutline Interface' in the on-line documentation) interface.

In order to set the Outline level for rows and columns, refer to the following example code.

Java

```
// 1:20 rows' outline level will be 3.
worksheet.getRange("1:20").setOutlineLevel(3);

// A:E columns' outline level will be 4.
```



```
worksheet.getRange("A:E").setOutlineLevel(4);
```

You can use the methods of the **IOutline** ('**IOutline Interface**' in the on-line documentation) interface to figure out whether the summary column is present at the left position or right position of the column groups or whether the summary row is above or below the row groups, respectively.

## Remove a Group

You can remove a group in GcExcel Java by referring to the following tasks in your worksheet.

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

### Ungroup rows and columns

You can ungroup the grouped rows or columns if you no longer want the information to be organized. Also, you can increment or decrement the outline level for the specified rows or columns using the **group** ('**group Method**' in the on-line documentation) method and **ungroup** ('**ungroup Method**' in the on-line documentation) method of the **IRange** ('**IRange Interface**' in the on-line documentation) interface respectively.

In order to ungroup row and column in a worksheet, refer to the following example code.

Java

```
// Row ungrouping
// 1:20 rows' outline level will be 2.
worksheet.getRange("1:20").group();
// 1:10 rows' outline level will be 3.
worksheet.getRange("1:10").group();
// 1:5 rows' outline level will be 2.
worksheet.getRange("1:5").ungroup();
// 1:5 rows' outline level will be 1.
worksheet.getRange("1:5").ungroup();

// Column ungrouping
// A:I columns grouping.
worksheet.getRange("A:I").group();
// A:I columns ungrouping
worksheet.getRange("A:I").ungroup();
```

### Clear outline

You can clear the outline level of the specified rows or columns using the **clearOutline** ('**clearOutline Method**' in the on-line documentation) method of the **IRange** ('**IRange Interface**' in the on-line documentation) interface.

In order to clear outline in a worksheet, refer to the following example code.

Java

```
// 1:20 rows' outline level will be 2.
worksheet.getRange("1:20").group();
// 1:10 rows' outline level will be 3.
worksheet.getRange("1:10").group();

// Clear outline
// 12:20 rows' outline level will be 1.
worksheet.getRange("12:20").clearOutline();
```

## Collapse a group

You can collapse a group by using the **setShowDetail** ('**setShowDetail Method** in the on-line documentation') method of the **IRange** ('**IRange Interface** in the on-line documentation') interface and setting it to boolean false.

In order to collapse a group in a worksheet, refer to the following example code.

```
Java

// 1:20 rows' outline level will be 2.
worksheet.getRange("1:20").group();
// 1:10 rows' outline level will be 3.
worksheet.getRange("1:10").group();

// Collapse the group
// 1:10 rows will be collapsed.
worksheet.getRange("11:11").setShowDetail(false);
```

## Summary Row

Summary rows are group header rows displaying the group names with complete information about an existing group. Corresponding to each group, a summary row is automatically created when a user executes the grouping operation in a spreadsheet.

With GcExcel Java, you can modify and customize the summary row based on your requirements by using the **setSummaryRow** ('**setSummaryRow Method** in the on-line documentation') method of the **IOutline** ('**IOutline Interface** in the on-line documentation') interface.

In order to set summary row, refer to the following example code.

```
Java

// Summary
worksheet.getOutline().setSummaryRow(SummaryRow.Above);

// Summary row will be row 4.
worksheet.getRange("5:20").group();
// Summary row will be row 14.
worksheet.getRange("15:20").group();
```

## Outline Subtotals

Sometimes, the amount of data in a spreadsheet is so huge that it becomes difficult to analyze it. In such cases, you can apply outline to organize the sorted data into groups. The outline data can be collapsed or expanded to hide or show specific groups from viewing. You can also derive meaningful and summarized insights from outline data by applying the subtotals to the grouped values.

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

### Create Outline Subtotals

The outline subtotals are created using the **subtotal ('subtotal Method' in the on-line documentation)** method of **IRange ('IRange Interface' in the on-line documentation)** interface. The method provides different parameters to group by fields, assign subtotal function, replace existing subtotals, add page breaks and place summary data.

The below sample data is used to create outline subtotals:

Java

```
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Defining data in the range
worksheet.getRange("A1:C20")
    .setValue(new Object[][] { { "Item", "Units", "Unit Price" }, { "Pen Set", 62,
4.99 },
        { "Binder", 29, 1.99 }, { "Pen Set", 55, 12.49 }, { "Binder", 81, 19.99
},
        { "Pen Set", 42, 23.95 }, { "Pencil", 35, 4.99 }, { "Desk", 3, 275 }, {
"Desk", 2, 125 },
        { "Pencil", 7, 1.29 }, { "Pen Set", 16, 15.99 }, { "Pen", 76, 1.99 }, {
"Binder", 28, 8.99 },
        { "Binder", 57, 19.99 }, { "Pen", 64, 8.99 }, { "Pencil", 14, 1.29 }, {
"Pen", 15, 19.99 },
        { "Binder", 11, 4.99 }, { "Pen Set", 96, 4.99 }, { "Binder", 94, 19.99 }
});
```

Refer to the below example code to create outline subtotals.

Java

```
IWorksheet _worksheet = workbook.getWorksheets().get(0);

// Sort by value, use Sort() method.
_worksheet.getRange("A2:C20").sort(_worksheet.getRange("A2:A20"), null,
SortOrientation.Columns);

// Create groups and sub-total the grouped values using Subtotal() method
_worksheet.getRange("A1:D20").subtotal(1, ConsolidationFunction.Sum, new int[] { 2, 3
});
```

```
// Save workbook
workbook.save("391-CreateSubtotals.xlsx", SaveFileFormat.Xlsx);
```

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

## Remove Outline Subtotals

The outline subtotals can be removed using the **removeSubtotal** ('**removeSubtotal Method**' in the **on-line documentation**) method of the **IRange** interface.

Refer to the below example code to remove outline subtotals.

Java

```
Workbook workbook = CreateSubtotals();
IWorksheet _worksheet = workbook.getWorksheets().get(0);

// Remove Subtotals, pass the cell range inclusive of the subtotal/total rows
_worksheet.getRange("A1:C26").removeSubtotal();

// Save workbook
workbook.save("392-RemoveSubtotals.xlsx", SaveFileFormat.Xlsx);
```

## Outline Column

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

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

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

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

## Using Code

Refer to the below example code to create outline column.

C#

```
// create a new workbook
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Set data.
Object[][] data = new Object[][] { { "Preface", "1", 1, 0 }, { "Java SE5 and SE6",
"1.1", 2, 1 },
{ "Java SE6", "1.1.1", 2, 2 }, { "The 4th edition", "1.2", 2, 1 }, { "Changes",
"1.2.1", 3, 2 },
{ "Note on the cover design", "1.3", 4, 1 }, { "Acknowledgements", "1.4", 4, 1 },
{ "Introduction", "2", 9, 0 }, { "Prerequisites", "2.1", 9, 1 }, { "Learning Java",
"2.2", 10, 1 },
{ "Goals", "2.3", 10, 1 }, { "Teaching from this book", "2.4", 11, 1 },
{ "JDK HTML documentation", "2.5", 11, 1 }, { "Exercises", "2.6", 12, 1 },
{ "Foundations for Java", "2.7", 12, 1 }, { "Source code", "2.8", 12, 1 },
{ "Coding standards", "2.8.1", 14, 2 }, { "Errors", "2.9", 14, 1 },
{ "Introduction to Objects", "3", 15, 0 }, { "The progress of abstraction", "3.1", 15,
1 },
{ "An object has an interface", "3.2", 17, 1 }, { "An object provides services", "3.3",
18, 1 },
{ "The hidden implementation", "3.4", 19, 1 }, { "Reusing the implementation", "3.5",
20, 1 },
{ "Inheritance", "3.6", 21, 1 }, { "Is-a vs. is-like-a relationships", "3.6.1", 24, 2
},
{ "Interchangeable objects with polymorphism", "3.7", 25, 1 },
{ "The singly rooted hierarchy", "3.8", 28, 1 }, { "Containers", "3.9", 28, 1 },
{ "Parameterized types (Generics)", "3.10", 29, 1 }, { "Object creation & lifetime",
"3.11", 30, 1 },
{ "Exception handling: dealing with errors", "3.12", 31, 1 },
{ "Concurrent programming", "3.13", 32, 1 }, { "Java and the Internet", "3.14", 33, 1
},
{ "What is the Web?", "3.14.1", 33, 2 }, { "Client-side programming", "3.14.2", 34, 2
},
{ "Server-side programming", "3.14.3", 38, 2 }, { "Summary", "3.15", 38, 1 } };
worksheet.getRange("A1:C38").setValue(data);

// Set ColumnWidth.
worksheet.getRange("A:A").setColumnWidthInPixel(310);
```

```
worksheet.getRange("B:C").setColumnWidthInPixel(150);

// Set IndentLevel.
for (int i = 0; i < data.length; i++) {
    worksheet.getRange(i, 0).setIndentLevel((int) data[i][3]);
}
// Show the summary row above the detail rows.
worksheet.getOutline().setSummaryRow(SummaryRow.Above);

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


// Set outline column, the corresponding row outlines will also be automatically
// created.
worksheet.getOutlineColumn().setColumnIndex(0);
worksheet.getOutlineColumn().setShowCheckBox(true);
worksheet.getOutlineColumn().setShowImage(true);
worksheet.getOutlineColumn().setMaxLevel(2);

worksheet.getOutlineColumn().getImages()
    .add(new ImageSource(new FileInputStream("archiverFolder.png"), ImageType.PNG));
worksheet.getOutlineColumn().getImages()
    .add(new ImageSource(new FileInputStream("newFolder.png"), ImageType.PNG));
worksheet.getOutlineColumn().getImages()
    .add(new ImageSource(new FileInputStream("docFile.png"), ImageType.PNG));
worksheet.getOutlineColumn()
    .setCollapseIndicator(new ImageSource(new FileInputStream("decreaseIndicator.png"),
ImageType.PNG));
worksheet.getOutlineColumn()
    .setExpandIndicator(new ImageSource(new FileInputStream("increaseIndicator.png"),
ImageType.PNG));

worksheet.getOutlineColumn().setCheckStatus(0, true);
worksheet.getOutlineColumn().setCollapsed(1, true);

// MSExcel does not support the outline column, so when exporting to the excel file,
// The checkbox, level images, expand&collapse images are not visible.
// But the data is seen with heirarchical structure.

// save to an excel and PDF file
workbook.save("outlinecolumn.pdf");
workbook.save("outlinecolumn.xlsx");
```

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

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

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

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

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

## Conditional Formatting

GcExcel Java enables users to highlight useful information in rows or columns of a worksheet with the help of conditional formatting rules for a single cell or a range of cells based on cell values.

In case the format condition is same as the cell value, it is assumed to be true and the cell is formatted as per the applied rule.

For instance, let's take an example of a scenario wherein you want to show a specific cell or a cell range in italic font style if the cell value is lower than 90. For achieving this, you can apply a conditional formatting rule that changes the cell format if the desired condition is met. Note that the other cells will be displayed in the default format of the cells in the spreadsheet i.e. general format.

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 can be represented using the methods of the **IRange ('IRange Interface' in the on-line documentation)** interface.

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

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

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

## Cell Value Rule

The cell value rule compares values entered in the cells with the condition specified in the conditional formatting rule. In order to add a cell value rule, you can use the **setNumberFormat ('setNumberFormat Method' in the on-line documentation)** method of the **IFormatCondition ('IFormatCondition Interface' in the on-line documentation)** 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.

Java

```
// Assigning values using Object
Object[][] data = new Object[][]
{
    { 1 },
    { 3 },
    { 5 },
    { 7 },
    { 9 }
};
worksheet.getRange("A1:A5").setValue(data);

// Defining the Cell Value Rule
IFormatCondition condition =
(IFormatCondition)
worksheet.getRange("A1:A5").getFormatConditions().add(FormatConditionType.
CellValue, FormatConditionOperator.Between, 1, 5);

condition.setNumberFormat("0.000");
```

## Date Occurring Rule

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

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

Java

```
// Adding Date Occuring Rules
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
IFormatCondition condition =
    (IFormatCondition)worksheet.getRange("A1:A4").getFormatConditions().add(FormatConditionType.TimePeriod);
condition.setDateOperator(TimePeriods.Yesterday);
condition.getInterior().setColor(Color.FromArgb(128,0,128));

Calendar now=Calendar.getInstance();

worksheet.getRange("A1").setValue(now.getTime());
now.add(Calendar.DAY_OF_YEAR, -1);
worksheet.getRange("A2").setValue(now.getTime());
now.add(Calendar.DAY_OF_YEAR, -1);
worksheet.getRange("A3").setValue(now.getTime());
now.add(Calendar.DAY_OF_YEAR, 3);
worksheet.getRange("A4").setValue(now.getTime());
```

## Average Rule

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

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

Java

```
// Adding average rule
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("A1").setValue(1);
worksheet.getRange("A2").setValue(2);
worksheet.getRange("A3").setValue(3);
worksheet.getRange("A4").setValue(4);
worksheet.getRange("A5").setValue(60000000);

IAboveAverage averageCondition =
    worksheet.getRange("A1:A5").getFormatConditions().addAboveAverage();
averageCondition.setAboveBelow(AboveBelow.AboveAverage);
averageCondition.setNumStdDev(2);
averageCondition.setNumberFormat("0.00");
```

## Color Scale Rule

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

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

Java

```
// Adding Color Scale Rule
IColorScale twoColorScaleRule =
worksheet.getRange("A2:E2").getFormatConditions().addColorScale(ColorScaleType.TwoColorScale);

worksheet.getRange("A2").setValue(1);
worksheet.getRange("B2").setValue(2);
worksheet.getRange("C2").setValue(3);
worksheet.getRange("D2").setValue(4);
worksheet.getRange("E2").setValue(5);

twoColorScaleRule.getColorScaleCriteria().get(0).setType(ConditionValueTypes.Number);
twoColorScaleRule.getColorScaleCriteria().get(0).setValue(1);
twoColorScaleRule.getColorScaleCriteria().get(0).getFormatColor().setColor(Color.FromArgb(255,0,0));

twoColorScaleRule.getColorScaleCriteria().get(1).setType(ConditionValueTypes.Number);
twoColorScaleRule.getColorScaleCriteria().get(1).setValue(5);
twoColorScaleRule.getColorScaleCriteria().get(1).getFormatColor().setColor(Color.FromArgb(0,255,0));
```

## Data Bar Rule

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

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

Java

```
// Adding Data Bar Rule
Object[][] data=new Object[][]
{
    {1},
    {2},
    {3},
    {4},
    {5}
};
worksheet.getRange("A1:A5").setValue(data);

IDataBar dataBar = worksheet.getRange("A1:A5").getFormatConditions().addDatabar();

dataBar.getMinPoint().setType(ConditionValueTypes.LowestValue);
dataBar.getMinPoint().setValue(null);
```

```
dataBar.getMaxPoint().setType(ConditionValueTypes.HighestValue);
dataBar.getMaxPoint().setValue(null);

dataBar.setBarFillType(DataBarFillType.Solid);
dataBar.getBarColor().setColor(Color.GetGreen());
dataBar.setDirection(DataBarDirection.Context);
dataBar.getAxisColor().setColor(Color.GetRed());
dataBar.setAxisPosition(DataBarAxisPosition.Automatic);
dataBar.getNegativeBarFormat().setBorderColorType(DataBarNegativeColorType.Color);
dataBar.getNegativeBarFormat().getBorderColor().setColor(Color.FromArgb(128,0,212));
dataBar.getNegativeBarFormat().setColorType(DataBarNegativeColorType.Color);
dataBar.getNegativeBarFormat().getColor().setColor(Color.FromArgb(128,0,240));
dataBar.setShowValue(false);
```

## Top Bottom Rule

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

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

- Top 10
- Top 10%
- Bottom 10
- Bottom 10%
- Above Average
- Below Average

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

### Java

```
// Adding Top Bottom Rule
Object[][] data=new Object[][]
{
    {1},
    {2},
    {3},
    {4},
    {5}
};
worksheet.getRange("A1:A5").setValue(data);

ITop10 condition = worksheet.getRange("A1:A5").getFormatConditions().addTop10();
condition.setTopBottom(TopBottom.Top);
condition.setRank(50);
condition.setPercent(true);
condition.getInterior().setColor(Color.FromArgb(128,0,128));
```

## Unique Rule

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

Unique rule can be added using the methods of the **IUniqueValues** ('**IUniqueValues Interface**' in the **on-line documentation**) interface.

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

Java

```
// Adding Unique Rule
Object[][] data = new Object[][]
{
    { 1 },
    { 2 },
    { 1 },
    { 3 },
    { 4 }
};
worksheet.getRange("A1:A5").setValue(data);

IUniqueValues condition2 =
worksheet.getRange("A1:A5").getFormatConditions().addUniqueValues();
condition2.setDupeUnique(DupeUnique.Unique);
condition2.getFont().setName("Arial");
```

## Icon Sets Rule

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

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

Java

```
// Adding Icon Sets Rule
IIconSetCondition condition =
worksheet.getRange("A1:A5").getFormatConditions().addIconSetCondition();
condition.setIconSet(workbook.getIconSets().get(IconSetType.Icon3Symbols));
condition.getIconCriteria().get(1).setOperator(FormatConditionOperator.GreaterEqual);
condition.getIconCriteria().get(1).setValue(50);
condition.getIconCriteria().get(1).setType(ConditionValueTypes.Percent);
condition.getIconCriteria().get(2).setOperator(FormatConditionOperator.GreaterEqual);
condition.getIconCriteria().get(2).setValue(70);
condition.getIconCriteria().get(2).setType(ConditionValueTypes.Percent);
```

```
worksheet.getRange("A1").setValue(1);
worksheet.getRange("A2").setValue(2);
worksheet.getRange("A3").setValue(3);
worksheet.getRange("A4").setValue(4);
worksheet.getRange("A5").setValue(5);
```

## Expression Rule

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

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

Java

```
// Adding Expression Rule
Object[][] data = new Object[][]
{
    { 1, 2 },
    { 0, 1 },
    { 0, 0 },
    { 0, 3 },
    { 4, 5 }
};

worksheet.getRange("A1:B5").setValue(data);

IFormatCondition condition = (IFormatCondition)
worksheet.getRange("B1:B5").getFormatConditions().add(FormatConditionType.Expression,
FormatConditionOperator.None, "=A1", null);
condition.getInterior().setColor(Color.FromArgb(255, 0, 0));
```

## Data Validations

GcExcel Java allows users to validate cell data via putting a control on its data type, information format and value. You can create separate validation scenarios for a single cell or a range of cells based on your requirements.

With the help of data validation feature, you can execute the following operations in the spreadsheets :

- Make a list of entries via verifying the values that users are allowed to enter in the cells.
- Evoke relevant pop-up messages to validate the description of data values that users can enter in a cell.
- Check whether the entry in a specific cell or a range of cells is accurate or not. This can be done based on the calculated values evaluated on other cells.
- Configure a range of values (numeric or alphabetic) that are allowed to be entered in a single cell or a range of cells.
- Show alert messages when users enter invalid data in the cell.

You can use the data validation feature in GcExcel Java 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 apply data validation to restrict and verify the data entered in a single cell or a range of cells in a worksheet.

Only one validation rule should be applied for a cell. One cell cannot have multiple validation rules applied to it.

In case you try to validate a cell that already has a validation rule, an exception will be thrown.

If you want to know whether a cell range already contains the validation rule, you can use the methods of the **IRange ('IRange Interface' in the on-line documentation)** interface. If all the cells in a range possess the same validation rule applied to them, you can check it using the methods of the IRange interface.

Shared below is a list of data validations operations that can be implemented in GcExcel Java.

- **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.

Java

```
// Add whole number validation
worksheet.getRange("D2:E5").getValidation().add(ValidationType.Whole, ValidationAlertStyle.Stop,
    ValidationOperator.Between, 3, 8);
IValidation validation = worksheet.getRange("D2:E5").getValidation();
validation.setIgnoreBlank(true);
validation.setInputTitle("Tips");
validation.setInputMessage("Input a value between 3 and 8, please");
validation.setErrorTitle("Error");
validation.setErrorMessage("input value does not between 3 and 8");
validation.setShowInputMessage(true);
validation.setShowError(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.

Java

```
// Add decimal validation
worksheet.getRange("C2:E4").getValidation().add(ValidationType.Decimal, ValidationAlertStyle.Stop,
    ValidationOperator.Between, 111.5, 72.3);
```

## 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.

Java

```
worksheet.getRange("A1").setValue("aaa");
worksheet.getRange("A2").setValue("bbb");
worksheet.getRange("A3").setValue("ccc");

// Use cell reference.
worksheet.getRange("C2:E4").getValidation().add(ValidationType.List, ValidationAlertStyle.Stop,
    ValidationOperator.Between, "=$a$1:$a$3", null);

// Or use string.
//
this._worksheet.getRange("C2:E4").getValidation().add(ValidationType.List, ValidationAlertStyle.Stop,
// ValidationOperator.Between, "aaa, bbb, ccc", null);

IValidation validation = worksheet.getRange("C2:E4").getValidation();
validation.setInCellDropdown(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.

Java

```
worksheet.getRange("C2:E4").getValidation().add(ValidationType.Date, ValidationAlertStyle.Stop,
    ValidationOperator.Between, new GregorianCalendar(2015, 11, 13), new GregorianCalendar(2015,
11, 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.

Java

```
Calendar time1 = new GregorianCalendar(1899, 11, 30, 13, 30, 0);
Calendar time2 = new GregorianCalendar(1899, 11, 30, 18, 30, 0);
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("hh:mm:ss");
worksheet.getRange("C2:E4").getValidation().add(ValidationType.Time, ValidationAlertStyle.Stop,
```



```
ValidationOperator.Between, simpleDateFormat.format(time1.getTime()),  
simpleDateFormat.format(time2.getTime()));
```

### 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.

Java

```
worksheet.getRange("C2:E4").getValidation().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.

Java

```
worksheet.getRange("A2").setValue(1);  
worksheet.getRange("A3").setValue(2);  
worksheet.getRange("C2").setValue(1);  
  
// While using custom validation, validationOperator and formula2 parameters will be ignored even if  
// you have provided.  
worksheet.getRange("A2:A3").getValidation().add(ValidationType.Custom,  
ValidationAlertStyle.Information,  
ValidationOperator.Between, "=C2", null);
```

## Delete Validation

You can delete an existing validation rule applied to a cells or range of cells in a worksheet using the **delete ('delete Method' in the on-line documentation)** method of the **IValidation ('IValidation Interface' in the on-line documentation)** interface.

You can use the following example code to delete an existing validation rule which was previously applied to a cell or a range of cells in a worksheet.

Java

```
Calendar time1 = new GregorianCalendar(1899, 11, 30, 13, 30, 0);  
Calendar time2 = new GregorianCalendar(1899, 11, 30, 18, 30, 0);  
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("hh:mm:ss");  
  
// Add Validation  
worksheet.getRange("A1:A2").getValidation().add(ValidationType.Date,  
ValidationAlertStyle.Stop,  
ValidationOperator.Between, simpleDateFormat.format(time1.getTime()),  
simpleDateFormat.format(time2.getTime()));
```

```
// Delete validation.  
worksheet.getRange("A1:A2").getValidation().delete();
```

## Modify Validation

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

- Use the **setType** ('**setType Method** in the on-line documentation') method, the **setAlertStyle** ('**setAlertStyle Method** in the on-line documentation') method and the **setOperator** ('**setOperator Method** in the on-line documentation') method of the **IVValidation** ('**IVValidation Interface** in the on-line documentation') interface.
- Use **delete** ('**delete Method** in the on-line documentation') method of the **IVValidation** interface to first delete validation rule and then use the **add** ('**add Method** in the on-line documentation') method to add the new rule.

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

```
Java  
  
Calendar time1 = new GregorianCalendar(1899, 11, 30, 13, 30, 0);  
Calendar time2 = new GregorianCalendar(1899, 11, 30, 18, 30, 0);  
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("hh:mm:ss");  
  
// Add Validation  
worksheet.getRange("A1:A2").getValidation().add(ValidationType.Date,  
ValidationAlertStyle.Stop,  
ValidationOperator.Between, simpleDateFormat.format(time1.getTime()),  
simpleDateFormat.format(time2.getTime()));  
  
// Modify validation.  
worksheet.getRange("A1:A2").getValidation().setType(ValidationType.Time);  
worksheet.getRange("A1:A2").getValidation().setAlertStyle(ValidationAlertStyle.Warning);  
worksheet.getRange("A1:A2").getValidation().setOperator(ValidationOperator.NotBetween);
```

## Data Binding

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

### Sheet Binding

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

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

To bind the data source fields to sheet columns manually, you can set the **setAutoGenerateColumns** method of

IWorksheet interface to false and use the **setBindingPath** method of **IRange** interface to set the binding path of the data source field to the sheet columns.

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

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

Java

```
public class SheetBinding {

    // create a new workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // create datasource
    SalesData datasource = new SalesData();
    datasource.records = new ArrayList<SalesRecord>();

    // Add data
    SalesRecord record1 = new SalesRecord();
    record1.area = "NorthChina";
    record1.salesman = "Hellen";
    record1.product = "Apple";
    record1.productType = "Fruit";
    record1.sales = 120;
    datasource.records.add(record1);

    SalesRecord record2 = new SalesRecord();
    record2.area = "NorthChina";
    record2.salesman = "Hellen";
    record2.product = "Banana";
    record2.productType = "Fruit";
    record2.sales = 143;
    datasource.records.add(record2);

    SalesRecord record3 = new SalesRecord();
    record3.area = "NorthChina";
    record3.salesman = "Hellen";
    record3.product = "Kiwi";
    record3.productType = "Fruit";
    record3.sales = 322;
    datasource.records.add(record3);

    // Set AutoGenerateColumns to false
    worksheet.setAutoGenerateColumns(false);
```

```
// Bind columns manually
worksheet.getRange("A:A").getEntireColumn().setBindingPath("area");
worksheet.getRange("B:B").getEntireColumn().setBindingPath("salesman");
worksheet.getRange("C:C").getEntireColumn().setBindingPath("product");
worksheet.getRange("D:D").getEntireColumn().setBindingPath("productType");
worksheet.getRange("E:E").getEntireColumn().setBindingPath("sales");

// Set data source
worksheet.setDataSource(datasource.records);

// save to an excel file
workbook.save("SheetBinding.xlsx");
}

public static class SalesRecord {
    public int sales;
    public String productType;
    public String product;
    public String salesman;
    public String area;
}

public static class SalesData {
    public ArrayList<SalesRecord> records;
}
```

## Cell Binding

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

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

Refer to the below example code to bind datasource to cells.

### Java

```
public static void CellBinding {

    // create a new workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Add Data
    SalesRecord record = new SalesRecord();
    record.area = "NorthChina";
    record.salesman = "Hellen";
}
```

```
record.product = "Apple";
record.productType = "Fruit";
record.sales = 120;

// Set binding path for cells
worksheet.getRange("A1").setBindingPath("area");
worksheet.getRange("B2").setBindingPath("salesman");
worksheet.getRange("C2").setBindingPath("product");
worksheet.getRange("D3").setBindingPath("productType");

// Set data source
worksheet.setDataSource(record);

// save to an excel file
workbook.save("CellBinding.xlsx");
}

public static class SalesRecord {
    public int sales;
    public String productType;
    public String product;
    public String salesman;
    public String area;
}
```

## Table Binding

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

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

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

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

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

Java

```
public class TableBinding {

    // create a new workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
```

```
IWorksheet worksheet = workbook.getWorksheets().get(0);

// create datasource
SalesData datasource = new SalesData();
datasource.records = new ArrayList<SalesRecord>();

// Add data
SalesRecord record1 = new SalesRecord();
record1.area = "NorthChina";
record1.salesman = "Hellen";
record1.product = "Apple";
record1.productType = "Fruit";
record1.sales = 120;
datasource.records.add(record1);

SalesRecord record2 = new SalesRecord();
record2.area = "NorthChina";
record2.salesman = "Hellen";
record2.product = "Banana";
record2.productType = "Fruit";
record2.sales = 143;
datasource.records.add(record2);

SalesRecord record3 = new SalesRecord();
record3.area = "NorthChina";
record3.salesman = "Hellen";
record3.product = "Kiwi";
record3.productType = "Fruit";
record3.sales = 322;
datasource.records.add(record3);

// Add a table
ITable table = worksheet.getTables().add(worksheet.getRange("B2:F5"), true);

// Set not to auto generate table columns
table.setAutoGenerateColumns(false);

// Set table binding path
table.setBindingPath("records");

// Set table column data field
table.getColumns().get(0).setDataField("area");
table.getColumns().get(1).setDataField("salesman");
table.getColumns().get(2).setDataField("product");
table.getColumns().get(3).setDataField("productType");
table.getColumns().get(4).setDataField("sales");

// Set custom object as data source
```

```
worksheet.setDataSource(datasource);

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

}

public static class SalesRecord {
    public int sales;
    public String productType;
    public String product;
    public String salesman;
    public String area;
}

public static class SalesData {
    public ArrayList<SalesRecord> records;
}
```

### Limitation

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

## Digital Signatures

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

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

## Signature Lines

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

### Add Signature Line

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

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

#### Java

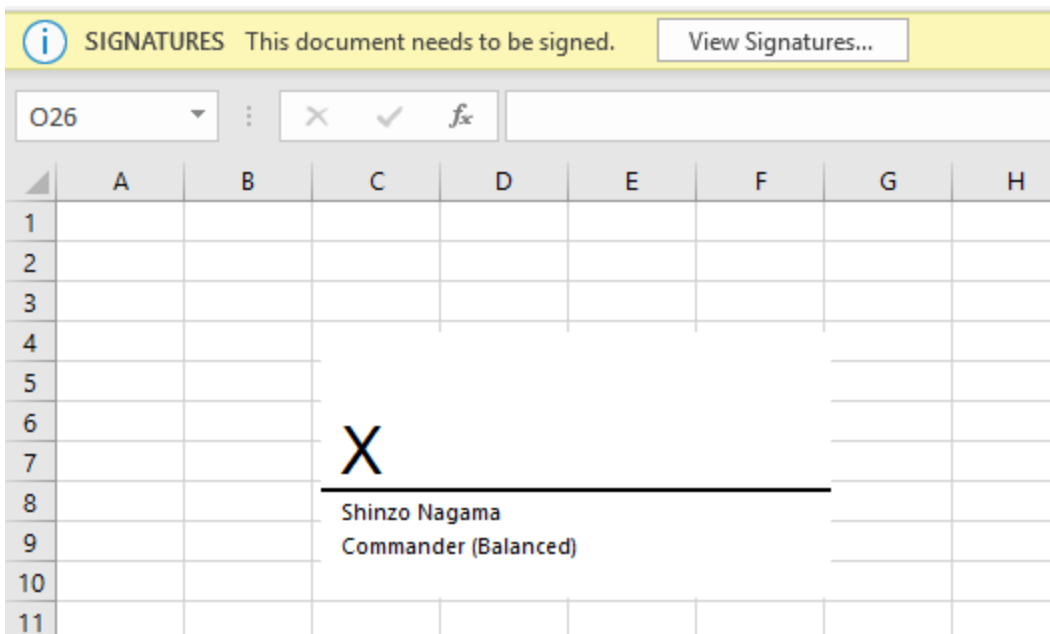
```
Workbook workbook = new Workbook();
ISignature signature =
```

```
workbook.getSignatures().addSignatureLine(workbook.getActiveSheet(), 100.0, 50.0);

// Add Signature lines
ISignatureSetup setup = signature.getSetup();
setup.setShowSignDate(false);
setup.setAllowComments(false);
setup.setSigningInstructions("Please check the content before signing.");
setup.setSuggestedSigner("Shinzo Nagama");
setup.setSuggestedSignerEmail("shinzo.nagama@ea.com");
setup.setSuggestedSignerLine2("Commander (Balanced)");

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

The below image shows the signature lines in Excel:



### Copy Signature Lines

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

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

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

Java

```
// Copy signature line with Range.Copy
IRange srcRange = activeSheet.getRange("A1:I15");
```



```
IRange destRange = activeSheet.getRange("A16:I30");
srcRange.copy(destRange);

// Copy signature line with Shape.Duplicate
signature.getSignatureLineShape().duplicate();

// Copy signature line with Worksheet.Copy
activeSheet.copy();
```

## Delete Signature Lines

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

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

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

### Java

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

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

## Move Signature Lines

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

### Java

```
// Move signature line
IShape signatureLineShape = signatureShinzo.getSignatureLineShape();
signatureLineShape.setTop(signatureLineShape.getTop() + 100);
signatureLineShape.setLeft(signatureLineShape.getLeft() + 50);
```

## List Signature Lines

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

### Java

```
// Add first signature line
ISignature signatureShinzo = signatures.addSignatureLine(activeSheet, 100.0, 50.0);
ISignatureSetup setup1 = signatureShinzo.getSetup();
setup1.setShowSignDate(false);
setup1.setAllowComments(false);
```

```

setup1.setSigningInstructions("Please check the content before signing.");
setup1.setSuggestedSigner("Shinzo Nagama");
setup1.setSuggestedSignerEmail("shinzo.nagama@ea.com");
setup1.setSuggestedSignerLine2("Commander (Balanced)");

ISignature signatureKenji = signatures.addSignatureLine(activeSheet, 100.0, 350.0);
ISignatureSetup setup2 = signatureKenji.getSetup();
setup2.setShowSignDate(true);
setup2.setAllowComments(true);
setup2.setSigningInstructions("Please check the content before signing!");
setup2.setSuggestedSigner("Kenji Tenzai");
setup2.setSuggestedSignerEmail("kenji.tenzai@ea.com");
setup2.setSuggestedSignerLine2("Commander (Mecha)");

// List signatures with indexes
for (int i = 0; i < signatures.getCount(); i++) {
    ISignature signature = signatures.get(i);
    // change SuggestedSigner
    if (i == 0)
        signature.getSetup().setSuggestedSigner("Shinzo Nagama 123");
    // change SuggestedSignerLine2
    if (i == 1)
        signature.getSetup().setSuggestedSignerLine2("Commander (Mecha 1234)");
}

```

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

SignatureLineShape members	Get or Call Behavior	Set Behavior
Adjustments	Supported	#N/A
Adjustments.Count	Supported	#N/A
Adjustments.Item	Not Supported	Not Supported
Adjustments.GetEnumerator	Not Supported	#N/A
AutoShapeType	Supported	Not Supported
BottomRightCell	Supported	#N/A
Chart	Not Supported	#N/A
Connector	Supported	#N/A
ConnectorFormat	Not Supported	#N/A
Fill	Not Supported	#N/A
GroupItems	Not Supported	#N/A
HasChart	Supported	#N/A

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

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

## Add Digital Signatures

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

The **sign** method of **ISignature** interface can be used to add digital signatures. In order to commit signatures, the

workbook should be saved with xlsx or xlsxm extension. A workbook containing digital signatures is 'marked as final' to discourage editing.

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

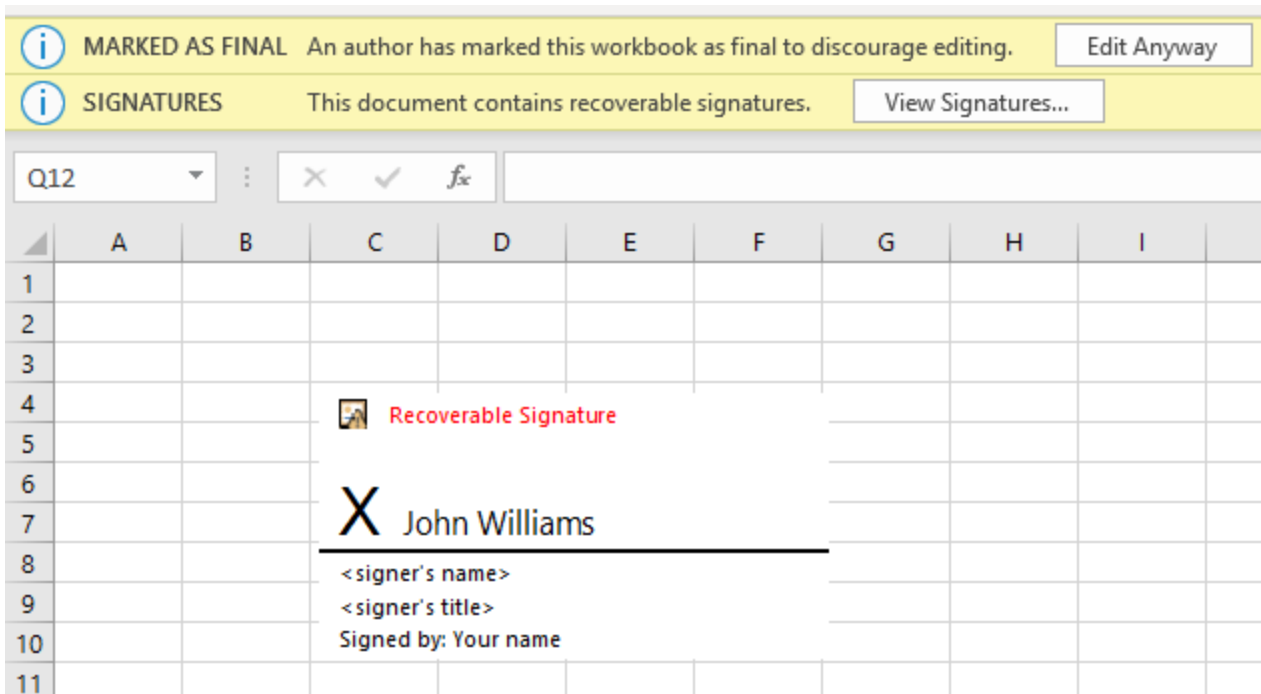
## Java

```
// Create a new workbook
Workbook workbook = new Workbook();
ISignature signature =
workbook.getSignatures().addSignatureLine(workbook.getActiveSheet(), 100.0, 50.0);
ISignatureSetup setup = signature.getSetup();
setup.setShowSignDate(true);
setup.setAllowComments(true);
setup.setSigningInstructions("<your signing instructions>");
setup.setSuggestedSigner("<signer's name>");
setup.setSuggestedSignerEmail("example@microsoft.com");
setup.setSuggestedSignerLine2("<signer's title>");

SignatureDetails details = new SignatureDetails();
details.setAddress1("<your address>");
details.setAddress2("<address 2>");
details.setSignatureComments("Final");
details.setCity("<your city>");
details.setStateOrProvince("<your state or province>");
details.setPostalCode("<your postal code>");
details.setCountryName("<your country or region>");
details.setClaimedRole("<your role>");
details.setCommitmentTypeDescription("Approved");
details.setCommitmentTypeQualifier("Final");

KeyStore ks = KeyStore.getInstance("pkcs12");
String password = "test@123";
char[] passwordChars = password.toCharArray();
String pfxFileKey = "GcExcelTest.pfx";
InputStream pfxStrm = new FileInputStream(pfxFileKey);
ks.load(pfxStrm, passwordChars);
System.out.println(Collections.list(ks.aliases()).size());
signature.sign(ks, password, "John Williams", details);
workbook.save("SignSignatureLines.xlsx");
```

The below image shows digital signature in Excel:



## Add Non Visible Signatures

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

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

Java

```
// Create a new workbook
Workbook workbook = new Workbook();
ISignature signature = workbook.getSignatures().addNonVisibleSignature();
SignatureDetails details = new SignatureDetails();
details.setAddress1("<your address>");
details.setAddress2("<address 2>");
details.setSignatureComments("Final");
details.setCity("<your city>");
details.setStateOrProvince("<your state or province>");
details.setPostalCode("<your postal code>");
details.setCountryName("<your country or region>");
details.setClaimedRole("<your role>");
details.setCommitmentTypeDescription("Approved");
details.setCommitmentTypeQualifier("Final");
KeyStore ks = KeyStore.getInstance("pkcs12");
String password = "test@123";
char[] passwordChars = password.toCharArray();
String pfxFileKey = "GcExcelTest.pfx";
```

```
InputStream pfxStrm = new FileInputStream(pfxFileKey);
ks.load(pfxStrm, passwordChars);
signature.sign(ks, password, details);
// Save to an excel file
workbook.save("AddInvisibleSignatures.xlsx");
```

## Countersign Signatures

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

### Countersign the Workbook

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

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

Java

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

// Modify
workbook.getWorksheets().get(0).getRange("A1").setValue("Modified");

// Countersign
workbook.getSignatures().get(0).countersign(ks, password);

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

### Open the Workbook in Digital Signature Only Mode

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

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

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

Java

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

// Use DigitalSignatureOnly mode, because the workbook was already signed.
```

```
// If you don't open it with digital signature only mode,  
// all existing signatures will be removed after saving the workbook.  
XlsxOpenOptions openOption = new XlsxOpenOptions();  
openOption.setDigitalSignatureOnly(true);  
  
// Add signature to this workbook  
ISignature signature = workbook.getSignatures().addNonVisibleSignature();  
signature.sign(ks, password, details);  
  
// Commit signatures  
workbook.save("AddNonVisibleSignatureToSignedWorkbook.xlsx");
```

## Verify Digital Signatures

GcExcel allows you to verify digital signatures by using **getIsValid** method of **ISignature** interface.

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

### Java

```
// Create a new workbook  
Workbook workbook = new Workbook();  
workbook.open("signsignaturelines.xlsx");  
ISignatureSet signatures = workbook.getSignatures();  
boolean signed = false;  
boolean valid = false;  
X509Certificate certificate = null;  
  
// Verify the first signature  
for (ISignature signature : signatures) {  
    if (signature.getIsSigned()) {  
        // Save the result in locals. You can print them later.  
        signed = true;  
        certificate = signature.getDetails().getSignatureCertificate();  
        valid = signature.getIsValid();  
        break;  
    }  
}  
  
// Verify the first certificate  
boolean certificateIsValid = true;  
// Check expiration date and start date  
try {  
    certificate.checkValidity();  
} catch (CertificateExpiredException e) {  
    certificateIsValid = false;  
    return;  
} catch (CertificateNotYetValidException e) {  
    certificateIsValid = false;
```

```
    return;  
}
```

## Remove Digital Signatures

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

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

### Java

```
// Create a new workbook  
Workbook workbook = new Workbook();  
  
// This file contains 1 signed signature line and  
// a not signed signature line.  
workbook.open("signsignaturelines.xlsx");  
  
// Use DigitalSignatureOnly mode, because the workbook was already signed.  
// If you don't open it with digital signature only mode,  
// all existing signatures will be removed after saving the workbook.  
XlsxOpenOptions openOption = new XlsxOpenOptions();  
openOption.setDigitalSignatureOnly(true);  
  
// Remove signature of signed signature line.  
for (ISignature signature : workbook.getSignatures()) {  
    if (signature.getIsSignatureLine() && signature.getIsSigned()) {  
        // Remove digital signature.  
        // The signature line will not be removed from the SignatureSet  
        // in digital signature only mode.  
        // Because signature lines are shapes.  
        signature.delete();  
        break;  
    }  
}  
  
//commit signatures  
workbook.save("DeleteDigitalSignature.xlsx");
```

## Dependencies

The complete list of GcExcel Java dependencies to use digital signatures can be downloaded from **here** ('GcExcel\_Java\_Dependencies\_DigitalSignature.docx' in the on-line documentation).

## Version Information

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

### Target Office version



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

This version can be observed by using `SignatureDetails.getApplicationVersion` method.

## Minimum Office version

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

## Certificate Compatibility

The certificate compatibility is tested with OpenJDK 14 and Oracle JDK 8. It requires BouncyCastleProvider (in `bcprov-jdk15on`).

The pfx certificate export has been tested on Windows 10, version 1909.

The jks,jce,bks and ubr certificate generation has been tested on JDK 14 keytool on Ubuntu 18.04 LTS.

File extension	Signature algorithm	Private key protection algorithm	Type name	Provider	Is JDK 8 compatible	Is JDK 13+ compatible
*.pfx, *.p12	RSA	AES-256	PKCS12	Not specified	Error 1	Warning 1
	RSA	Triple-DES	PKCS12	Not specified	Warning 1	Warning 1
	DSA, ECDsa	AES-256, Triple-DES	PKCS12	Not specified	Error 3	Error 3
*.jks	RSA	Custom	JKS or PKCS12	Not specified	Warning 1	Warning 1
	DSA, ECDsa	Custom	JKS	Not specified	Error 3	Error 3
*.jce	RSA	Triple-DES	JCEKS	SunJCE	TRUE	TRUE
	DSA, ECDsa	Triple-DES	JCEKS	SunJCE	Error 3	Error 3
*.bks	RSA	Triple-DES	BKS	BC	TRUE	TRUE
	DSA, ECDsa	Triple-DES	BKS	BC	Error 3	Error 3
*.ubr	RSA	PBE/SHA1/TwoFish	UBER	BC	TRUE	TRUE
	DSA, ECDsa	PBE/SHA1/TwoFish	UBER	BC	Error 3	Error 3
*.pem	RSA, DSA, ECDsa	Not supported	I've tried almost all possible type names	Not specified	Error 2	Error 2

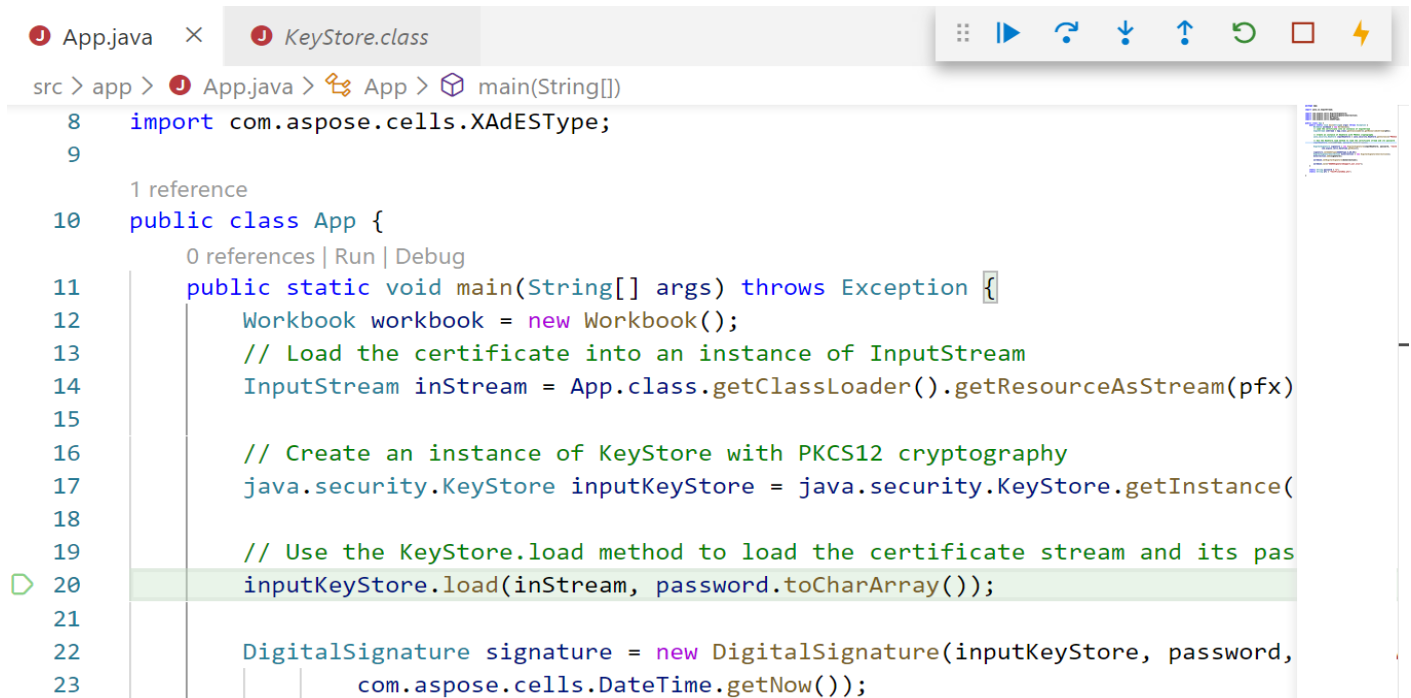
**\*Error 1** - Unable to load certificate if private key is encrypted with AES256-SHA256 mode.

**\*Error 2** - `java.io.IOException: Invalid keystore format`.

**\*Error 3** - If provider was not specified, then throws `java.security.UnrecoverableKeyException: Get Key failed: Given final block not properly padded`. Such issues can arise if a bad key is used during decryption.

**\*Warning 1** - Make sure the code cleanup provides newly registered providers. Otherwise, there might be an error when using this certificate format.

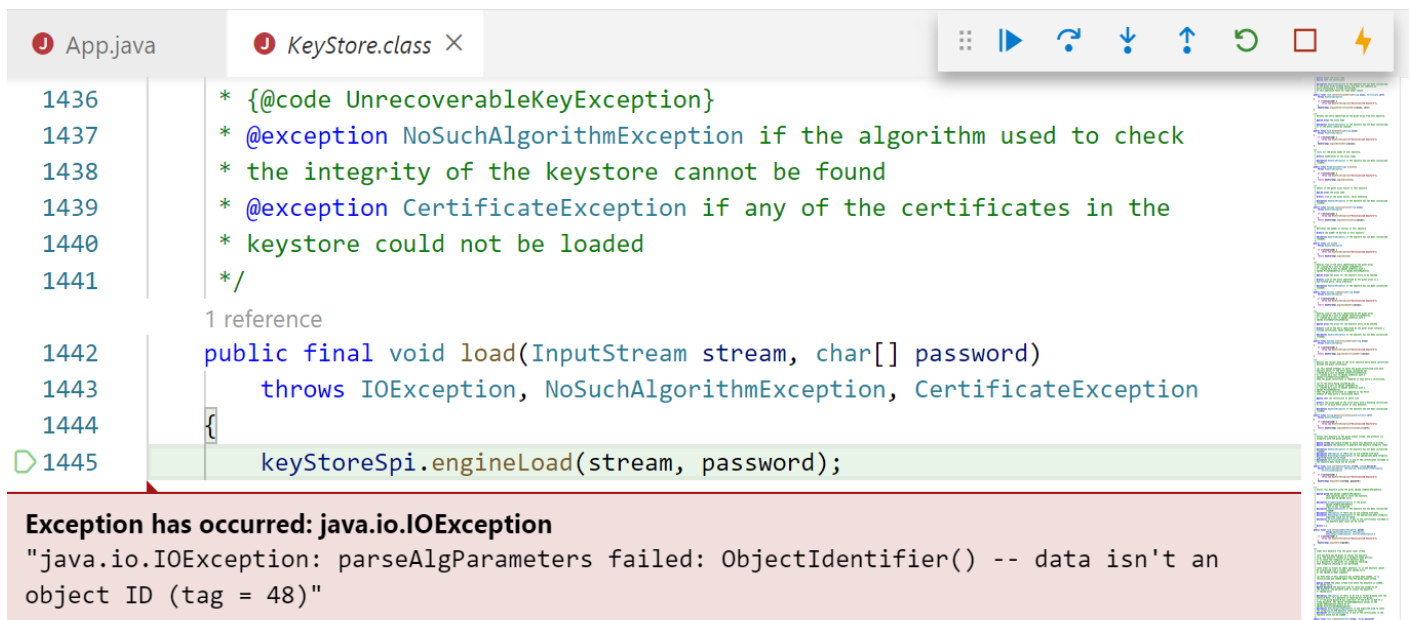
For example, assume that in order to use the bouncy castle provider to open BKS certificates, you have registered BouncyCastleProvider. Then you need to unregister it before opening JKS or pfx certificates. If you are writing unit tests, consider configuring them run synchronously.



```

src > app > App.java > App > main(String[])
8  import com.aspose.cells.XAdESType;
9
10 1 reference
10 public class App {
11     0 references | Run | Debug
11     public static void main(String[] args) throws Exception {
12         Workbook workbook = new Workbook();
13         // Load the certificate into an instance of InputStream
14         InputStream inStream = App.class.getClassLoader().getResourceAsStream(pfx)
15
16         // Create an instance of KeyStore with PKCS12 cryptography
17         java.security.KeyStore inputKeyStore = java.security.KeyStore.getInstance(
18
19         // Use the KeyStore.load method to load the certificate stream and its pas
20         inputKeyStore.load(inStream, password.toCharArray());
21
22         DigitalSignature signature = new DigitalSignature(inputKeyStore, password,
23         com.aspose.cells.DateTime.getNow());

```



```

App.java  KeyStore.class X
1436 * {@code UnrecoverableKeyException}
1437 * @exception NoSuchAlgorithmException if the algorithm used to check
1438 * the integrity of the keystore cannot be found
1439 * @exception CertificateException if any of the certificates in the
1440 * keystore could not be loaded
1441 */
1442 1 reference
1442 public final void load(InputStream stream, char[] password)
1443     throws IOException, NoSuchAlgorithmException, CertificateException
1444 {
1445 keyStoreSpi.engineLoad(stream, password);

```

**Exception has occurred: java.io.IOException**  
"java.io.IOException: parseAlgParameters failed: ObjectIdentifier() -- data isn't an object ID (tag = 48)"

### Possible solutions

- Upgrade to the latest JDK (Recommended).
- Convert certificate format to supported formats.
- Use weaker encryption algorithms that the platform supports. For example, downgrade AES256-SHA256 to TripleDES-SHA1 with OpenSSL.

- Use 3rd-party certificate providers (if you trust them).
- Develop a new certificate provider by yourself (advanced).

## Limitations

- Only Microsoft Office signature lines are supported.
- Emf image files are not supported. Hence, when exporting signature lines, the signature image is skipped if it is in emf format. The preview images are also emf images. Hence, placeholder preview images are exported instead.
- The date format of signature line does not follow system configurations but Excel follows it.
- The X.509 certificate being used must have a password.
- Java 8 or higher is required to use digital signatures. Otherwise, an exception will be thrown at runtime while opening a signed workbook or signing a workbook.
- If you are using PKCS#12 files (\*.pfx) to store private key with AES-256 encryption, your app or service must run on OpenJDK or Oracle JDK 11.0.3, 12.0.2 or 13+ . Refer these bugs ([JDK-8214513](#) and [JDK-8220734](#)) for details. Caution: Triple-DES is not safe enough for protecting your private key. Refer [this](#).
- When running on JDK 9 or higher, a warning occurs "An illegal reflective access operation has occurred".
- Certificate validation returns incorrect result if the certificate chain contains 2 or more items. This is because KeyStore.getCertificateChain(String) method doesn't get certificate chain from Windows certificate storage or OpenSSL certificate storage.
- The use of Apache POI may change class load order or break component version constraints.
- SLF4J prints warning "SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder"". This warning cannot be removed, because users might use SLF4J to write logs.
- While signing or verifying a workbook, you can only use RSA. This is caused by limitations of default Java key store and org.bouncycastle.jcajce.provider.asymmetric.rsa.DigestSignatureSpi.

Key Algorithm	Action	Supported
RSA	Sign/Verify	Yes
DSA	Sign/Verify	No
ECDsa	Sign/Verify	No

## Formulas

GcExcel Java allows users to create and use formulas in order to facilitate financial analysis and improve data processing while saving both time and efforts.

Basically, a formula refers to an expression that helps in calculating the value of a cell quickly and accurately when applied in a worksheet. While applying formulas, you can also use some built-in functions and operators to generate formulas and calculate values in the cells.

In order to carry out complex arithmetic calculations, GcExcel provides support for adding and using formulas in a workbook. Formula computation always begins from left and extends towards the right based on 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 formulas in GcExcel Java. The first one holds the maximum precedence and last one holds the minimum precedence.

1. Parentheses evaluation of expressions
2. Range evaluation
3. Evaluation of spaces within the expression

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

In GcExcel Java, you can manage formulas in the following ways:

- [Formula Functions](#)
- [Set Formula to Range](#)
- [Set Table Formula](#)
- [Set Array Formula](#)
- [Precedents and Dependents](#)

## Formula Functions

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

Function Name	Function Category	Function Description
ABS	Math and Trigonometry	Returns the absolute value of a number.
ACCRINT	Financial	Returns the accrued interest for a security that pays periodic interest.
ACCRINTM	Financial	Returns the accrued interest for a security that pays interest at maturity.
ACOS	Math and Trigonometry	Returns the arccosine of a number.
ACOSH	Math and Trigonometry	Returns the inverse hyperbolic cosine of a number.
ACOT	Math and Trigonometry	Returns the arccotangent of a number.
ACOTH	Math and Trigonometry	Returns the hyperbolic arccotangent of a number.
ADDRESS	Lookup and Reference	Returns a reference as text to a single cell in a worksheet.
AMORDEGRC	Financial	Returns the depreciation for each accounting period by using a depreciation coefficient.
AMORLINC	Financial	Returns the depreciation for each accounting period.
AND	Logical	Returns TRUE if all of its arguments are TRUE.
ARABIC	Math and	Converts a Roman number to Arabic, as a number.

	Trigonometry	
AREAS	Lookup and Reference	Returns the number of areas in a reference.
ASIN	Math and Trigonometry	Returns the arcsine of a number.
ASINH	Math and Trigonometry	Returns the inverse hyperbolic sine of a number.
ATAN	Math and Trigonometry	Returns the arctangent of a number.
ATAN2	Math and Trigonometry	Returns the arctangent from x- and y-coordinates.
ATANH	Math and Trigonometry	Returns the inverse hyperbolic tangent of a number.
AVEDEV	Statistical	Returns the average of the absolute deviations of data points from their mean.
AVERAGE	Statistical	Returns the average of its arguments.
AVERAGEA	Statistical	Returns the average of its arguments, including numbers, text, and logical values.
AVERAGEIF	Statistical	Returns the average (arithmetic mean) of all the cells in a range that meet a given criteria.
AVERAGEIFS	Statistical	Returns the average (arithmetic mean) of all cells that meet multiple criteria.
BAHTTEXT	Text	Converts a number to text, using the ₮ (baht) currency format.
BASE	Math and Trigonometry	Converts a number into a text representation with the given radix (base).
BESSELI	Engineering	Returns the modified Bessel function $I_n(x)$ .
BESSELJ	Engineering	Returns the Bessel function $J_n(x)$ .
BESSELK	Engineering	Returns the modified Bessel function $K_n(x)$ .
BESSELY	Engineering	Returns the Bessel function $Y_n(x)$ .
BETA.DIST	Statistical	Returns the beta cumulative distribution function.
BETA.INV	Statistical	Returns the inverse of the cumulative distribution function for a specified beta distribution.
BETADIST	Compatibility	Returns the beta cumulative distribution function.
BETAINV	Compatibility	Returns the inverse of the cumulative distribution function for a specified beta distribution.
BIN2DEC	Engineering	Converts a binary number to decimal.

BIN2HEX	Engineering	Converts a binary number to hexadecimal.
BIN2OCT	Engineering	Converts a binary number to octal.
BINOM.DIST	Statistical	Returns the individual term binomial distribution probability.
BINOM.DIST.RANGE	Statistical	Returns the probability of a trial result using a binomial distribution.
BINOM.INV	Statistical	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value.
BINOMDIST	Compatibility	Returns the individual term binomial distribution probability.
BITAND	Engineering	Returns a 'Bitwise And' of two numbers.
BITLSHIFT	Engineering	Returns a value number shifted left by shift_amount bits.
BITOR	Engineering	Returns a bitwise OR of 2 numbers.
BITRSHIFT	Engineering	Returns a value number shifted right by shift_amount bits.
BITXOR	Engineering	Returns a bitwise 'Exclusive Or' of two numbers.
CEILING	Math and Trigonometry	Rounds a number to the nearest integer or to the nearest multiple of significance.
CEILING.MATH	Math and Trigonometry	Rounds a number up, to the nearest integer or to the nearest multiple of significance.
CELL	Information	Returns information about the formatting, location, or contents of a cell.
CHAR	Text	Returns the character specified by the code number.
CHIDIST	Compatibility	Returns the one-tailed probability of the chi-squared distribution.
CHIINV	Compatibility	Returns the inverse of the one-tailed probability of the chi-squared distribution.
CHISQ.DIST	Statistical	Returns the cumulative beta probability density function.
CHISQ.DIST.RT	Statistical	Returns the one-tailed probability of the chi-squared distribution.
CHISQ.INV	Statistical	Returns the cumulative beta probability density function.
CHISQ.INV.RT	Statistical	Returns the inverse of the one-tailed probability of the chi-squared distribution.
CHISQ.TEST	Statistical	Returns the test for independence.
CHITEST	Compatibility	Returns the test for independence.
CHOOSE	Lookup and reference	Chooses a value from a list of values.
CLEAN	Text	Removes all nonprintable characters from text.
CODE	Text	Returns a numeric code for the first character in a text string.
COLUMN	Lookup and reference	Returns the column number of a reference.
COLUMNS	Lookup and reference	Returns the number of columns in a reference.

COMBIN	Math and trigonometry	Returns the number of combinations for a given number of objects.
COMBINA	Math and trigonometry	Returns the number of combinations for a specified number of items including the repetitions.
COMPLEX	Engineering	Converts real and imaginary coefficients into a complex number.
CONCAT	Text	Combines the text from multiple ranges and/or strings, but it doesn't provide the delimiter or IgnoreEmpty arguments.
CONCATENATE	Text	Joins several text items into one text item.
CONFIDENCE	Compatibility	Returns the confidence interval for a population mean.
CONFIDENCE.NORM	Statistical	Returns the confidence interval for a population mean.
CONFIDENCE.T	Statistical	Returns the confidence interval for a population mean, using a Student's t distribution.
CONVERT	Engineering	Converts a number from one measurement system to another.
CORREL	Statistical	Returns the correlation coefficient between two data sets.
COS	Math and trigonometry	Returns the cosine of a number.
COSH	Math and trigonometry	Returns the hyperbolic cosine of a number.
COT	Math and trigonometry	Returns the cotangent of an angle.
COTH	Math and trigonometry	Returns the hyperbolic cotangent of an angle.
COUNT	Statistical	Counts how many numbers are in the list of arguments.
COUNTA	Statistical	Counts how many values are in the list of arguments.
COUNTBLANK	Statistical	Counts the number of blank cells within a range.
COUNTIF	Statistical	Counts the number of cells within a range that meet the given criteria.
COUNTIFS	Statistical	Counts the number of cells within a range that meet multiple criteria.
COUPDAYBS	Financial	Returns the number of days from the beginning of the coupon period to the settlement date.
COUPDAYS	Financial	Returns the number of days in the coupon period that contains the settlement date.
COUPDAYSNC	Financial	Returns the number of days in the coupon period that contains the settlement date.
COUPNCD	Financial	Returns the next coupon date after the settlement date.
COUPNUM	Financial	Returns the number of coupons payable between the settlement date and maturity date.

COUPPCD	Financial	Returns the previous coupon date before the settlement date.
COVAR	Compatibility	Returns covariance, the average of the products of paired deviations.
COVARIANCE.P	Statistical	Returns covariance, the average of the products of paired deviations.
COVARIANCE.S	Statistical	Returns the sample covariance, the average of the products deviations for each data point pair in two data sets.
CRITBINOM	Compatibility	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value.
CSC	Math and trigonometry	Returns the cosecant of an angle.
CSCH	Math and trigonometry	Returns the hyperbolic cosecant of an angle.
CUMIPMT	Financial	Returns the cumulative interest paid between two periods.
CUMPRINC	Financial	Returns the cumulative principal paid on a loan between two periods.
DATE	Date and time	Returns the serial number of a particular date.
DATEDIF	Date and time	Calculates the number of days, months, or years between two dates. This function is useful in formulas where you need to calculate an age.
DATEVALUE	Date and time	Converts a date in the form of text to a serial number.
DAVERAGE	Database	Returns the average of selected database entries.
DAY	Date and time	Converts a serial number to a day of the month.
DAYS	Date and time	Returns the number of days between two dates.
DAYS360	Date and time	Calculates the number of days between two dates based on a 360-day year.
DB	Financial	Returns the depreciation of an asset for a specified period by using the fixed-declining balance method.
DCOUNT	Database	Changes half-width (single-byte) English letters or katakana within a character string to full-width (double-byte) characters.
DCOUNTA	Database	Counts nonblank cells in a database.
DDB	Financial	Returns the depreciation of an asset for a specified period by using the double-declining balance method or some other method that you specify.
DEC2BIN	Engineering	Converts a decimal number to binary.
DEC2HEX	Engineering	Converts a decimal number to hexadecimal.
DEC2OCT	Engineering	Converts a decimal number to octal.
DECIMAL	Math and trigonometry	Converts a text representation of a number in a given base into a decimal number.
DEGREES	Math and	Converts radians to degrees.



	trigonometry	
DELTA	Engineering	Tests whether two values are equal.
DEVSQ	Statistical	Returns the sum of squares of deviations.
DGET	Database	Extracts from a database a single record that matches the specified criteria.
DISC	Financial	Returns the discount rate for a security.
DMAX	Database	Returns the maximum value from selected database entries.
DMIN	Database	Returns the minimum value from selected database entries.
DOLLAR	Text	Converts a number to text, using the \$ (dollar) currency format.
DOLLARDE	Financial	Converts a dollar price, expressed as a fraction, into a dollar price, expressed as a decimal number.
DOLLARFR	Financial	Converts a dollar price, expressed as a decimal number, into a dollar price, expressed as a fraction.
DPRODUCT	Database	Multiplies the values in a particular field of records that match the criteria in a database.
DSTDEV	Database	Estimates the standard deviation based on a sample of selected database entries.
DSTDEVP	Database	Calculates the standard deviation based on the entire population of selected database entries.
DSUM	Database	Adds the numbers in the field column of records in the database that match the criteria.
DURATION	Financial	Returns the annual duration of a security with periodic interest payments.
DVAR	Database	Estimates variance based on a sample from selected database entries.
DVARP	Database	Calculates variance based on the entire population of selected database entries.
EDATE	Date and time	Returns the serial number of the date that is the indicated number of months before or after the start date.
EFFECT	Financial	Returns the effective annual interest rate.
ENCODEURL	Web	Returns a URL-encoded string.
EOMONTH	Date and time	Returns the serial number of the last day of the month before or after a specified number of months.
ERF	Engineering	Returns the error function.
ERF.PRECISE	Engineering	Returns the error function.
ERFC	Engineering	Returns the complementary error function.
ERFC.PRECISE	Engineering	Returns the complementary ERF function integrated between x and

		infinity.
ERROR.TYPE	Information	Returns a number corresponding to an error type.
EUROCONVERT	Add-in and Automation	Converts a number to euros, converts a number from euros to a euro member currency, or converts a number from one euro member currency to another by using the euro as an intermediary (triangulation).
EVEN	Math and Trigonometry	Rounds a number up to the nearest even integer.
EXACT	Text	Checks to see if two text values are identical.
EXP	Math and Trigonometry	Returns e raised to the power of a given number.
EXPON.DIST	Statistical	Returns the exponential distribution.
EXPONDIST	Compatibility	Returns the exponential distribution.
F.DIST	Statistical	Returns the F probability distribution
F.DIST.RT	Statistical	Returns the F probability distribution
F.INV	Statistical	Returns the inverse of the F probability distribution.
F.INV.RT	Statistical	Returns the inverse of the F probability distribution.
F.TEST	Statistical	Returns the result of an F-test.
FACT	Math and trigonometry	Returns the factorial of a number.
FACTDOUBLE	Math and trigonometry	Returns the double factorial of a number.
FALSE	Logical	Returns the logical value FALSE.
FDIST	Compatibility	Returns the F probability distribution.
FIND	Text	Finds one text value within another (case-sensitive).
FINDBs	Text	Finds one text value within another (case-sensitive).
FINV	Statistical	Returns the inverse of the F probability distribution.
FISHER	Statistical	Returns the Fisher transformation.
FISHERINV	Statistical	Returns the inverse of the Fisher transformation.
FIXED	Text	Formats a number as text with a fixed number of decimals.
FLOOR	Compatibility	Rounds a number down, toward zero.
FLOOR.MATH	Math and trigonometry	Rounds a number down, to the nearest integer or to the nearest multiple of significance.
FLOOR.PRECISE	Math and trigonometry	Rounds a number the nearest integer or to the nearest multiple of significance. Regardless of the sign of the number, the number is

		rounded up.
FORECAST	Statistical	Returns a value along a linear trend.
FORMULATEXT	Lookup and reference	Returns the formula at the given reference as text.
FREQUENCY	Statistical	Returns a frequency distribution as a vertical array.
FTEST	Compatibility	Returns the result of an F-test.
FV	Financial	Returns the future value of an investment.
FVSCHEDULE	Financial	Returns the future value of an initial principal after applying a series of compound interest rates.
GAMMA	Statistical	Returns the Gamma function value.
GAMMA.DIST	Statistical	Returns the Gamma distribution.
GAMMA.INV	Statistical	Returns the inverse of the gamma cumulative distribution.
GAMMADIST	Compatibility	Returns the gamma distribution.
GAMMAINV	Compatibility	Returns the inverse of the gamma cumulative distribution.
GAMMALN	Statistical	Returns the natural logarithm of the gamma function, $\Gamma(x)$ .
GAMMALN.PRECISE	Statistical	Returns the natural logarithm of the gamma function, $\Gamma(x)$ .
GAUSS	Statistical	Returns 0.5 less than the standard normal cumulative distribution.
GCD	Math and trigonometry	Returns the greatest common divisor.
GEOMEAN	Statistical	Returns the geometric mean.
GESTEP	Engineering	Tests whether a number is greater than a threshold value.
GROWTH	Statistical	Returns values along an exponential trend.
HARMEAN	Statistical	Returns the harmonic mean.
HEX2BIN	Engineering	Converts a hexadecimal number to binary.
HEX2DEC	Engineering	Converts a hexadecimal number to decimal.
HEX2OCT	Engineering	Converts a hexadecimal number to octal.
HLOOKUP	Lookup and reference	Looks in the top row of an array and returns the value of the indicated cell.
HOUR	Date and time	Converts a serial number to an hour.
HYPERLINK	Lookup and reference	Creates a shortcut or jump that opens a document stored on a network server, an intranet, or the Internet.
HYPGEOM.DIST	Statistical	Returns the hypergeometric distribution.
HYPGEOMDIST	Compatibility	Returns the hypergeometric distribution.
IF	Logical	Specifies a logical test to perform

IFERROR	Logical	Returns a value you specify if a formula evaluates to an error; otherwise, returns the result of the formula.
IFNA	Logical	Returns the value you specify if the expression resolves to #N/A, otherwise returns the result of the expression.
IFS	Logical	Checks whether one or more conditions are met and returns a value that corresponds to the first TRUE condition..
IMABS	Engineering	Returns the absolute value (modulus) of a complex number.
IMAGINARY	Engineering	Returns the imaginary coefficient of a complex number.
IMARGUMENT	Engineering	Returns the argument theta, an angle expressed in radians.
IMCONJUGATE	Engineering	Returns the complex conjugate of a complex number.
IMCOS	Engineering	Returns the cosine of a complex number.
IMCOSH	Engineering	Returns the hyperbolic cosine of a complex number.
IMCOT	Engineering	Returns the cotangent of a complex number.
IMCSC	Engineering	Returns the cosecant of a complex number.
IMCSCH	Engineering	Returns the hyperbolic cosecant of a complex number.
IMDIV	Engineering	Returns the quotient of two complex numbers.
IMEXP	Engineering	Returns the exponential of a complex number.
IMLN	Engineering	Returns the natural logarithm of a complex number.
IMLOG10	Engineering	Returns the base-10 logarithm of a complex number.
IMLOG2	Engineering	Returns the base-2 logarithm of a complex number.
IMPOWER	Engineering	Returns a complex number raised to an integer power.
IMPRODUCT	Engineering	Returns the product of complex numbers.
IMREAL	Engineering	Returns the real coefficient of a complex number.
IMSEC	Engineering	Returns the secant of a complex number.
IMSECH	Engineering	Returns the hyperbolic secant of a complex number.
IMSIN	Engineering	Returns the sine of a complex number.
IMSINH	Engineering	Returns the hyperbolic sine of a complex number.
IMSQRT	Engineering	Returns the square root of a complex number.
IMSUB	Engineering	Returns the difference between two complex numbers.
IMSUM	Engineering	Returns the sum of complex numbers.
IMTAN	Engineering	Returns the tangent of a complex number.
INDEX	Lookup and reference	Uses an index to choose a value from a reference or array.
INDIRECT	Lookup and reference	Returns a reference indicated by a text value.

INT	Math and trigonometry	Rounds a number down to the nearest integer.
INTERCEPT	Statistical	Returns the intercept of the linear regression line.
INTRATE	Financial	Returns the interest rate for a fully invested security.
IPMT	Financial	Returns the interest payment for an investment for a given period.
IRR	Financial	Returns the internal rate of return for a series of cash flowsReturns the internal rate of return for a series of cash flows.
ISBLANK	Information	Returns TRUE if the value is blank.
ISERR	Information	Returns TRUE if the value is any error value except #N/A.
ISERROR	Information	Returns TRUE if the value is any error value.
ISEVEN	Information	Returns TRUE if the number is even.
ISFORMULA	Information	Returns TRUE if there is a reference to a cell that contains a formula.
ISLOGICAL	Information	Returns TRUE if the value is a logical value.
ISNA	Information	Returns TRUE if the value is the #N/A error value.
ISNONTEXT	Information	Returns TRUE if the value is not text.
ISNUMBER	Information	Returns TRUE if the value is a number.
ISO.CEILING	Math and trigonometry	Returns a number that is rounded up to the nearest integer or to the nearest multiple of significance.
ISODD	Information	Returns TRUE if the number is odd.
ISOWEEKNUM	Date and time	Returns the number of the ISO week number of the year for a given date.
ISPMT	Financial	Calculates the interest paid during a specific period of an investment.
ISREF	Information	Returns TRUE if the value is a reference.
ISTEXT	Information	Returns TRUE if the value is text.
KURT	Statistical	Returns TRUE if the value is text.
LARGE	Statistical	Returns the k-th largest value in a data set.
LCM	Math and trigonometry	Returns the least common multiple.
LEFT	Text	Returns the leftmost characters from a text value.
LEFTBs	Text	Returns the leftmost characters from a text value.
LEN	Text	Returns the number of characters in a text string.
LENBs	Text	Returns the number of characters in a text string.
LINEST	Statistical	Returns the parameters of a linear trend.
LN	Math and	Returns the natural logarithm of a number.

	trigonometry	
LOG	Math and trigonometry	Returns the logarithm of a number to a specified base.
LOG10	Math and trigonometry	Returns the base-10 logarithm of a number.
LOGEST	Statistical	Returns the parameters of an exponential trend.
LOGINV	Compatibility	Returns the inverse of the lognormal cumulative distribution.
LOGNORM.DIST	Statistical	Returns the cumulative lognormal distribution.
LOGNORM.INV	Statistical	Returns the inverse of the lognormal cumulative distribution.
LOGNORMDIST	Compatibility	Returns the cumulative lognormal distribution.
LOOKUP	Lookup and reference	Looks up values in a vector or array.
LOWER	Text	Converts text to lowercase.
MATCH	Lookup and reference	Looks up values in a reference or array.
MAX	Statistical	Returns the maximum value in a list of arguments.
MAXA	Statistical	Returns the maximum value in a list of arguments, including numbers, text, and logical values.
MAXIFS	Statistical	Returns the maximum value among cells specified by a given set of conditions or criteria.
MDETERM	Math and trigonometry	Returns the matrix determinant of an array.
MDURATION	Financial	Returns the Macauley modified duration for a security with an assumed par value of \$100.
MEDIAN	Statistical	Returns the median of the given numbers.
MID	Text	Returns a specific number of characters from a text string starting at the position you specify.
MIDBs	Text	Returns a specific number of characters from a text string starting at the position you specify.
MIN	Statistical	Returns the minimum value in a list of arguments.
MINA	Statistical	Returns the smallest value in a list of arguments, including numbers, text, and logical values.
MINIFS	Statistical	Returns the minimum value among cells specified by a given set of conditions or criteria.
MINUTE	Date and time	Converts a serial number to a minute.
MINVERSE	Math and trigonometry	Returns the matrix inverse of an array.
MIRR	Financial	Returns the internal rate of return where positive and negative cash

		flows are financed at different rates.
MMULT	Math and trigonometry	Returns the matrix product of two arrays.
MOD	Math and trigonometry	Returns the remainder from division.
MODE	Compatibility	Returns the most common value in a data set.
MODE.MULT	Statistical	Returns a vertical array of the most frequently occurring, or repetitive values in an array or range of data.
MODE.SNGL	Statistical	Returns the most common value in a data set.
MONTH	Date and time	Converts a serial number to a month.
MROUND	Math and trigonometry	Returns a number rounded to the desired multiple.
MULTINOMIAL	Math and trigonometry	Returns the multinomial of a set of numbers.
MUNIT	Math and trigonometry	Returns the unit matrix or the specified dimension.
N	Information	Returns a value converted to a number.
NA	Information	Returns the error value #N/A.
NEGBINOM.DIST	Statistical	Returns the negative binomial distribution.
NEGBINOMDIST	Compatibility	Returns the negative binomial distribution.
NETWORKDAYS	Date and time	Returns the number of whole workdays between two dates.
NETWORKDAYS.INTL	Date and time	Returns the number of whole workdays between two dates using parameters to indicate which and how many days are weekend days.
NOMINAL	Financial	Returns the annual nominal interest rate.
NORM.DIST	Statistical	Returns the normal cumulative distribution.
NORM.INV	Compatibility	Returns the inverse of the normal cumulative distribution.
NORM.S.DIST	Statistical	Returns the standard normal cumulative distribution.
NORM.S.INV	Statistical	Returns the inverse of the standard normal cumulative distribution.
NORMDIST	Compatibility	Returns the normal cumulative distribution.
NORMINV	Statistical	Returns the inverse of the normal cumulative distribution.
NORMSDIST	Compatibility	Returns the standard normal cumulative distribution.
NORMSINV	Compatibility	Returns the inverse of the standard normal cumulative distribution.
NOT	Logical	Reverses the logic of its argument.
NOW	Date and time	Returns the serial number of the current date and time.

NPER	Financial	Returns the number of periods for an investment.
NPV	Financial	Returns the net present value of an investment based on a series of periodic cash flows and a discount rate.
NUMBERVALUE	Text	Converts text to number in a locale-independent manner.
OCT2BIN	Engineering	Converts an octal number to binary.
OCT2DEC	Engineering	Converts an octal number to decimal.
OCT2HEX	Engineering	Converts an octal number to hexadecimal.
ODD	Math and trigonometry	Rounds a number up to the nearest odd integer.
ODDFPRICE	Financial	Returns the price per \$100 face value of a security with an odd first period.
ODDFYIELD	Financial	Returns the yield of a security with an odd first period.
ODDLPRICE	Financial	Returns the price per \$100 face value of a security with an odd last period.
ODDLYIELD	Financial	Returns the yield of a security with an odd last period.
OFFSET	Lookup and reference	Returns a reference offset from a given reference.
OR	Logical	Returns TRUE if any argument is TRUE.
PDURATION	Financial	Returns the number of periods required by an investment to reach a specified value.
PEARSON	Statistical	Returns the Pearson product moment correlation coefficient.
PERCENTILE	Compatibility	Returns the k-th percentile of values in a range.
PERCENTILE.EXC	Statistical	Returns the k-th percentile of values in a range, where k is in the range 0..1, exclusive.
PERCENTILE.INC	Statistical	Returns the k-th percentile of values in a range.
PERCENTRANK	Compatibility	Returns the percentage rank of a value in a data set.
PERCENTRANK.EXC	Statistical	Returns the rank of a value in a data set as a percentage (0..1, exclusive) of the data set.
PERCENTRANK.INC	Statistical	Returns the percentage rank of a value in a data set.
PERMUT	Statistical	Returns the number of permutations for a given number of objects.
PERMUTATIONA	Statistical	Returns the number of permutations for a given number of objects (with repetitions) that can be selected from the total objects.
PHI	Statistical	Returns the value of the density function for a standard normal distribution.
PI	Math and trigonometry	Returns the value of pi.



PMT	Financial	Returns the periodic payment for an annuity.
POISSON	Compatibility	Returns the Poisson distribution.
POISSON.DIST	Statistical	Returns the Poisson distribution.
POWER	Math and trigonometry	Returns the result of a number raised to a power.
PPMT	Financial	Returns the payment on the principal for an investment for a given period.
PRICE	Financial	Returns the price per \$100 face value of a security that pays periodic interest.
PRICEDISC	Financial	Returns the price per \$100 face value of a discounted security.
PRICEMAT	Financial	Returns the price per \$100 face value of a security that pays interest at maturity.
PROB	Statistical	Returns the probability that values in a range are between two limits.
PRODUCT	Math and trigonometry	Multiplies its arguments.
PROPER	Text	Capitalizes the first letter in each word of a text value.
PV	Financial	Returns the present value of an investment.
QUARTILE	Compatibility	Returns the quartile of a data set.
QUARTILE.EXC	Statistical	Returns the quartile of the data set, based on percentile values from 0..1, exclusive.
QUARTILE.INC	Statistical	Returns the quartile of a data set.
QUOTIENT	Math and trigonometry	Returns the integer portion of a division.
RADIANS	Math and trigonometry	Converts degrees to radians.
RAND	Math and trigonometry	Returns a random number between 0 and 1.
RANDBETWEEN	Math and trigonometry	Returns a random number between the numbers you specify.
RANK	Compatibility	Returns the rank of a number in a list of numbers.
RANK.AVG	Statistical	Returns the rank of a number in a list of numbers.
RANK.EQ	Statistical	Returns the rank of a number in a list of numbers.
RATE	Financial	Returns the interest rate per period of an annuity.
RECEIVED	Financial	Returns the amount received at maturity for a fully invested security.
REPLACE	Text	Replaces characters within text.

REPLACEBs	Text	Replaces characters within text
REPT	Text	Repeats text a given number of times.
RIGHT	Text	Returns the rightmost characters from a text value.
RIGHTBs	Text	Returns the rightmost characters from a text value.
ROMAN	Math and trigonometry	Converts an arabic numeral to roman, as text.
ROUND	Math and trigonometry	Rounds a number to a specified number of digits.
ROUNDSDOWN	Math and trigonometry	Rounds a number down, toward zero.
ROUNDUP	Math and trigonometry	Rounds a number up, away from zero.
ROW	Lookup and reference	Returns the row number of a reference.
ROWS	Lookup and reference	Returns the number of rows in a reference.
RRI	Financial	Returns an equivalent interest rate for the growth of an investment.
RSQ	Statistical	Returns the square of the Pearson product moment correlation coefficient.
SEARCH	Text	Finds one text value within another (not case-sensitive).
SEARCHBs	Text	Finds one text value within another (not case-sensitive).
SEC	Math and trigonometry	Returns the secant of an angle.
SECH	Math and trigonometry	Returns the hyperbolic secant of an angle.
SECOND	Date and Time	Converts a serial number to a second.
SERIESSUM	Math and trigonometry	Returns the sum of a power series based on the formula.
SHEET	Information	Returns the sheet number of the referenced sheet.
SHEETS	Information	Returns the number of sheets in a reference.
SIGN	Math and trigonometry	Returns the sign of a number.
SIN	Math and trigonometry	Returns the sine of the given angle.
SINH	Math and trigonometry	Returns the hyperbolic sine of a number.
SKEW	Statistical	Returns the skewness of a distribution.
SKEW.P	Statistical	Returns the skewness of a distribution based on a population: a

		characterization of the degree of asymmetry of a distribution around its mean.
SLN	Financial	Returns the straight-line depreciation of an asset for one period.
SLOPE	Statistical	Returns the slope of the linear regression line.
SMALL	Statistical	Returns the k-th smallest value in a data set.
SQRT	Math and trigonometry	Returns a positive square root.
SQRTPI	Math and trigonometry	Returns the square root of (number * pi).
STANDARDIZE	Statistical	Returns a normalized value.
STDEV	Compatibility	Estimates standard deviation based on a sample.
STDEV.P	Statistical	Calculates standard deviation based on the entire population.
STDEV.S	Statistical	Estimates standard deviation based on a sample.
STDEVA	Statistical	Estimates standard deviation based on a sample, including numbers, text, and logical values.
STDEVP	Compatibility	Calculates standard deviation based on the entire population.
STDEVPA	Statistical	Calculates standard deviation based on the entire population, including numbers, text, and logical values.
STEYX	Statistical	Returns the standard error of the predicted y-value for each x in the regression.
SUBSTITUTE	Text	Substitutes new text for old text in a text string.
SUBTOTAL	Math and trigonometry	Returns a subtotal in a list or database.
SUM	Math and trigonometry	Adds its arguments.
SUMIF	Math and trigonometry	Adds the cells specified by a given criteria.
SUMIFS	Math and trigonometry	Adds the cells in a range that meet multiple criteria.
SUMPRODUCT	Math and trigonometry	Returns the sum of the products of corresponding array components.
SUMSQ	Math and trigonometry	Returns the sum of the squares of the arguments.
SUMX2MY2	Math and trigonometry	Returns the sum of the difference of squares of corresponding values in two arrays.
SUMX2PY2	Math and trigonometry	Returns the sum of the sum of squares of corresponding values in two arrays.

SUMXMY2	Math and trigonometry	Returns the sum of squares of differences of corresponding values in two arrays.
SWITCH	Logical	Evaluates an expression against a list of values and returns the result corresponding to the first matching value. If there is no match, an optional default value may be returned.
SYD	Financial	Returns the sum-of-years' digits depreciation of an asset for a specified period.
T	Text	Converts its arguments to text.
T.DIST	Statistical	Returns the Percentage Points (probability) for the Student t-distribution.
T.DIST.2T	Statistical	Returns the Percentage Points (probability) for the Student t-distribution.
T.DIST.RT	Statistical	Returns the Student's t-distribution.
T.INV	Statistical	Returns the t-value of the Student's t-distribution as a function of the probability and the degrees of freedom.
T.INV.2T	Statistical	Returns the inverse of the Student's t-distribution.
T.TEST	Statistical	Returns the probability associated with a Student's t-test.
TAN	Math and trigonometry	Returns the tangent of a number.
TANH	Math and trigonometry	Returns the hyperbolic tangent of a number.
TBILLEQ	Financial	Returns the bond-equivalent yield for a Treasury bill.
TBILLPRICE	Financial	Returns the price per \$100 face value for a Treasury bill.
TBILLYIELD	Financial	Returns the yield for a Treasury bill.
TDIST	Compatibility	Returns the Student's t-distribution.
TEXT	Text	Formats a number and converts it to text.
TEXTJOIN	Text	Combines the text from multiple ranges and/or strings, and includes a delimiter you specify between each text value that will be combined. If the delimiter is an empty text string, this function will effectively concatenate the ranges.
TIME	Date and time	Returns the serial number of a particular time.
TIMEVALUE	Date and time	Converts a time in the form of text to a serial number.
TINV	Compatibility	Returns the inverse of the Student's t-distribution.
TODAY	Date and time	Returns the serial number of today's date.
TRANSPOSE	Lookup and reference	Returns the transpose of an array.
TREND	Statistical	Returns values along a linear trend.

TRIM	Text	Removes spaces from text.
TRIMMEAN	Statistical	Returns the mean of the interior of a data set.
TRUE	Logical	Returns the logical value TRUE.
TRUNC	Math and trigonometry	Truncates a number to an integer.
TTEST	Compatibility	Returns the probability associated with a Student's t-test.
TYPE	Information	Returns a number indicating the data type of a value.
UNICHAR	Text	Returns the Unicode character that is references by the given numeric value.
UNICODE	Text	Returns the number (code point) that corresponds to the first character of the text.
UPPER	Text	Converts text to uppercase.
VALUE	Text	Converts a text argument to a number.
VAR	Compatibility	Estimates variance based on a sample.
VAR.P	Statistical	Calculates variance based on the entire population.
VAR.S	Statistical	Estimates variance based on a sample.
VARA	Statistical	Estimates variance based on a sample, including numbers, text, and logical values.
VARP	Compatibility	Calculates variance based on the entire population.
VARPA	Statistical	Calculates variance based on the entire population, including numbers, text, and logical values.
VDB	Financial	Returns the depreciation of an asset for a specified or partial period by using a declining balance method.
VLOOKUP	Lookup and reference	Looks in the first column of an array and moves across the row to return the value of a cell.
WEEKDAY	Date and time	Converts a serial number to a day of the week.
WEEKNUM	Date and time	Converts a serial number to a number representing where the week falls numerically with a year.
WEIBULL	Compatibility	Calculates variance based on the entire population, including numbers, text, and logical values.
WEIBULL.DIST	Statistical	Returns the Weibull distribution.
WORKDAY	Date and time	Returns the serial number of the date before or after a specified number of workdays.
WORKDAY.INTL	Date and time	Returns the serial number of the date before or after a specified number of workdays using parameters to indicate which and how many days are weekend days.

XIRR	Financial	Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.
XNPV	Financial	Returns the net present value for a schedule of cash flows that is not necessarily periodic.
XOR	Logical	Returns a logical exclusive OR of all arguments.
YEAR	Date and time	Converts a serial number to a year.
YEARFRAC	Date and time	Returns the year fraction representing the number of whole days between start_date and end_date.
YIELD	Financial	Returns the yield on a security that pays periodic interest.
YIELDDISC	Financial	Returns the annual yield for a discounted security; for example, a Treasury bill.
YIELDMAT	Financial	Returns the annual yield of a security that pays interest at maturity.
Z.TEST	Statistical	Returns the one-tailed probability-value of a z-test.
ZTEST	Compatibility	Returns the one-tailed probability-value of a z-test.

## Set Formula to Range

In GcExcel Java, users can set formula to a cell range using the **setFormula** ('**setFormula Method**' in the **on-line documentation**) method of the **IRange** ('**IRange Interface**' in the **on-line documentation**) interface.

In order to add custom names and set formula to a range in a worksheet, refer to the following example code.

Java

```
// Add custom name and set formula to range
worksheet.getNames().add("test1", "=Sheet1!$A$1");
worksheet.getNames().add("test2", "=Sheet1!test1*2");
worksheet.getRange("A1").setValue(1);

// C6's value is 1.
worksheet.getRange("C6").setFormula("=test1");

// C7's value is 3.
worksheet.getRange("C7").setFormula("=test1 + test2");

// C8's value is 6.283185307
worksheet.getRange("C8").setFormula("=test2*PI()");
```



**Note:** Formula values are stored in a cache. Users can verify the cached value by invoking the Dirty method of the IRange interface. This method eliminates the cached value of the specified range and all the ranges dependent on it, or the entire workbook.

## Reference style

GcExcel Java supports the R1C1 reference style in order to enable users to execute calculations easily and quickly. To set reference style, you can use the **setReferenceStyle** ('**setReferenceStyle Method**' in the **on-line documentation**) method of the **IWorkbook** ('**IWorkbook Interface**' in the **on-line documentation**) interface.

In order to see how reference style can be set in a workbook, refer to the following example code.

Java

```
// set workbook's reference style to R1C1.
workbook.setReferenceStyle(ReferenceStyle.R1C1);
```

## Set Table Formula

Table formula refers to a formula that is used as a structured reference in a worksheet instead of an explicit cell reference.

While creating a table formula, users must apply structured reference (the combination of table and column names in a spreadsheet) along with the syntax rules.

For example, let's refer to the table formula in a worksheet as shown below.

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

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

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

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

### Reference operators

In GcExcel Java, you can use reference operators in order to combine column specifiers in a table formula.

Refer to the following table that describes the reference operators along with structured reference components and cell range corresponding to the table formula.

Operators	Description	Example
: (colon) range operator	All of the cells in two or more adjacent columns.	=DeptSales[[SalesPerson]:[Region]]
, (comma) union operator	A combination of two or more columns.	=DeptSales[SalesAmount],DeptSales[ComAmt]
(space) intersection operator	The intersection of two or more columns.	=DeptSales[[SalesPerson]:[SalesAmount]]DeptSales[[Region]:[ComPct]]

### Special item specifier

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

GcExcel Java 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.

Java
<pre>Object[][] data = new Object[][] {     { "SalesPerson", "Region", "SalesAmount", "ComPct", "ComAmt" },     { "Joe", "North", 260, 0.10, null },     { "Robert", "South", 660, 0.15, null }, }; worksheet.getRange("A1:E3").setValue(data); worksheet.getTables().add(worksheet.getRange("A1:E3"), true); worksheet.getTables().get(0).setName("DeptSales"); worksheet.getTables().get(0).getColumns().get("ComPct").getDataBodyRange().setNumberFormat("0%");  // Use table formula in table range. worksheet.getTables().get(0).getColumns().get("ComAmt").getDataBodyRange().setFormula("= [@ComPct]*[@SalesAmount]");</pre>



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

## Set Array Formula

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

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

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

In GcExcel Java, you can use **setFormulaArray** ('**setFormulaArray Method**' in the **on-line documentation**) method of the **IRange** ('**IRange Interface**' in the **on-line documentation**) interface to set array formula for a range. In case, you want to find out whether a range has array formula or not, you can use the **getHasArray** ('**getHasArray Method**' in the **on-line documentation**) method of the **IRange** interface. In order to get an entire array if specified range is part of an array, you can use **getCurrentArray** ('**getCurrentArray Method**' in the **on-line documentation**) method.

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

```
Java

// Setting cell value using arrays
worksheet.getRange("E4:J5").setValue(new Object[][] { { 1, 2, 3 }, { 4, 5, 6 } });

worksheet.getRange("I6:J8").setValue(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.getRange("O9:Q11").setFormulaArray("=E4:G5*I6:J8");

// Verify if Range O9 has array formula.
if (worksheet.getRange("O9").getHasArray()) {
```

```
// Set Range O9's entire array's interior color.
IRange currentarray = worksheet.getRange("O9").getCurrentArray();
currentarray.getInterior().setColor(Color.GetGreen());
}
```

## Precedents and Dependents

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

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

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

### Precedents

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

#### Java

```
private static void GetPrecedents() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Set Formula in Cell E2
    worksheet.getRange("E2").setFormula("=sum(A1:A2, B4,C1:C3)");
    // Set Value of Cells
    worksheet.getRange("A1").setValue(1);
    worksheet.getRange("A2").setValue(2);
    worksheet.getRange("B4").setValue(3);
    worksheet.getRange("C1").setValue(4);
    worksheet.getRange("C2").setValue(5);
    worksheet.getRange("C3").setValue(6);

    // Get Precedent cells of Range E2
    for (IRange item : worksheet.getRange("E2").getPrecedents()) {
        item.getInterior().setColor(Color.GetPink());
    }
}
```

```
// Saving workbook to Xlsx
workbook.save("36-Precedents.xlsx", SaveFileFormat.Xlsx);
```

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

	A	B	C	D	E	F
1	1		4			
2	2		5		21	
3			6			
4		3				
5						
6						

## Dependents

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

Java

```
private static void GetDependents() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Set Value of Cell A1
    worksheet.getRange("A1").setValue(100);
    // Set Formula in Cell C1
    worksheet.getRange("C1").setFormula("=$A$1");
    // Set Formula in Range E1:E5
    worksheet.getRange("E1:E5").setFormula("=$A$1");

    // Get Dependent cells of Range A1
    for (IRange item : worksheet.getRange("A1").getDependents()) {
        item.getInterior().setColor(Color.GetLightGreen());
    }

    // Saving workbook to Xlsx
    workbook.save("35-Dependents.xlsx", SaveFileFormat.Xlsx);
}
```

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

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

### All Precedents

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

Java

```
private static void GetAllPrecedents() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);
    // Set Formula in Cell E2
    worksheet.getRange("E2").setFormula("=sum(C1:C2)");
    // Set Formula in Cell C1
    worksheet.getRange("C1").setFormula("=B1");
    // Set Formula in Cell B1
    worksheet.getRange("B1").setFormula("=sum(A1:A2)");
    // Set Value of Cells
    worksheet.getRange("A1").setValue(1);
    worksheet.getRange("A2").setValue(2);
    worksheet.getRange("C2").setValue(3);

    // Get Precedent cells of Range E2
    ArrayList<IRange> list = new ArrayList<IRange>();
    for (IRange item : worksheet.getRange("E2").getPrecedents()) {
        list.add(item);
    }

    while (list.size() > 0) {
        ArrayList<IRange> temp = list;
        list = new ArrayList<IRange>();
        for (IRange item : temp) {
            for (int i = 0; i < item.getRowCount(); i++) {
                for (int j = 0; j < item.getColumnCount(); j++) {
                    List<IRange> dependents = item.getCells().get(i, j).getPrecedents();
                    if (dependents.size() == 0) {
                        item.getCells().get(i,
```

```

j).getInterior().setColor(Color.GetSkyBlue());
        } else {
            item.getCells().get(i,
j).getInterior().setColor(Color.GetLightGreen());
            list.addAll(dependents);
        }
    }
}
}

// Saving workbook to Xlsx
workbook.save("37-GetAllPrecedents.xlsx", SaveFileFormat.Xlsx);

```

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

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

## Custom Functions

GcExcel Java offers extensive support for adding custom functions, thus enabling users to implement custom arithmetic logic to spreadsheets. Custom functions run extremely fast, can be used to make web service calls, work similar to the native Excel functions, and can be used across all Excel platforms including major operating systems (Windows, Mac, Mobile OS and Office: both online and offline).

For example, you can make use of company's proprietary functions, apply a nested formula with custom functions, or use a combination of standard built-in functions while handling complex spreadsheet calculations.

In order to implement custom functions in GcExcel Java, you need to derive a class from the **CustomFunction** ('CustomFunction Class' in the on-line documentation) class and declare the custom function in the newly created class along with the function name, return type, and parameters.


### Using Code

- Step 1: Define a custom function
- Step 2: Register the custom function in your worksheet using the **AddCustomFunction()** ('AddCustomFunction Method' in the on-line documentation) method
- Step 3: Implement the custom function

Shared below are some examples of custom functions that can be created and used to perform complex calculation tasks:

- **Example 1: Conditional Sum Function**
- **Example 2: Custom Concatenation Function**

- **Example 3: Merged Range Function**
- **Example 4: Error Detection Function**

 **Note:** GcExcel Java doesn't allow users to export custom functions i.e. saving custom functions to an excel file is not supported. If a user tries to do so, the #NAME exception will be thrown.

### Example 1: Conditional Sum Function

In order to create and use custom conditional sum function in your spreadsheet, refer to the following example code. This function can sum cell values based on the desired display format or style (like cells with interior color as red).

#### Java

```
// Step 1- Defining custom function: MyConditionalSum
// Creating a new class MyConditionalSumFunctionX by inheriting the CustomFunction class
class MyConditionalSumFunctionX extends CustomFunction
{
    public MyConditionalSumFunctionX()
    {
        super("MyConditionalSum", FunctionValueType.Number, CreateParameters());
    }
    private static Parameter[] CreateParameters()
    {
        Parameter[] parameters = new Parameter[254];
        for (int i = 0; i < 254; i++)
        {
            parameters[i] = new Parameter(FunctionValueType.Object, true);
        }
        return parameters;
    }
    @Override
    public Object evaluate(Object[] arguments, ICalcContext context)
    {
        double sum = 0d;
        for (Object argument : arguments)
        {
            Iterable<Object> iterator = toIterable(argument);
            for (Object item : iterator)
            {
                if (item instanceof CalcError)
                {
                    return item;
                }
                else if (item instanceof Double)
                {
                    sum += (double) item;
                }
            }
        }
    }
}
```

```
        return sum;
    }
    private static Iterable<Object> toIterable(Object obj) {
        if (obj instanceof Iterable)
        {
            return (Iterable) obj;
        }
        else if (obj instanceof Object[][])
        {
            List<Object> list = new ArrayList<Object>();
            Object[][] array = (Object[][]) obj;
            for (int i = 0; i < array.length; i++)
            {
                for (int j = 0; j < array[i].length; j++)
                {
                    list.add(array[i][j]);
                }
            }
            return list;
        }
        else if (obj instanceof CalcReference)
        {
            List<Object> list = new ArrayList<Object>();
            CalcReference reference = (CalcReference) obj;
            for (IRange range : reference.getRanges())
            {
                int rowCount = range.getRows().getCount();
                int colCount = range.getColumns().getCount();
                for (int i = 0; i < rowCount; i++)
                {
                    for (int j = 0; j < colCount; j++)
                    {
                        if (range.getCells().get(i,
j).getDisplayFormat().getInterior().getColor().equals(Color.getRed()))
                        {
                            list.add(range.getCells().get(i, j).getValue());
                        }
                    }
                }
            }
            return list;
        }
        else
        {
            List<Object> list = new ArrayList<Object>();
            list.add(obj);
            return list;
        }
    }
}
```

```

    }
}

```

#### Java

```

// Step 2: Register the custom function using the AddCustomFunction method.
Workbook workbook = new Workbook();
Workbook.AddCustomFunction(new MyConditionalSumFunctionX());
IWorksheet worksheet = workbook.getActiveSheet();

// Step 3: Implement the custom function
worksheet.getRange("A1:A10").setValue(new Object[][]
{
    { 1 }, { 2 }, { 3 }, { 4 }, { 5 },
    { 6 }, { 7 }, { 8 }, { 9 }, { 10 }
});
IFormatCondition cellValueRule = (IFormatCondition)
worksheet.getRange("A1:A10").getFormatConditions()
.add(FormatConditionType.CellValue, FormatConditionOperator.Greater, 5, null);
cellValueRule.getInterior().setColor(Color.getRed());
// Sum cells value which display format interior color are red.
worksheet.getRange("C1").setFormula("=MyConditionalSum(A1:A10)");
// Range["C1"]'s value is 40.
Object result = worksheet.getRange("C1").getValue();
// Display result in cell D1
worksheet.getRange("D1").setValue(result);

```

### Example 2: Custom Concatenation Function

In order to create and use custom concatenation function in your spreadsheet, refer to the following example code.

#### Java

```

// Step 1- Defining custom function: MyConcatenate
// Creating a new class MyConcatenateFunctionX by inheriting the CustomFunction class
class MyConcatenateFunctionX extends CustomFunction
{
    public MyConcatenateFunctionX() {
        super("MyConcatenate", FunctionValueType.Text, CreateParameters());
    }
    static Parameter[] CreateParameters()
    {
        Parameter[] parameters = new Parameter[254];
        for (int i = 0; i < 254; i++)
        {
            parameters[i] = new Parameter(FunctionValueType.Variant);
        }
        return parameters;
    }
}

```



```

@Override
public Object evaluate(Object[] arguments, ICalcContext context)
{
    StringBuilder sb = new StringBuilder();
    for (Object argument : arguments)
    {
        if (argument instanceof CalcError)
        {
            return argument;
        }
        if (argument instanceof String || argument instanceof Double) {
            sb.append(argument);
        }
    }
    return sb.toString();
}
}

```

#### Java

```

// Step 2: Register the custom function using the AddCustomFunction method.
Workbook workbook = new Workbook();
Workbook.AddCustomFunction(new MyConcatenateFunctionX());
IWorksheet worksheet = workbook.getActiveSheet();

// Step 3: Implement the custom function
worksheet.getRange("A1").setFormula("=MyConcatenate(\"I\", \" \", \"work\", \" \", \"with\", \" \", \"GcExcel\", \".\")");
worksheet.getRange("A2").setFormula("=MyConcatenate(A1, \"Documents.\")");
// Value of cell A1 is "I work with GcExcel."
Object resultA1 = worksheet.getRange("A1").getValue();
// Value of cell A2 is "I work with GcExcel Documents."
Object resultA2 = worksheet.getRange("A2").getValue();
// Display result in cell D1
worksheet.getRange("D1").setValue(resultA2);

```

### Example 3: Merged Range Function

In order to create and use custom merged range function in your spreadsheet, refer to the following example code.

#### Java

```

// Step 1- Defining custom function: MyIsMergedRange
// Creating a new class MyIsMergedRangeFunctionX by inheriting the CustomFunction class
class MyIsMergedRangeFunctionX extends CustomFunction
{
    public MyIsMergedRangeFunctionX()
    {
        super("MyIsMergedRange", FunctionValueType.Boolean,

```

```

        new Parameter[] { new Parameter(FunctionValueType.Object, true) });
    }
    @Override
    public Object evaluate(Object[] arguments, ICalcContext context)
    {
        if (arguments[0] instanceof CalcReference) {
            if (arguments[0] instanceof CalcReference) {
                List<IRange> ranges = ((CalcReference) arguments[0]).getRanges();
                for (IRange range : ranges) {
                    return range.getMergeCells();
                }
            }
        }
        return false;
    }
}

```

#### Java

```

// Step 2: Register the custom function using the AddCustomFunction method.
Workbook workbook = new Workbook();
Workbook.AddCustomFunction(new MyIsMergedRangeFunctionX());
IWorksheet worksheet = workbook.getActiveSheet();

// Step 3: Implement the custom function
worksheet.getRange("A1:B2").merge();
worksheet.getRange("C1").setFormula("=MyIsMergedRange(A1)");
worksheet.getRange("C2").setFormula("=MyIsMergedRange(H2)");
// A1 is a merged cell, getRange("C1")'s value is true.
Object resultC1 = worksheet.getRange("C1").getValue();
// H2 is not a merged cell, getRange("C2")'s value is false.
Object resultC2 = worksheet.getRange("C2").getValue();
// Display result in cell D1
worksheet.getRange("D1").setValue(resultC2);

```

#### Example 4: Error Detection Function

In order to create and use custom error detection function in your spreadsheet, refer to the following example code.

#### Java

```

// Step 1- Defining custom function: MyIsError
// Creating a new class MyIsErrorFunctionX by inheriting the CustomFunction class
class MyIsErrorFunctionX extends CustomFunction
{
    public MyIsErrorFunctionX()
    {
        super("MyIsError", FunctionValueType.Boolean, new Parameter[]{new
Parameter(FunctionValueType.Variant)});
    }
}

```

```
    }
    @Override
    public Object evaluate(Object[] arguments, ICalcContext context)
    {
        if (arguments[0] instanceof CalcError)
        {
            if ((CalcError) arguments[0] != CalcError.None && (CalcError)
arguments[0] != CalcError.GettingData)
            {
                return true;
            } else
            {
                return false;
            }
        }
        return false;
    }
}
```

#### Java

```
// Step 2: Register the custom function using the AddCustomFunction method.
Workbook workbook = new Workbook();
Workbook.AddCustomFunction(new MyIsErrorFunctionX());
IWorksheet worksheet = workbook.getActiveSheet();

// Step 3: Implement the custom function
worksheet.getRange("A1").setValue(CalcError.Num);
worksheet.getRange("A2").setValue(100);
worksheet.getRange("B1").setFormula("=MyIsError(A1)");
worksheet.getRange("B2").setFormula("=MyIsError(A2)");
// getRange("B1")'s value is true.
Object resultB1 = worksheet.getRange("B1").getValue();
// getRange("B2")'s value is false.
Object resultB2 = worksheet.getRange("B2").getValue();
// Display result in cell D2
worksheet.getRange("D2").setValue(resultB2);
```

## Shapes And Pictures

GcExcel Java allows users to insert drawing objects like shapes and pictures on cells of a spreadsheet.

You can draw and insert arrows, lines, pictures and general shapes of your choice based on the specific requirements.

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

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

## Connector

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

Java

```
// To configure the connector shape
IShape ShapeBegin = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 1, 1, 100, 100);
IShape EndBegin = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 200, 200, 100, 100);
IShape ConnectorShape = worksheet.getShapes().addConnector(ConnectorType.Straight, 1, 1, 101, 101);

// To detach the ends of the connector to other shapes
ConnectorShape.getConnectorFormat().beginConnect(ShapeBegin, 3);
ConnectorShape.getConnectorFormat().endConnect(EndBegin, 0);
```



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

## Shape

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

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

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

Java

```
// create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Create shape with custom name
IShape shape = worksheet.getShapes().addShape("Custom Name to Balloon Shape",
AutoShapeType.Balloon, 50, 50, 100, 200);

// Get shape by custom name
IShape balloonShape = worksheet.getShapes().get("Custom Name to Balloon Shape");
```

```
balloonShape.setFill().getColor().setRGB(Color.GetGreen());

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

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

#### Java

```
// create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
IShape shape = worksheet.getShapes().addChart("Area Chart with CustomName",
ChartType.Area, 250, 20, 360, 230);
worksheet.getRange("A1:C13").setValue(new Object[][] {
{ null, "Blue Series", "Orange Series" },
{ "Jan", 0, 59.1883603948205 },
{ "Feb", 44.6420211591501, 52.2280901938606 },
{ "Mar", 45.2174930051225, 49.8093056416248 },
{ "Apr", 62, 37.3065749226828 },
{ "May", 53, 34.4312192530766 },
{ "Jun", 31.8933622049831, 69.7834561753736 },
{ "Jul", 41.7930895085093, 63.9418103906982 },
{ "Aug", 73, 57.4049534494926 },
{ "Sep", 49.8773891668518, 33 },
{ "Oct", 50, 74 },
{ "Nov", 54.7658428630216, 22.9587876597096 },
{ "Dec", 32, 54 },
});

//Get chart by custom name
IShape areaChart = worksheet.getShapes().get("Area Chart with CustomName");
areaChart.getChart().getSeriesCollection().add(worksheet.getRange("A1:C13"),
RowCol.Columns);
areaChart.getChart().getChartTitle().setText("Area Chart");

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

#### Picture

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

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

#### Java

```
// Add shape of picture type
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 20, 20, 100,
100);
String path = "C:\\Users\\GPCTAdmin\\Pictures\\cat.jpg";

try {
    FileInputStream stream = new FileInputStream(path);
    shape.getFill().userPicture(stream, ImageType.JPG);
    stream.close();
} catch (IOException e) {
    e.printStackTrace();
}

// Recolor the picture
shape.getPictureFormat().setColorType(PictureColorType.Grayscale);

// Set picture brightness and contrast ratio
shape.getPictureFormat().setBrightness(0.6);
shape.getPictureFormat().setContrast(0.3);

// Set height, width, x-axis offset and y-axis offset of the specified picture
shape.getPictureFormat().getCrop().setPictureOffsetX(10);
shape.getPictureFormat().getCrop().setPictureOffsetY(-5);
shape.getPictureFormat().getCrop().setPictureWidth(120);
shape.getPictureFormat().getCrop().setPictureHeight(80);
```

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

#### Java

```
// create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Create shape with custom name
IShape shape = worksheet.getShapes().addPicture("Custom Name to Image", "image.png", 10,
10, 250, 150);

// Get the picture name
System.out.println(shape.getName().toString());

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

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

[Customize Shape Format and Shape Text](#)

[Hyperlink on Shape](#)

[Group or Ungroup Shapes](#)[Shape Adjustment](#)[Background Image](#)[Size and Position of Image](#)

## Customize Shape Format and Shape Text

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

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

### Shape Format

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

#### Solid Fill

To format the shape with Solid fill, first you need to use the **Solid** ('**solid Method** in the on-line documentation') method of the **IFillFormat** interface to specify the fill format and then set the **setRGB** ('**setRGB Method** in the on-line documentation') and **setTransparency** ('**setTransparency Method** in the 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.

Java

```
// Solid fill
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Parallelogram, 1, 1, 200, 100);
shape.getFill().solid();
shape.getFill().getColor().setRGB(Color.GetRed());
```

#### Gradient Fill

With gradient fill, you can configure the shape fill to the gradient fill using the **oneColorGradient** ('**oneColorGradient Method** in the on-line documentation') method, **twoColorGradient** ('**twoColorGradient Method** in the on-line documentation') method or **presetGradient** ('**presetGradient Method** in the on-line documentation') method of the **IFillFormat** ('**IFillFormat Interface** in the on-line documentation') interface.

After setting the gradient fill, you can insert, delete or change gradient stops; configure the fill style rotation along with the shape and the angle of the gradient fill via the **getGradientStops** ('**getGradientStops Method** in the on-line documentation') method, **setRotateWithObject** ('**setRotateWithObject Method** in the on-line documentation') method and **setGradientAngle** ('**setGradientAngle Method** in the on-line documentation') method of the **IFillFormat** ('**IFillFormat Interface** in the on-line documentation') interface.

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

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

Java

```
// Gradient fill
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Heart, 1, 1, 100, 100);
shape.getFill().presetGradient(GradientStyle.Vertical, 3, PresetGradientType.Silver);
shape.getFill().setRotateWithObject(false);
```

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

Java

```
// Initialize workbook
```

```
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

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

// Init a two color gradient fill.
rectangle.setFill().twoColorGradient(GradientStyle.Horizontal, 1);

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

To set the radial, rectangular or path gradient fill, you also need to set the **PathShapeType** along with using the **twoColorGradient** method. Refer to the following example code to fill the shape with 'Radial' gradient fill.

```
Java

// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

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

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

rectangle.setFill().getGradientPathType().equals(PathShapeType.Radial);

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

### Pattern Fill

With pattern fill, you can set the shape fill to pattern fill using the **patterned** ('**patterned Method** in the on-line documentation) method of the **IFillFormat** ('**IFillFormat Interface** in the on-line documentation) interface.

Further, you can also configure the background color and the pattern color using **setObjectThemeColor** ('**setObjectThemeColor Method** in the on-line documentation) method of the **IColorFormat** ('**IColorFormat Interface** in the on-line documentation) interface and **getPatternColor** ('**getPatternColor Method** in the on-line documentation) method of the **IFillFormat** ('**IFillFormat Interface** in the on-line documentation) interface.

In order to fill the shape with pattern fill, refer to the following example code.

```
Java

// Pattern fill
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 1, 1, 100, 100);
shape.setFill().patterned(PatternType.Percent10);
shape.setFill().getColor().setObjectThemeColor(ThemeColor.Accent2);
shape.setFill().getPatternColor().setObjectThemeColor(ThemeColor.Accent6);
```

### Picture Fill

In picture fill, you can use the **addShape** ('**addShape Method** in the on-line documentation) method of the **IShapes** ('**IShapes Interface** in the on-line documentation) interface to insert the shape that you want to fill with a picture.

Also, you can configure the picture format with characteristics like picture height, picture width, brightness, contrast ratio, re-coloring, x-axis and y-axis offset etc using the methods of the **IPictureFormat** ('**IPictureFormat Interface** in the on-line documentation) interface.

In order to fill the shape with picture, refer to the following example code.



Java

```
// Add shape of picture type
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 20, 20, 100, 100);
String path = "C:\\Users\\GPCTAdmin\\Pictures\\cat.jpg";

try {
    FileInputStream stream = new FileInputStream(path);
    shape.getFill().userPicture(stream, ImageType.JPG);
    stream.close();
} catch (IOException e) {
    e.printStackTrace();
}

// Recolor the picture
shape.getPictureFormat().setColorType(PictureColorType.Grayscale);

// Set picture brightness and contrast ratio
shape.getPictureFormat().setBrightness(0.6);
shape.getPictureFormat().setContrast(0.3);

// Set height, width, x-axis offset and y-axis offset of the specified picture
shape.getPictureFormat().getCrop().setPictureOffsetX(10);
shape.getPictureFormat().getCrop().setPictureOffsetY(-5);
shape.getPictureFormat().getCrop().setPictureWidth(120);
shape.getPictureFormat().getCrop().setPictureHeight(80);
```

### Texture Fill

Using texture fill, you can fill the shape with texture of your choice using the **presetTextured** ('**presetTextured Method**' in the on-line documentation) method of the **IFillFormat** ('**IFillFormat Interface**' in the on-line documentation) interface.

Further, you can also configure the layout of the texture using the **setTextureAlignment** ('**getTextureAlignment Method**' in the on-line documentation) method, **setTextureHorizontalScale** ('**getTextureHorizontalScale Method**' in the on-line documentation) method, **setTextureOffsetX** ('**getTextureOffsetX Method**' in the on-line documentation) method, **setTextureOffsetY** ('**getTextureOffsetY Method**' in the on-line documentation) method and **setTextureVerticalScale** ('**getTextureVerticalScale Method**' in the on-line documentation) method.

In order to fill the shape with texture fill, refer to the following example code.

Java

```
// Texture Fill
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 1, 1, 100, 100);
shape.getFill().presetTextured(PresetTexture.Canvas);
shape.getFill().setTextureAlignment(TextureAlignment.Center);
shape.getFill().setTextureOffsetX(2.5);
shape.getFill().setTextureOffsetY(3.2);
shape.getFill().setTextureHorizontalScale(0.9);
shape.getFill().setTextureVerticalScale(0.2);
shape.getFill().setTransparency(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** ('**ILineFormat Interface**' in the on-line documentation) interface.

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

Java

```
// To set shape's line style.
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 1, 1, 100, 100);
shape.getLine().setDashStyle(LineDashStyle.Dash);
```

```
shape.getLine().setStyle(LineStyle.Single);
shape.getLine().setWeight(2);
shape.getLine().getColor().setObjectThemeColor(ThemeColor.Accent6);
shape.getLine().setTransparency(0.3);
```

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

### 3D Formatting

GcExcel Java enables users to format the three-dimensional layout for the inserted shape via configuring its rotation degree around x, y and z axis. This can be done using the **setRotationX** ('**setRotationX Method** in the on-line documentation') method, the **setRotationY** ('**setRotationY Method** in the on-line documentation') method and the **setRotationZ** ('**setRotationZ Method** in the on-line documentation') method of the **IThreeDFormat** interface.

In order to apply 3D formatting to the embedded shape, refer to the following example code.

```
Java

// To set rotation degree for the shape around x, y, z axis.
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 1, 1, 100, 100);
shape.getThreeD().setRotationX(50);
shape.getThreeD().setRotationY(20);
shape.getThreeD().setRotationZ(30);
shape.getThreeD().setDepth(7);
shape.getThreeD().setZ(20);
```

### Shape Text

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

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

```
Java

// To configure the text and text style of the shape.
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Rectangle, 40, 40, 100, 100);
shape.getTextFrame().getTextRange().getFont().getColor().setRGB(Color.FromArgb(0, 255, 0));
shape.getTextFrame().getTextRange().getFont().setBold(true);
shape.getTextFrame().getTextRange().getFont().setItalic(true);
shape.getTextFrame().getTextRange().getFont().setSize(20);
shape.getTextFrame().getTextRange().getFont().setStrikethrough(true);
shape.getTextFrame().getTextRange().getParagraphs().add("This is a rectangle shape.");
shape.getTextFrame().getTextRange().getParagraphs().add("My name is xxx.");
shape.getTextFrame().getTextRange().getParagraphs().get(1).getRuns().add("Hello World!");
shape.getTextFrame().getTextRange().getParagraphs().get(1).getRuns().get(0).getFont().setStrikethrough(false);
shape.getTextFrame().getTextRange().getParagraphs().get(1).getRuns().get(0).getFont().setSize(35);
```

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

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

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

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

```
Java

// create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
```


```
IWorksheet worksheet = workbook.getWorksheets().get(0);

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

// Add two paragraphs for the shape.
shape.getTextFrame().getTextRange().getParagraphs().add("GrapeCity Documents for Excel");
shape.getTextFrame().getTextRange().getParagraphs().add("Middle Centered");


// Centers text vertically.
shape.getTextFrame().setVerticalAnchor(VERTICALAnchor.AnchorMiddle);
// Centers text horizontally.
shape.getTextFrame().setHorizontalAnchor(HORIZONTALAnchor.Center);

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

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

## Hyperlink on Shape

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

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

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

1. The **setAddress** (**'setAddress Method' in the on-line documentation**) and **setSubAddress** (**'setSubAddress Method' in the on-line documentation**) methods of the **IHyperlink** interface can be used to configure the hyperlink references. The table shown below illustrates both the properties with examples:

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

2. The **setEmailSubject** (**'setEmailSubject Method' in the on-line documentation**) method can be used to set the text of hyperlink's email subject line.
3. The **setScreenTip** (**'setScreenTip Method' in the on-line documentation**) method can be used to set the tip text for the specified hyperlink.
4. The **setTextToDisplay** (**'setTextToDisplay Method' in the on-line documentation**) method can be used to set the text to be displayed for the specified hyperlink.

### Add Hyperlink

A user can add hyperlink to a shape in a worksheet using the **Add ('add Method' in the on-line documentation)** method of the **IHyperLinks ('IHyperlinks Interface' in the on-line documentation)** interface.

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

Java

```
// Add a hyperlink to external file

// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Add a Shape
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Oval, 1, 1, 200, 100);
shape.getTextFrame().getTextRange().getParagraphs().add("Link to Test.xlsx file");
// Add Hyperlink
worksheet.getHyperlinks().add(shape, "C:\\Test.xlsx", null, "Link to Test.xlsx file",
"Test.xlsx");

// Save to an excel file
workbook.save("402-LinkExternalFile.xlsx", SaveFileFormat.Xlsx);
```

Java

```
// Add a hyperlink to web page

// Add a Shape
IShape picture = null;
try {
    picture = worksheet.getShapes().addPicture("grapecity-logo.jpg", 1, 1, 100, 100);
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
// Add Hyperlink
worksheet.getHyperlinks().add(picture, "https://www.grapecity.com/", null, "Click to
Open", "GrapeCity");
// Save to an excel file
workbook.save("401-ShapeHyperlink.xlsx", SaveFileFormat.Xlsx);
```

Java

```
// Add a Shape
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Oval, 1, 1, 200, 100);
shape.getTextFrame().getTextRange().getParagraphs().add("Go To sheet1 J3:K4");
// Add Hyperlink
worksheet.getHyperlinks().add(shape, null, "Sheet1!J3:K4", "Go To Sheet1 D3:E4", null);

// Save to an excel file
```

```
workbook.save("403-HyperlinkRange.xlsx", SaveFileFormat.Xlsx);
```

#### Java

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

// Add a Shape
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Oval, 1, 1, 200, 100);
shape.getTextFrame().getTextRange().getParagraphs().add("Send Feedback");
// Add Hyperlink
worksheet.getHyperlinks().add(shape, "mailto:web_feedback@grapecity.com", null, "Send
your valuable feedback.",
    "Feedback");

// Save to an excel file
workbook.save("404-HyperlinkMailTo.xlsx", SaveFileFormat.Xlsx);
```

### Delete Hyperlink

The hyperlink on the shape can be removed using the **Delete** ('delete Method' in the on-line documentation) method of the **IHyperlink** interface.

Refer to the following example code to delete hyperlink.

#### Java

```
//Delete a hyperlink.

// Add a Shape
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Oval, 1, 1, 200, 100);

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

// Delete hyperlink1.
hyperlink1.delete();

// Save to an excel file
workbook.save("405-DeleteHyperlink.xlsx", SaveFileFormat.Xlsx);
```

## Group or Ungroup Shapes

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

## Group Shapes

Several shapes can be grouped together using the **Group** ('group Method' in the on-line documentation) method of the **IShapeRange** interface. The **IShapeRange** interface represents the range of the shapes which needs to be grouped together. The grouped shapes behave as a single shape.

Refer to the following example code to group shapes.

Java

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Creating shapes collection for activeSheet
IShapes shapes = worksheet.getShapes();

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

// Connecting ShapeBegin & EndBegin shapes by connector shape
ConnectorShape.getConnectorFormat().beginConnect(ShapeBegin, 3);
ConnectorShape.getConnectorFormat().endConnect(EndBegin, 0);

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

// Creating shpRange collection to group certain shapes as given in array
IShapeRange shpRange = shapes
    .getRange(new String[] { shapes.get(0).getName(), shapes.get(1).getName(),
        shapes.get(2).getName() });

// Grouping Shapes
IShape grouped = shpRange.group();
// Setting Style for Grouped shape together
grouped.getLine().getColor().setRGB(Color.GetDarkOrange());
grouped.getFill().getColor().setRGB(Color.GetLightGreen());
System.out.println("Group Name is: " + grouped.getName());

// Saving workbook to Xlsx
workbook.save("9-GroupedShapes.xlsx", SaveFileFormat.Xlsx);
```

## Ungroup Shapes

A group of shapes in a specified range can be ungrouped using the **Ungroup** ('ungroup Method' in the on-line documentation) method of the **IShape** ('IShape Interface' in the on-line documentation) interface.

Refer to the following example code to ungroup shapes.

#### Java

```
// Initialize workbook
Workbook workbook = new Workbook();
// Open workbook
workbook.open("9-GroupedShapes.xlsx");
IShapes shapes = workbook.getWorksheets().get(0).getShapes();

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

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

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

## Shape Adjustment

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

GcExcel provides the **Adjustments ('getAdjustments Method' in the on-line documentation)** method in the **IShape ('IShape Interface' in the on-line documentation)** interface to get a collection of adjustment values for the specified AutoShape or Connector.

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

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

Radial	Value 1.0 represents the shape width. Hence, the maximum value for radial adjustment is 0.5, which is half way across the shape.
Angle	Value is expressed in degrees. If you specify the value outside the range of 180 degree, it will be normalized to be within that range.

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

### Using Code

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

Java

```
private static void AdjustmentPointForShape() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);
    // Add a right arrow callout
    IShape shape = worksheet.getShapes().addShape(AutoShapeType.RightArrowCallout, 20,
20, 200, 100);

    // Set adjustment points for shapes
    IAdjustments adjustments = shape.getAdjustments();

    // To count adjustment points
    int c = adjustments.getCount();
    System.out.println("Count of Adjustment Values: " + c);

    adjustments.set(0, 0.5); // arrow neck width
    adjustments.set(1, 0.4); // arrow head width
    adjustments.set(2, 0.5); // arrow head height
    adjustments.set(3, 0.6); // text box width

    // Saving workbook to Xlsx
    workbook.save("17-AdjustmentPointForShape.xlsx", SaveFileFormat.Xlsx);
}
```

## Background Image

GcExcel allows you to set background image in a worksheet using the **setBackgroundPicture** ('setBackgroundPicture Method' in the on-line documentation) method of the **IWorksheet** ('IWorksheet Interface' in the on-line documentation) interface. The background image can be saved to Excel and is rendered multiple times, side by side, to cover the whole area of the worksheet.

### Using Code

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



## Java

```
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("A1").setValue("GrapeCity Documents for Excel");
worksheet.getRange("A1").getFont().setSize(25);

// Load an image from a specific file in input stream
InputStream inputStream = new FileInputStream("grapecity.png");
try {
    byte[] bytes = new byte[inputStream.available()];
    // Read an image from input stream
    inputStream.read(bytes, 0, bytes.length);

    // Add background image of the worksheet
    worksheet.setBackgroundPicture(bytes);
} catch (IOException ioe) {
    ioe.printStackTrace();
}

// Save workbook
workbook.save("PrintBackgroundPicture.xlsx", SaveFileFormat.Xlsx);
```

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

## Size and Position of Image

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

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

### Using Code

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

## Java

```
IWorkbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
IRange range = worksheet.getRange("A1:D4");

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

```
// Add the image to the Range["D4:H8"].
worksheet.getShapes().addPictureInPixel("image.png", rect.getX(), rect.getY(),
rect.getWidth(), rect.getHeight());

workbook.save("GetRangePosition.xlsx");
```

## Image Transparency

GcExcel supports controlling the transparency of an image by providing **setTransparency** method in **IPictureFormat** interface. The value of transparency can vary between 0.0 (opaque) to 1.0 (clear).

### Using Code

Refer to the following example code to set the transparency of an image.

```
Java

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

// Use sheet index to get worksheet.
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Add a picture
IShape picture = worksheet.getShapes().addPicture("Image.png", 10, 10, 200, 100);

// Set picture's transparency as 60%.
picture.getPictureFormat().setTransparency(0.6);

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

### Limitation


SpreadJS does not support image transparency, hence this info would be lost when using json I/O.

## Control Position of Overlapping Shapes

The order of overlapping shapes in a worksheet is decided by their z-order positions. GcExcel allows its users to set the z-order of shapes so that their positions can be controlled while creating flow charts or business diagrams etc.

The **zOrder** method in GcExcel API can be used to move the specified shape in front of or behind the other shapes. It takes **ZOrderType** enum as a parameter to specify the position of a shape relative to the other shapes.

The **getZOrderPosition** method of the **IShape** interface can be used to retrieve the position of a specified shape in the z-order.

 **Note:** If the z-order of a shape is changed, the index of the shape in Worksheet.Shapes collection is also changed.

## Using Code

Refer to the below example code to add various shapes, change their z-order and get their positions in z-order in a worksheet.

### Java

```
// Create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getActiveSheet();

IShapes shapes = worksheet.getShapes();

// Add shapes
IShape rectangle = shapes.addShape(AutoShapeType.Rectangle, 20, 20, 100, 100);
rectangle.getFill().getColor().setRGB(Color.GetBlue());

IShape oval = shapes.addShape(AutoShapeType.Oval, 50, 50, 100, 100);
oval.getFill().getColor().setRGB(Color.GetGreen());

IShape pentagon = shapes.addShape(AutoShapeType.Pentagon, 80, 80, 100, 100);
pentagon.getFill().getColor().setRGB(Color.GetRed());

IShape triangle = shapes.addShape(AutoShapeType.IsoscelesTriangle, 100, 100, 100, 100);
triangle.getFill().getColor().setRGB(Color.GetOrange());

// Set rectangle above oval
rectangle.zOrder(ZOrderType.BringForward);

// Get position of rectangle in z-order
System.out.println("Z-Order rectangle: " + rectangle.getZOrderPosition());

// Set triangle to bottom
triangle.zOrder(ZOrderType.SendToBack);

// Get position of triangle in z-order
System.out.println("Z-Order triangle: " + triangle.getZOrderPosition());

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

## Styles

GcExcel Java provides users with the ability to customize the formatting of the cells in a worksheet.

Styling a cell includes customizing the characteristics such as fonts, alignment, borders, fill (solid fill, gradient fill, pattern fill), named style and display format. Users can apply, create or remove custom cell styles based on their aesthetic

preferences and specific requirements. This helps in ensuring enhanced clarity and increased readability.

In order to apply style in a worksheet, refer to the following tasks.

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

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

Category	Description	Methods
Number Format	Cell number format.	<b>IRange.setNumberFormat ('setNumberFormat Method' in the on-line documentation)</b>  <b>IRange.getNumberFormat ('getNumberFormat Method' in the on-line documentation)</b>
Alignment	Horizontal and vertical alignment of cell content, indentation, text wrap, text rotation and text shrinking.	<b>IRange.setAddIndent ('setAddIndent Method' in the on-line documentation)</b>  <b>IRange.getAddIndent ('getAddIndent Method' in the on-line documentation)</b>  <b>IRange.getIndentLevel ('getIndentLevel Method' in the on-line documentation)</b>  <b>IRange.setIndentLevel ('setIndentLevel Method' in the on-line documentation)</b>  <b>IRange.getWrapText ('getWrapText Method' in the on-line documentation)</b>  <b>IRange.setWrapText ('setWrapText Method' in the on-line documentation)</b>  <b>IRange.setShrinkToFit ('setShrinkToFit Method' in the on-line documentation)</b>  <b>IRange.getShrinkToFit ('getShrinkToFit Method' in the on-line documentation)</b>  <b>IRange.setMergeCells ('setMergeCells Method' in the on-line documentation)</b>  <b>IRange.getMergeCells ('getMergeCells Method' in the on-line documentation)</b>  <b>IRange.getReadingOrder ('getReadingOrder Method' in the on-line documentation)</b>  <b>IRange.setReadingOrder ('setReadingOrder Method' in the on-line documentation)</b>  <b>IRange.getOrientation ('getOrientation Method' in the on-line documentation)</b>  <b>IRange.setOrientation ('setOrientation Method' in the on-line documentation)</b>

Font	The font style of the text in the cells.	<b>IRange.getFont(IFont) ('getFont Method' in the on-line documentation)</b>
Borders	Cell border line styles and colors.	<b>IRange.getBorders(IBorders) ('getBorders Method' in the on-line documentation)</b>
Fill	Cell pattern fill or gradient fill.	<b>IRange.getInterior(IInterior) ('getInterior Method' in the on-line documentation)</b>
Protection	Cell protection options (Locked and Hidden)	<b>IRange.getLocked ('getLocked Method' in the on-line documentation)</b>  <b>IRange.setLocked ('setLocked Method' in the on-line documentation)</b>  <b>IRange.getFormulaHidden ('getFormulaHidden Method' in the on-line documentation)</b>  <b>IRange.setFormulaHidden ('setFormulaHidden Method' in the on-line documentation)</b>

Besides the built-in styles, users can also create their own custom styles with description for individual cells or a range of cells in a worksheet by defining all the style attributes including font, font size, number format, alignment etc.

## Set Sheet Styling

GcExcel Java enables users to set sheet styling to 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 **getInterior ('getInterior Method' in the on-line documentation)** method of the **IRange ('IRange Interface' in the on-line documentation)** interface. A cell interior can be of three types, namely, solid fill, pattern fill and gradient fill.

#### Solid fill

You can specify the fill style for the cell as solid by setting the **setPattern ('setPattern Method' in the on-line documentation)** method of the **IInterior ('IInterior Interface' in the on-line documentation)** interface.

Refer to the following example code to set solid fill.

Java

```
// Solid fill for B5
worksheet.getRange("B5").getInterior().setPattern(Pattern.Solid);
worksheet.getRange("B5").getInterior().setColor(Color.FromArgb(255, 0, 255));
```

### Pattern fill

You can integrate pattern fill in cells using the **Pattern** method 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 the methods of the **IInterior** interface to set the background color and the foreground color as per your preferences.

Refer to the following example code to set pattern fill.

Java

```
// Pattern fill for A1
worksheet.getRange("A1").getInterior().setPattern(Pattern.LightDown);
worksheet.getRange("A1").getInterior().setColor(Color.FromArgb(255, 0, 255));
worksheet.getRange("A1").getInterior().setPatternColorIndex(5);
```

### Gradient Fill

You can integrate gradient fill in cells using the **getGradient** ('**getGradient Method** in the on-line documentation) method of the **IInterior** ('**IInterior Interface** in the on-line documentation) interface.

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

#### Linear gradient fill

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

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

Java

```
// Gradient fill for A1
worksheet.getRange("A1").getInterior().setPattern(Pattern.LinearGradient);
((ILinearGradient) worksheet.getRange("A1").getInterior().getGradient()).getColorStops().get(0)
    .setColor(Color.FromArgb(255, 0, 0));
((ILinearGradient) worksheet.getRange("A1").getInterior().getGradient()).getColorStops().get(1)
    .setColor(Color.FromArgb(255, 255, 0));
((ILinearGradient) worksheet.getRange("A1").getInterior().getGradient()).setDegree(90);
```

#### Rectangular gradient fill

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

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

Java

```
// Rectangular gradient fill for A1
worksheet.getRange("A1").getInterior().setPattern(Pattern.RectangularGradient);
((IRectangularGradient) worksheet.getRange("A1").getInterior().getGradient()).getColorStops().get(0)
    .setColor(Color.FromArgb(255, 0, 0));
((IRectangularGradient) worksheet.getRange("A1").getInterior().getGradient()).getColorStops().get(1)
    .setColor(Color.FromArgb(0, 255, 0));

((IRectangularGradient) worksheet.getRange("A1").getInterior().getGradient()).setBottom(0.2);
((IRectangularGradient) worksheet.getRange("A1").getInterior().getGradient()).setRight(0.3);
((IRectangularGradient) worksheet.getRange("A1").getInterior().getGradient()).setTop(0.4);
((IRectangularGradient) worksheet.getRange("A1").getInterior().getGradient()).setLeft(0.5);
```

### Set font

You can customize the font of a worksheet using the **getFont** ('**getFont Method** in the on-line documentation) method of **IRange** ('**IRange Interface** in the on-line documentation) interface.

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

## Java

```
// Set font
worksheet.getRange("A1").setValue("aaa");
worksheet.getRange("A1").getFont().setThemeColor(ThemeColor.Accent1);
worksheet.getRange("A1").getFont().setTintAndShade(-0.5);
worksheet.getRange("A1").getFont().setThemeFont(ThemeFont.Major);
worksheet.getRange("A1").getFont().setBold(true);
worksheet.getRange("A1").getFont().setSize(20);
worksheet.getRange("A1").getFont().setStrikethrough(true);
```

### Set border

You can customize the border of a worksheet using the **getBorders** ('getBorders Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface.

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

## Java

```
// Set border
worksheet.getRange("A1:B5").getBorders().setLineStyle(BorderLineStyle.DashDot);
worksheet.getRange("A1:B5").getBorders().setThemeColor(ThemeColor.Accent1);
worksheet.getRange("A1:B5").getBorders().get(BordersIndex.EdgeRight).setLineStyle(BorderLineStyle.Double);
worksheet.getRange("A1:B5").getBorders().get(BordersIndex.EdgeRight).setThemeColor(ThemeColor.Accent2);
worksheet.getRange("A1:B5").getBorders().get(BordersIndex.DiagonalDown).setLineStyle(BorderLineStyle.Double);
worksheet.getRange("A1:B5").getBorders().get(BordersIndex.DiagonalDown).setThemeColor(ThemeColor.Accent5);
```

### Set number format

You can set the number format in a worksheet using the **setNumberFormat** ('setNumberFormat Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface.

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

## Java

```
// Set number format
worksheet.getRange("A1").setValue(12);
worksheet.getRange("A1").setNumberFormat("$#,##0.00");
```

### Set alignment

You can customize the alignment of a worksheet using the **setHorizontalAlignment** ('setHorizontalAlignment Method' in the on-line documentation) method, **setVerticalAlignment** ('setVerticalAlignment Method' in the on-line documentation) method, **setAddIndent** ('setAddIndent Method' in the on-line documentation) method and **setReadingOrder** ('setReadingOrder Method' in the on-line documentation) methods of the **IRange** ('IRange Interface' in the on-line documentation) interface.

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

## Java

```
// Set alignment
worksheet.getRange("A1").setHorizontalAlignment(HorizontalAlignment.Distributed);
worksheet.getRange("A1").setAddIndent(true);
worksheet.getRange("A1").setVerticalAlignment(VerticalAlignment.Top);
worksheet.getRange("A1").setReadingOrder(ReadingOrder.RightToLeft);
```

### Set protection

You can set protection for your worksheet using the **setFormulaHidden** ('setFormulaHidden Method' in the on-line documentation) method and **setLocked** ('setLocked Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface.

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

Java

```
// Set protection
worksheet.getRange("A1").setLocked(true);
worksheet.getRange("A1").setFormulaHidden(true);
```

## Create and Set Custom Named Style

A custom cell style that is applied to the worksheet with a unique name is called Named style. Named styles are typically different from the built-in style names defined for a spreadsheet.

You can create and set custom named styles based on specific requirements. You can also modify an existing style and save it as a new workbook style. In GcExcel Java, Styles refers to the named style collection that stores both the built-in and custom named styles.

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

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

### Create and Set a Custom Named Style

GcExcel Java 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 methods of **IStyleCollection ('IStyleCollection Interface' in the on-line documentation)** interface. This method can also be used to return an IStyle instance. If you want to configure the named style settings in your spreadsheet, you can use the methods of the **IStyle ('IStyle Interface' in the on-line documentation)** interface.

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

Java

```
// Add custom name style.
IStyle style = workbook.getStyles().add("testStyle");

// Configure custom name style settings begin.
// Border
style.getBorders().get(BordersIndex.EdgeLeft).setLineStyle(BorderLineStyle.Thin);
style.getBorders().get(BordersIndex.EdgeTop).setLineStyle(BorderLineStyle.Thick);
style.getBorders().get(BordersIndex.EdgeRight).setLineStyle(BorderLineStyle.Double);
style.getBorders().get(BordersIndex.EdgeBottom).setLineStyle(BorderLineStyle.Double);
style.getBorders().setColor(Color.FromArgb(0, 255, 0));

// Font
style.getFont().setThemeColor(ThemeColor.Accent1);
style.getFont().setTintAndShade(0.8);
style.getFont().setItalic(true);
style.getFont().setBold(true);
style.getFont().setName("LiSu");
style.getFont().setSize(28);
```



```
style.getFont().setStrikethrough(true);
style.getFont().setSubscript(true);
style.getFont().setSuperscript(false);
style.getFont().setUnderline(UnderlineType.Double);

// Protection
style.setFormulaHidden(true);
style.setLocked(false);

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

// Alignment
style.setHorizontalAlignment(HorizontalAlignment.Right);
style.setVerticalAlignment(VerticalAlignment.Bottom);
style.setWrapText(true);
style.setIndentLevel(5);
style.setOrientation(45);

// Fill
style.getInterior().setColorIndex(5);
style.getInterior().setPattern(Pattern.Down);
style.getInterior().setPatternColor(Color.FromArgb(0, 0, 255));
style.setIncludeAlignment(false);
style.setIncludeBorder(true);
style.setIncludeFont(false);
style.setIncludeNumber(true);
style.setIncludePatterns(false);
style.setIncludeProtection(true);
```

You can get or set named style in a worksheet using the **setStyle** ('**setStyle Method**' in the on-line documentation) method of the **IRange** ('**IRange Interface**' in the on-line documentation) interface.

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

Java

```
// Set range's style to custom name style.
worksheet.getRange("A1").setStyle(worksheet.getWorkbook().getStyles().get("testStyle"));
worksheet.getRange("A1").setValue(123);
```

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

With GcExcel Java, it is not necessary to create a custom named style right from the scratch. Instead, you can tweak an existing style (via getting the existing style from the Styles collection) as per your custom requirements and save the new style as another workbook style that can be used as and when required.

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

style in the future.

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

## Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Fetch existing Style "Good" and set to Range A1's Style
worksheet.getRange("A1").setStyle(workbook.getStyles().get("Good"));

// Setting Cell Text
worksheet.getRange("A1").setValue("Good");

// Create and modify a style based on current existing style "Good" and name it as "MyGood"
IStyle myGood = workbook.getStyles().add("MyGood", workbook.getStyles().get("Good"));
myGood.getFont().setBold(true);
myGood.getFont().setItalic(true);

// Set new style "MyGood" to Range B1's Style
worksheet.getRange("B1").setStyle(workbook.getStyles().get("MyGood"));

// Setting Cell Text
worksheet.getRange("B1").setValue("MyGood");

// Saving workbook
workbook.save("2-CreateModifyStyleBasedOnAStyle.xlsx");
```



**Note:** The following limitations must be kept in mind while exporting Excel files with vertical text to PDF -

- The orientation can only be set to 0, 90, -90 and 255. Other values will be treated as 0 while rendering the PDF file.
- If the font name starts with "@" and the orientation is 255, GcExcel will ignore the "@".

## Theme

GcExcel Java enables users to choose from a set of built-in themes in order to enhance the overall appearance of the workbook. Also, it provides users with the ability to add and apply custom themes for configuring a workbook as per their choice.

When a theme is modified, it impacts all the areas including the theme fonts, theme colors, range, chart titles etc. For example : if you apply a built-in or a custom theme to your workbook, it is possible that the color of the range as well as the font will also be modified based on the customized theme. The default theme of a workbook is the standard Office theme. The current theme of a workbook is represented by the **ITheme ('ITheme Interface' in the on-line documentation)** interface.

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

Refer to the following tasks to apply theme in your workbook:

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

## Apply built-in theme to the workbook

To maintain consistency in the appearance across all the worksheets in the workbook, GcExcel Java enables users to add and apply theme from a set of built-in themes.

In order to apply a built-in theme to your workbook, refer to the following example code.

Java

```
// Change workbook's theme to Berlin
workbook.setTheme(Themes.GetBerlin());
```

## Add a custom theme and set to workbook

The **Theme** class can be used to add a custom theme to a workbook. After adding the custom theme, users can apply it to the workbook.

In order to add a custom theme and apply it to the workbook, refer to the following example code.

Java

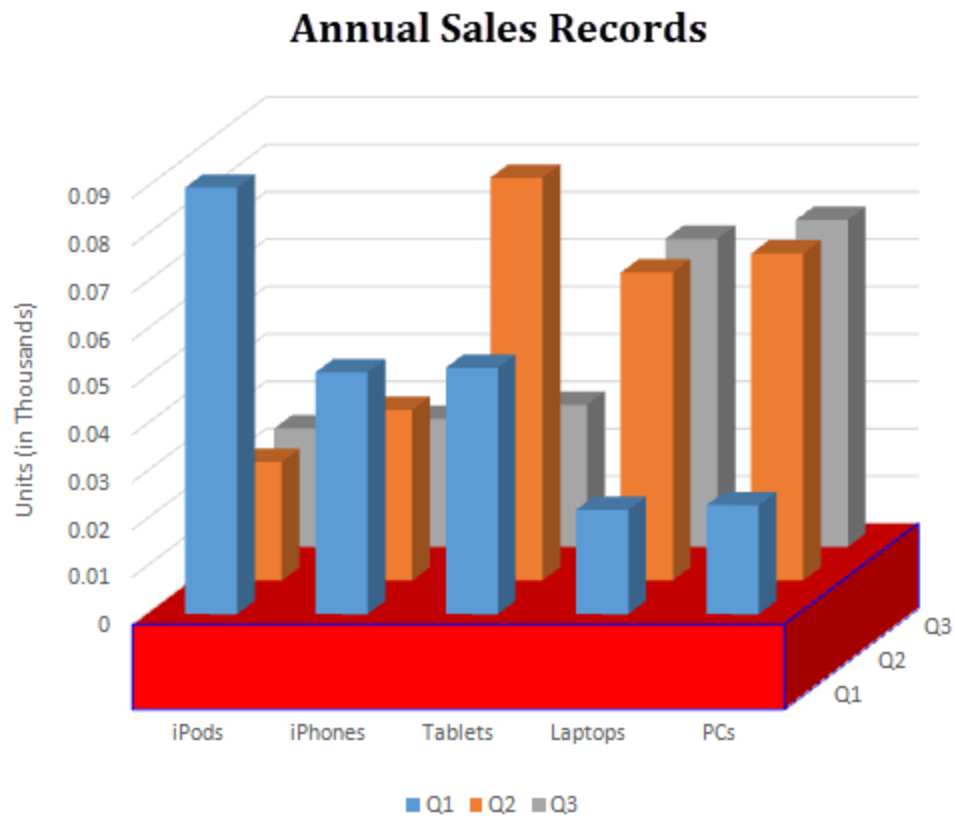
```
// Add Custom Theme
theme.getThemeColorScheme().get(ThemeColor.Light1).setRGB(Color.GetAntiqueWhite());
theme.getThemeColorScheme().get(ThemeColor.Accent1).setRGB(Color.GetAliceBlue());
theme.getThemeFontScheme().getMajor().get(FontLanguageIndex.Latin).setName("Buxton Sketch");
theme.getThemeFontScheme().getMinor().get(FontLanguageIndex.Latin).setName("Segoe UI");

// Apply Custom Theme
workbook.setTheme(theme);
```

## Chart

GcExcel Java allows users to communicate, display and manipulate useful information in charts.

This feature not only helps users in visualizing trends swiftly and effectively but also helps business analysts and managers to compare numbers across bulk data, analyse essential patterns and visualize significant trends quickly and effectively.



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](#)
- [Chart Types](#)
- [Chart Sheet](#)

## Create and Delete Chart

GcExcel Java enables users to add charts in spreadsheets for improved data analysis and enhanced data visualization.

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

### Create Chart

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

To create a chart, refer to the following example code.

```
Java
```

```
// Add Chart
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 300, 10, 300,
300);
worksheet.getRange("A1:D6").setValue(
    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 Chart
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns,
true, true);
```

### Delete Chart

You can delete an existing chart by using the **delete** ('delete Method' in the on-line documentation) method of the **IShape** ('IShape Interface' in the on-line documentation) interface.

To delete a chart from your worksheet, refer to the following example code.

Java

```
// Delete Chart
shape.getChart().delete();
```

## Configure Chart

In GcExcel Java, you can configure an existing chart in a spreadsheet via setting its display as per your preferences.

While configuring a chart, you can refer to the following topics:

- [Chart Title](#)
- [Chart Area](#)
- [Plot Area](#)

### Chart Title

In GcExcel Java, you can use the methods of the **IShape** ('IShape Interface' in the on-line documentation) interface in order to configure the chart title of your choice.

You can work with Chart title in the following ways:

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

#### Set formula for chart title

To set formula for chart title, refer to the following example code.

Java

```
// Set formula for chart title.
shape.getChart().setHasTitle(true);
shape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs().add("ChartSubtitle");
shape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs().add("ChartTitle", 0);
```

### Set format for chart title and font style

To set format for chart title and font style, refer to the following example code.

Java

```
// Set chart titale's format and font style.
shape.getChart().setHasTitle(true);
//shape.getChart().getChartTitle().setText("MyChartTitle");
shape.getChart().getChartTitle().getFormat().getFill().getColor().setRGB(Color.GetDarkOrange());
shape.getChart().getChartTitle().getFormat().getLine().getColor().setRGB(Color.GetCornflowerBlue());
```

### Set text angle for chart title

You can also configure the text angle for chart title by using the **setOrientation** method of **IChartTitle** interface. The text angle can also be exported or imported to JSON.

Refer to the following example code to set text angle for chart title.

Java

```
//create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

//add chart
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(new Object[] []
{
{null, "S1", "S2", "S3"},
{"Item1", 10, 25, 25},
{"Item2", -20, 36, 27},
{"Item3", 62, 70, -30},
{"Item4", 22, 65, 65},
{"Item5", 23, 50, 50}
});
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

//add chart title
shape.getChart().setHasTitle(true);

shape.getChart().getChartTitle().setText("MyChartTitle");

//config chart title angle
shape.getChart().getChartTitle().setOrientation(30);

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

## Chart Area

In GcExcel Java, you can use the methods of the **IChartArea** (**'IChartArea Interface' in the on-line documentation**) interface in order to set up the chart area as per your preferences.

You can work with Chart Area in the following ways:

- **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 **getFont** (**'getFont Method' in the on-line documentation**) method, **getFormat** (**'getFormat Method' in the on-line documentation**) method and **setRoundedCorners** (**'setRoundedCorners Method' in the on-line documentation**) method of the **IChartArea** (**'IChartArea Interface' in the on-line documentation**) interface.

To configure chart area style in your worksheet, refer to the following example code.

Java

```
// Configure chart area style
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(
    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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns,
true, true);
IChartArea chartarea = shape.getChart().getChartArea();

// Font
chartarea.getFont().getColor().setRGB(Color.GetMediumSeaGreen());
chartarea.getFont().setName("Times New Roman");
chartarea.getFont().setSize(12);

// Rounded corners.
chartarea.setRoundedCorners(true);
```

### Set chart area format

To set chart area format in your worksheet, refer to the following example code.

Java

```
chartarea.getFormat().getFill().getColor().setRGB(Color.GetLightGray());
chartarea.getFormat().getLine().getColor().setRGB(Color.GetMediumSeaGreen());
```

```
chartarea.getFormat().getLine().setWeight(1.5);
```

## Plot Area

In GcExcel Java, you can use the methods of the **IPlotArea** (**'IPlotArea Interface' in the on-line documentation**) interface in order to set up the plot area in a chart as per your preferences.

### Configure plot area format

You can configure the plot area format via modifying its fill color, line color and other essential attributes using the **getPlotArea** method of the **IChart** interface.

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

Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(new Object[][] { { null, "S1", "S2", "S3" }, {
    "Item1", 10, 25, 25 },
    { "Item2", -51, 36, 27 }, { "Item3", 52, 50, -30 }, { "Item4", 22, 65, 30 }, {
    "Item5", 23, 40, 69 } });
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns,
true, true);

IPlotArea plotarea = shape.getChart().getPlotArea();

// Format.
plotarea.getFormat().getFill().getColor().setRGB(Color.GetLightGray());
plotarea.getFormat().getLine().getColor().setRGB(Color.GetGray());
```

## Customize Chart Objects

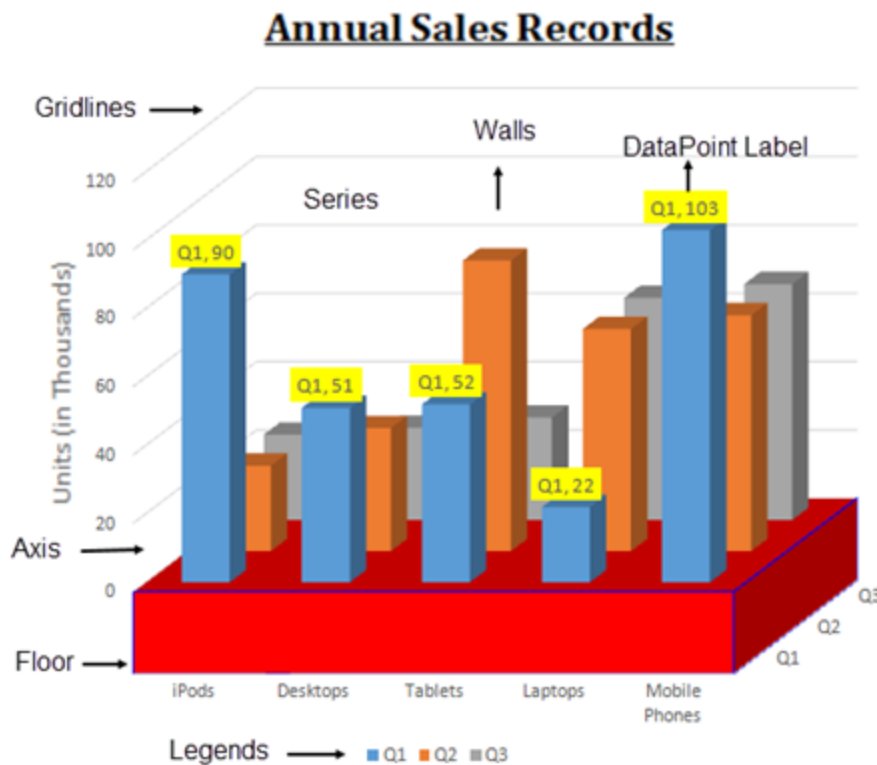
The chart feature in GcExcel Java enables users to create different types of charts including both 2-D and 3-D views.

You can completely customize all the Chart objects in GcExcel Java. The following list of charting objects can be modified while creating charts:

1. [Series](#)
2. [Walls](#)
3. [Axis and other Lines](#)
4. [Floor](#)
5. [Data Label](#)
6. [Legends](#)

Shared below is a diagram that displays a sample chart depicting the annual sales records of different electronic gadgets per quarter along with the chart objects that users can customize in a worksheet.





## Series

Series refers to a set of data points, or simply a list of values plotted in a chart.

While working with spreadsheets, you can plot one or more data series in a chart. Each series is represented with a legend item and provides access to the chart control's collection of series objects.

In GcExcel Java, the methods of the **ISeries** (**'ISeries Interface' in the on-line documentation**) interface and the **ISeriesCollection** (**'ISeriesCollection Interface' in the on-line documentation**) interface enables users to insert individual series, access it, delete it and perform other useful operations as per the requirements.

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

Java

```
// Adding Charts
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(new Object[][]
{
    {null, "S1", "S2", "S3"},
    {"Item1", 10, 25, 25},
    {"Item2", -51, -36, 27},
    {"Item3", 52, -85, -30},
    {"Item4", 22, 65, 65},
    {"Item5", 23, 69, 69}
});
```

```
// Detects three series, B2:B6, C2:C6, D2:D6.
// Does not detect out series labels and category labels, auto generated.
shape.getChart().getSeriesCollection().add(worksheet.getRange("B2:D6"));

IShape shape2 = worksheet.getShapes().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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"));

IShape shape3 = worksheet.getShapes().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.getChart().getSeriesCollection().add(worksheet.getRange("B2:D6"), RowCol.Rows);

IShape shape4 = worksheet.getShapes().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.getChart().getSeriesCollection().add(worksheet.getRange("B2:D6"),
RowCol.Columns);

IShape shape5 = worksheet.getShapes().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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"),
RowCol.Columns);

IShape shape6 = worksheet.getShapes().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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns,
true, true);

workbook.getWorksheets().add();
IWorksheet worksheet1 = workbook.getWorksheets().get(1);
worksheet1.getRange("A1:D6").setValue(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.getShapes().addChart(ChartType.ColumnClustered, 200, 50, 300,
300);
ISeries series1 = shape7.getChart().getSeriesCollection().newSeries();
ISeries series2 = shape7.getChart().getSeriesCollection().newSeries();
ISeries series3 = shape7.getChart().getSeriesCollection().newSeries();
series1.setFormula("=SERIES(Sheet1!$B$1,Sheet1!$A$2:$A$6,Sheet1!$B$2:$B$6,1)");
series2.setFormula("=SERIES(Sheet1!$C$1,Sheet1!$A$2:$A$6,Sheet1!$C$2:$C$6,2)");
series3.setFormula("=SERIES(Sheet1!$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.getShapes().addChart(ChartType.ColumnClustered, 200, 450,
300, 300);
shape8.getChart().getSeriesCollection().add(worksheet1.getRange("A1:D6"),
RowCol.Columns, true, true);
worksheet1.getRange("A12:D14").setValue(new Object[][]
{
    {"Item6", 50, 20, -30},
    {"Item7", 60, 50, 50},
    {"Item8", 35, 80, 60}
});
shape8.getChart().getSeriesCollection().extend(worksheet1.getRange("A12:D14"),
RowCol.Columns, true);

workbook.getWorksheets().add();
IWorksheet worksheet2 = workbook.getWorksheets().get(1);
worksheet2.getRange("A1:D6").setValue(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.getShapes().addChart(ChartType.Line, 200, 50, 300, 300);
shape9.getChart().getSeriesCollection().add(worksheet2.getRange("A1:D6"),
RowCol.Columns, true, true);
ISeries series4 = shape9.getChart().getSeriesCollection().get(0);
ISeries series5 = shape9.getChart().getSeriesCollection().get(1);
series4.setAxisGroup(AxisGroup.Secondary);
series5.setChartType(ChartType.ColumnClustered);

// Set 3D column chart's bar shape.
IShape shape10 = worksheet2.getShapes().addChart(ChartType.Column3D, 200, 450, 300,
300);
shape10.getChart().getSeriesCollection().add(worksheet2.getRange("A1:D6"),
RowCol.Columns, true, true);
ISeries series6 = shape10.getChart().getSeriesCollection().get(0);
ISeries series7 = shape10.getChart().getSeriesCollection().get(1);
ISeries series8 = shape10.getChart().getSeriesCollection().get(2);;
series6.setBarShape(BarShape.ConeToMax);
series7.setBarShape(BarShape.Cylinder);
series8.setBarShape(BarShape.PyramidToPoint);

// Set negative point's fill color.
IShape shape11 = worksheet2.getShapes().addChart(ChartType.Column3D, 200, 800, 300,
300);
shape11.getChart().getSeriesCollection().add(worksheet2.getRange("A1:D6"),
RowCol.Columns, true, true);
ISeries series9 = shape11.getChart().getSeriesCollection().get(0);
series9.setInvertIfNegative(true);
series9.getInvertColor().setRGB(Color.GetGreen());

// Set series' plot order as 6
IShape shape12 = worksheet2.getShapes().addChart(ChartType.ColumnClustered, 200, 1100,
300, 300);
worksheet.getRange("A1:E6").setValue(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.getChart().getSeriesCollection().add(worksheet2.getRange("A1:E6"),
RowCol.Columns, true, true);

ISeries series10 = shape12.getChart().getSeriesCollection().get(0);
ISeries series11 = shape12.getChart().getSeriesCollection().get(1);;
ISeries series12 = shape12.getChart().getSeriesCollection().get(2);;
ISeries series13 = shape12.getChart().getSeriesCollection().get(3);;
```

```
// series11 and series13 plot on secondary axis.
series11.setAxisGroup(AxisGroup.Secondary);
series13.setAxisGroup(AxisGroup.Secondary);

// series10 and series12 are in one chart group.
series12.setPlotOrder(1);
series10.setPlotOrder(2);

// series4 and series2 are in one chart group.
series13.setPlotOrder(1);
series11.setPlotOrder(2);

// Configure series' marker.
IShape shape13 = worksheet2.getShapes().addChart(ChartType.Line, 200, 1450, 300, 300);
shape13.getChart().getSeriesCollection().add(worksheet2.getRange("A1:D6"),
RowCol.Columns, true, true);

ISeries series14 = shape13.getChart().getSeriesCollection().get(0);

series14.setMarkerStyle(MarkerStyle.Diamond);
series14.setMarkerSize(10);
series1.getMarkerFormat().getFill().getColor().setRGB(Color.GetRed());
series1.getMarkerFormat().getLine().setStyle(Linestyle.ThickThin);
series1.getMarkerFormat().getLine().getColor().setRGB(Color.GetGreen());
series1.getMarkerFormat().getLine().setWeight(3);
```

## Configure Chart Series

GcExcel Java allows users to configure chart series in the following ways:

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

### DataPoint

The Points collection in GcExcel Java is used to represent all the points in a specific series and the indexer notation of the **IPoints ('IPoints Interface' in the on-line documentation)** interface to get a specific point in the series. Also, you can use the **getDataLabel ('getDataLabel Method' in the on-line documentation)** method of the **IPoint ('IPoint Interface' in the on-line documentation)** interface in order to get data label of a specific point.

#### Set the format of DataPoint

In order to set data point format for the chart added in your worksheet, refer to the following example code.

Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(
    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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);
ISeries series1 = shape.getChart().getSeriesCollection().get(0);

series1.getPoints().get(2).getFormat().getFill().getColor().setRGB(Color.FromArgb(0, 176, 240));
series1.getPoints().get(2).getFormat().getLine().getColor().setRGB(Color.GetBlue());

```

### Configure secondary section for pie of a pie chart

You can use the **setSecondaryPlot** ('**setSecondaryPlot Method**' in the **on-line documentation**) method 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.

In order to configure secondary section for pie of a pie chart, refer to the following example code.

#### Java

```

IShape shape = worksheet.getShapes().addChart(ChartType.PieOfPie, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(
    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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);

shape.getChart().getChartGroups().get(0).setSplitType(ChartSplitType.SplitByCustomSplit);
series1.getPoints().get(0).setSecondaryPlot(true);
series1.getPoints().get(1).setSecondaryPlot(false);
series1.getPoints().get(2).setSecondaryPlot(true);
series1.getPoints().get(3).setSecondaryPlot(false);
series1.getPoints().get(4).setSecondaryPlot(true);

```

### DataLabel

You can use the **DataLabels** collection to represent the collection of all the data labels for a specific series.

The **getFormat** ('**getFormat Method**' in the **on-line documentation**) method of the **IDataLabel** ('**IDataLabel Interface**' in the **on-line documentation**) interface can be used to set font style, fill, line and 3-D formatting for all the data labels of a specific series. Users can also configure the layout of the data labels using other methods of the **IDataLabel** interface.

### Set all data labels and specific data label format for series

In order to set all data labels and specific data label format of a series, refer to the following example code.

#### Java

```

IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:B5").setValue(
    new Object[][] { { null, "S1" }, { "Item1", -20 }, { "Item2", 30 }, { "Item3", 50 }, { "Item3", 40 } }));
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:B5"), RowCol.Columns, true, true);

ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);

// set series1's all data label's format.
series1.getDataLabels().getFormat().getFill().getColor().setRGB(Color.GetPink());
series1.getDataLabels().getFormat().getLine().getColor().setRGB(Color.GetGreen());
series1.getDataLabels().getFormat().getLine().setWeight(1);

// set series1's specific data label's format.
series1.getDataLabels().get(2).getFormat().getFill().getColor().setRGB(Color.GetLightGreen());
series1.getPoints().get(2).getDataLabel().getFormat().getLine().getColor().setRGB(Color.GetGray());
series1.getPoints().get(2).getDataLabel().getFormat().getLine().setWeight(2);

```

**Customize data label text**

In order to set the text of the data label as per your choice, refer to the following example code.

Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:B5").setValue(new Object[][] { { null, "S1", "S2" }, { "Item1", -20 }, { "Item2", 30 },
    { "Item3", 50 }, { "Item3", 40 } });
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:B5"), RowCol.Columns, true, true);
ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);

// customize data label's text.
series1.getDataLabels().setShowCategoryName(true);
series1.getDataLabels().setShowSeriesName(true);
series1.getDataLabels().setShowLegendKey(true);
```

**Trendline**

The Trendlines collection in GcExcel Java is used to represent a collection of trend lines for a specific series. You can use the **get** ('get Method' in the **on-line documentation**) method of the **ITrendlines** ('Trendlines Interface' in the **on-line documentation**) interface to create a new trendline for a specific series. Also, the indexer notation of the ITrendlines interface can be used to get a specific trend line.

**Add trendline for series and configure its style**

In order to add trendline for series and configure its style, refer to the following example code.

Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.getTrendlines().add();
series1.getTrendlines().get(0).setType(TrendlineType.Linear);
series1.getTrendlines().get(0).setForward(5);
series1.getTrendlines().get(0).setBackward(0.5);
series1.getTrendlines().get(0).setIntercept(2.5);
series1.getTrendlines().get(0).setDisplayEquation(true);
series1.getTrendlines().get(0).setDisplayRSquared(true);
```

**Add two trendlines for one series**

In order to add two trendlines for one series, refer to the following example code.

Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.getTrendlines().add();
series1.getTrendlines().get(0).setType(TrendlineType.Linear);
series1.getTrendlines().get(0).setForward(5);
series1.getTrendlines().get(0).setBackward(0.5);
series1.getTrendlines().get(0).setIntercept(2.5);
```

```
series1.getTrendlines().get(0).setDisplayEquation(true);
series1.getTrendlines().get(0).setDisplayRSquared(true);

series1.getTrendlines().add();
series1.getTrendlines().get(1).setType(TrendlineType.Polynomial);
series1.getTrendlines().get(1).setOrder(3);
```

### Set trendline's name

You can also set the trendline's name in GcExcel using the **setName** method of **ITrendline** interface. The trendline's name can also be exported to a PDF document.

Refer to the following example code to set trendline's name in GcExcel.

#### Java

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Add a chart
IShape columnChart = worksheet.getShapes().addChart(ChartType.ColumnClustered, 300, 10, 300, 300);

worksheet.getRange("A1:D6").setValue(new Object[][] {
{
{null, "S1", "S2", "S3"},
{"Item1", 10, 25, 25},
{"Item2", -51, -36, 27},
{"Item3", 52, -85, -30},
{"Item4", 22, 65, 65},
{"Item5", 23, 69, 69}
}});

// Add series
columnChart.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

// Get first series
ISeries series1 = columnChart.getChart().getSeriesCollection().get(0);

// Add a trend line
ITrendline trendline = series1.getTrendlines().add();

// Set trend line's name.
trendline.setName("Theoretical data");

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

### Chart Group

A Chart Group possesses 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

In order to set different colors for a column chart (that contains only one series), refer to the following example code.

#### Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
```



```

shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

shape.getChart().getSeriesCollection().get(2).delete();
shape.getChart().getSeriesCollection().get(1).delete();

// Chart's series count is 1.
// int count = shape.getChart().getSeriesCollection().getCount();
shape.getChart().getSeriesCollection().getCount();

// set vary colors for column chart which only has one series.
shape.getChart().getColumnGroups().get(0).setVaryByCategories(true);

```

### Set split setting and gap width for pie of a pie chart

In order to set split setting and gap width for pie of a pie chart, refer to the following example code.

#### Java

```

IShape shape = worksheet.getShapes().addChart(ChartType.PieOfPie, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);

shape.getChart().getPieGroups().get(0).setSplitType(ChartSplitType.SplitByValue);
shape.getChart().getPieGroups().get(0).setSplitValue(20);
shape.getChart().getPieGroups().get(0).setGapWidth(350);

```

### Set gap width of column chart and overlap

In order to set the gap width of the column chart along with overlap, refer to the following example code.

#### Java

```

IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);

shape.getChart().getColumnGroups().get(0).setGapWidth(120);
shape.getChart().getColumnGroups().get(0).setOverlap(-20);

```

### Configure the layout of the bubble chart

In order to configure the layout of the bubble chart as per your preferences, refer to the following example code.

#### Java

```

IShape shape = worksheet.getShapes().addChart(ChartType.Bubble, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

```

```
ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);

shape.getChart().getXYGroups().get(0).setBubbleScale(150);
shape.getChart().getXYGroups().get(0).setSizeRepresents(SizeRepresents.SizeIsArea);
shape.getChart().getXYGroups().get(0).setShowNegativeBubbles(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.

```
Java

IShape shape = worksheet.getShapes().addChart(ChartType.Doughnut, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);

shape.getChart().getDoughnutGroups().get(0).setFirstSliceAngle(50);
shape.getChart().getDoughnutGroups().get(0).setDoughnutHoleSize(20);
```

### Dropline, HiLoline and SeriesLine

You can use the methods of the **IChartGroup** ('**IChartGroup Interface** in the on-line documentation) interface to configure Dropline, HiLoline and Series lines in a chart.

### Configure the drop lines of the line chart

In order to configure the drop lines of the line chart as per your preferences, refer to the following example code.

```
Java

IShape shape = worksheet.getShapes().addChart(ChartType.Line, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

shape.getChart().getLineGroups().get(0).setHasDropLines(true);
shape.getChart().getLineGroups().get(0).getDropLines().getFormat().getLine().getColor().setRGB(Color.GetRed());
```

### Configure the high-low lines of the line chart

In order to configure the high-low lines of the line chart as per your preferences, refer to the following example code.

```
Java

IShape shape = worksheet.getShapes().addChart(ChartType.Line, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);

shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

shape.getChart().getLineGroups().get(0).setHasHiLoLines(true);
shape.getChart().getLineGroups().get(0).getHiLoLines().getFormat().getLine().getColor().setRGB(Color.GetRed());
```

### Configure the series lines for column chart

In order to configure the column chart's series lines as per your preferences, refer to the following example code.

#### Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnStacked, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

shape.getChart().getColumnGroups().get(0).setHasSeriesLines(true);
shape.getChart().getColumnGroups().get(0).getSeriesLines().getFormat().getLine().getColor()
    .setRGB(Color.GetRed());
```

#### Configure the connector lines for pie of a pie chart

In order to configure the connector lines for pie of a pie chart as per your preferences, refer to the following example code.

#### Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.PieOfPie, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

shape.getChart().getPieGroups().get(0).setHasSeriesLines(true);
shape.getChart().getPieGroups().get(0).getSeriesLines().getFormat().getLine().getColor().setRGB(Color.GetRed());
```

#### Up-Down Bars

You can use the methods of the IChartGroup interface 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

In order to configure the up-down bars for the line chart as per your preferences, refer to the following example code.

#### Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.Line, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

shape.getChart().getLineGroups().get(0).setHasUpDownBars(true);
shape.getChart().getLineGroups().get(0).getUpBars().getFormat().getFill().getColor().setRGB(Color.GetGreen());
shape.getChart().getLineGroups().get(0).getDownBars().getFormat().getFill().getColor().setRGB(Color.GetRed());
```

## Error Bars

Error bars are used in charts to indicate the error or uncertainty of data. They act as an extremely useful tool for scientists, statisticians, and research analysts to showcase data variability and measurement accuracy.

GcExcel allows you to configure error bars in charts using **IErrorBar** interface. The interface represents error bars in a chart series and provides properties to configure various types, end styles and value types of error bars. The error bars can also be exported or imported to JSON or a PDF document.




#### Supported Chart Types

The following chart types are supported while adding error bars in charts:

- Area Charts
- Bar Charts
- Column charts

- Line Charts
- xyScatter Charts

### Error Bar Types

Type	Snapshot	Description
Plus		<p>Error bar depicts only the positive values.</p> <pre>Java series.getYErrorBar.setType(ErrorBarInclude.Plus);</pre>
Minus		<p>Error bar depicts only the negative values.</p> <pre>Java series.getYErrorBar.setType(ErrorBarInclude.Minus);</pre>
Both		<p>Error bar depicts positive and negative values at the same time</p> <pre>Java series.getYErrorBar.setType(ErrorBarInclude.Both);</pre>



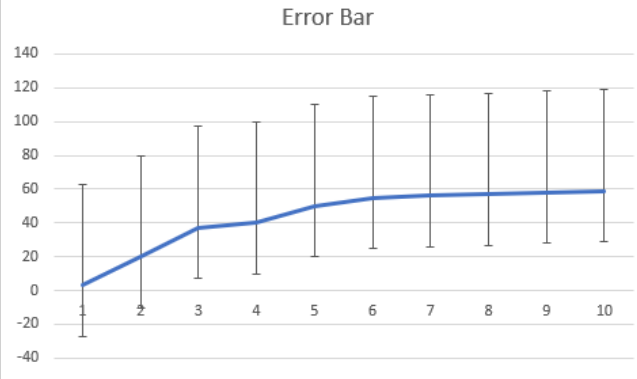
### Error Bar End Styles

Type	Snapshot	Description
------	----------	-------------

Cap		Error bar displays caps at the end of error bar lines.
No Cap		Error bar does not display caps at the end of error bar lines.

## Error Bar Value Types

Type	Snapshot	Description
Fixed Value		Error bar represents the error as an absolute value.
Percentage		Error bar represents the error as a percentage of data value in the same direction axis.

Standard Deviation		<p>Error bar represents the error as a calculating value which depends on the set deviation and chart data values.</p> <pre>Java series.getYErrorBar.setValueType(ErrorBarType.StDev);</pre>
Standard Error		<p>Error bar represents the error as a calculating value which only depends on the chart data values.</p> <pre>Java series.getYErrorBar.setValueType(ErrorBarType.StError);</pre>
Custom		<p>Error bar represents the error values that are set with positive and negative values respectively by formulas or fixed values.</p> <pre>Java series.getYErrorBar.setValueType(ErrorBarType.Custom);</pre>

**Note:** In Custom value type, the array and reference formula string for plus or minus is supported. The final count of error bar values is evaluated by custom formula which has different behavior.

If count = 1: all error bars are the same as the only one value.

If count < number of data points: the rest error bar value will be zero.

If count > number of data points: the beyond values will do nothing.

## Using Code

Refer to the following example code to add error bars using various properties in the chart.

```
Java
// create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Prepare data for chart
worksheet.getRange("A1:D4")
.setValue(new Object[][] { { null, "Q1", "Q2", "Q3" }, { "Mobile Phones", 1330, 2345, 3493 },
```

```

{ "Laptops", 2032, 3632, 2197 }, { "Tablets", 6233, 3270, 2030 } }));
worksheet.getRange("A:D").getColumns().autoFit();
// Add Column Chart
IShape columnChartshape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);

// Adding series to SeriesCollection
columnChartshape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D4"), RowCol.Columns, true, true);

// Get first series
ISeries series1 = columnChartshape.getChart().getSeriesCollection().get(0);

// Config first series' properties
series1.setHasErrorBars(true);
series1.getYErrorBar().setType(ErrorBarInclude.Both);
series1.getYErrorBar().setValueType(ErrorBarType.Custom);
series1.getYErrorBar().setEndStyle(EndStyleCap.Cap);
series1.getYErrorBar().setPlus("={200,400,600}");
series1.getYErrorBar().setMinus("={600,400,200}");

// Get second series
ISeries series2 = columnChartshape.getChart().getSeriesCollection().get(1);

// Config second series' properties
series2.setHasErrorBars(true);
series2.getYErrorBar().setType(ErrorBarInclude.Plus);
series2.getYErrorBar().setValueType(ErrorBarType.FixedValue);
series2.getYErrorBar().setEndStyle(EndStyleCap.Cap);
series2.getYErrorBar().setAmount(1000);
series2.getYErrorBar().getFormat().getLine().getColor().setRGB(Color.GetRed());
series2.getYErrorBar().getFormat().getLine().setWeight(2);

// Get last series
ISeries series3 = columnChartshape.getChart().getSeriesCollection().get(2);

// Config last series' properties
series3.setHasErrorBars(true);
series3.getYErrorBar().setType(ErrorBarInclude.Both);
series3.getYErrorBar().setValueType(ErrorBarType.StError);
series3.getYErrorBar().setEndStyle(EndStyleCap.NoCap);

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

```

### Important Points

- Only series in scatter chart groups can have x and y error bars. Otherwise, an exception would be thrown.
- `ISeries.setHasErrorBars` must be set as "true" to display error bar.
- `IErrorBar.setAmount` only takes effect when `IErrorBar.setValueType` is `FixedValue` or `Percentage` or `Standard Deviation`.
- `IErrorBar.setPlus` or `IErrorBar.setMinus` only takes effect when `IErrorBar.setValueType` is `Custom`.
- `IErrorBar.setPlus` or `IErrorBar.setMinus` accepts a formula string like `"=Sheet1!$B$2:$D$2"` or `"={1,2,3}"`.

### Limitation

There can be some difference between the exported PDF and Excel containing error bars. It is caused due to different ways of calculating error bar value between GcExcel and Excel.

## Walls

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

Using GcExcel Java, you can configure a chart as per your custom preferences via defining the thickness, fill color, line color and format of the back wall as well as the side wall, using the methods of the **IWall ('IWall Interface' in the on-line documentation)** interface and the **IShape ('IShape Interface' in the on-line documentation)** interface.

In order to configure the walls of the chart inserted in a worksheet, refer to the following example code.

```

Java

// Config back wall and side wall's format together.

```

```
IShape shape1 = worksheet.getShapes().addChart(ChartType.Column3D, 250, 20, 350, 250);
shape1.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true,
true);
shape1.getChart().getSideWall().setThickness(5);
shape1.getChart().getSideWall().getFormat().getFill().getColor().setRGB(Color.GetLightGreen());
shape1.getChart().getSideWall().getFormat().getLine().getColor().setRGB(Color.GetLightBlue());

// Config back wall's format individually.
IShape shape2 = worksheet.getShapes().addChart(ChartType.Column3D, 250, 20, 350, 250);
shape2.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true,
true);
shape2.getChart().getBackWall().setThickness(5);
shape2.getChart().getBackWall().getFormat().getFill().getColor().setRGB(Color.GetLightGreen());
shape2.getChart().getBackWall().getFormat().getLine().getColor().setRGB(Color.GetLightBlue());
```

## Axis and Other Lines

Axis refers to a charting element that displays the scale for a single dimension of a plot area.

Using GcExcel Java, you can configure a title, major tick mark, minor tick mark, tick mark labels, major gridlines and minor gridlines for the Axis and other lines in a chart.

Generally, a two-dimensional chart comprises two types of axes - category axis and value axis. The category axis is also known as horizontal axis (x-axis) and can be 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. A 3-D chart can have the following three types of axes:

1. Category axis - Displays categories 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, i.e. 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, i.e. 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.

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

To configure axis in your chart, refer to the following example code.

Java

```
// Use IAxis.CategoryType to set category axis's scale type
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360,
230);

worksheet.getRange("A1:D6").setValue(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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns,
true, true);
worksheet.getRange("A2:A6").setNumberFormat("m/d/yyyy");
IAxis category_axis = shape.getChart().getAxes().item(AxisType.Category);

// Category axis's category type is automatic scale
category_axis.setCategoryType(CategoryType AutomaticScale);

// Category axis's actual category type is time scale
CategoryType actualcategorytype = category_axis.getActualCategoryType();

IWorksheet worksheet1 = workbook.getWorksheets().add();
worksheet1.getRange("A1:D6").setValue(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 }
});
IShape shape2 = worksheet1.getShapes().addChart(ChartType.Line, 250, 20, 360, 230);
shape2.getChart().getSeriesCollection().add(worksheet1.getRange("A1:D6"),
RowCol.Columns, true, true);
IAxis category_axis1 = shape2.getChart().getAxes().item(AxisType.Category);

// Set category axis's format.
category_axis1.getFormat().getFill().getColor().setObjectThemeColor(ThemeColor.Accent1);
category_axis1.getFormat().getLine().getColor().setRGB(Color.GetLightSkyBlue());
category_axis1.getFormat().getLine().setWeight(3);
category_axis1.getFormat().getLine().setStyle(LineStyle.Single);

IAxis value_axis = shape2.getChart().getAxes().item(AxisType.Value);

// Set value axis's format.
value_axis.getFormat().getLine().getColor().setRGB(Color.FromArgb(91, 155, 213));
value_axis.getFormat().getLine().setWeight(2);
value_axis.getFormat().getLine().setStyle(LineStyle.Single);

// Configure time scale category axis's units.
worksheet1.getRange("A8:A12").setNumberFormat("m/d/yyyy");
worksheet1.getRange("A7:D12").setValue(new Object[][]
```

```
{
    {null, "S1", "S2", "S3"},
    {new GregorianCalendar(2015, 9, 7), 10, 25, 25},
    {new GregorianCalendar(2015, 9, 24), 51, 36, 27},
    {new GregorianCalendar(2015, 10, 8), 52, 85, 30},
    {new GregorianCalendar(2015, 10, 25), 22, 65, 65},
    {new GregorianCalendar(2015, 11, 10), 23, 69, 69}
});

IShape shape3 = worksheet1.getShapes().addChart(ChartType.ColumnClustered, 200, 450,
300, 300);
shape3.getChart().getSeriesCollection().add(worksheet1.getRange("A7:D12"),
RowCol.Columns, true, true);
IAxis category_axis2 = shape3.getChart().getAxes().item(AxisType.Category);

category_axis2.setMaximumScale(DateInfo.ToOADate(new GregorianCalendar(2019, 9, 1)));
category_axis2.setMinimumScale(DateInfo.ToOADate(new GregorianCalendar(2015, 9, 1)));

category_axis2.setBaseUnit(TimeUnit.Years);
category_axis2.setMajorUnitScale(TimeUnit.Months);
category_axis2.setMajorUnit(4);
category_axis2.setMinorUnitScale(TimeUnit.Days);
category_axis2.setMinorUnit(60);

// Configure value axis's units.
IShape shape4 = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360,
230);
worksheet.getRange("A1:D6").setValue(new Object[][]
{
    { null, "S1", "S2", "S3" },
    { "Item1", 10, 25, 25 },
    { "Item2", -51, 36, 27 },
    { "Item3", 52, 90, -30 },
    { "Item4", 22, 65, 50 },
    { "Item5", 23, 55, 69 }
});
shape4.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns,
true, true);
IAxis value_axis1 = shape4.getChart().getAxes().item(AxisType.Value);
value_axis1.setMaximumScale(100);
value_axis1.setMinimumScale(-100);
value_axis1.setMajorUnit(30);
value_axis1.setMinorUnit(6);

// Set axis crosses at.
IAxis value_axis_cross = shape.getChart().getAxes().item(AxisType.Value);
value_axis_cross.setCrosses(AxisCrosses.Maximum);
```

```
// Set axis's scale type.
IAxis value_axis_scale = shape.getChart().getAxes().item(AxisType.Value);
value_axis_scale.setScaleType(ScaleType.Logarithmic);
value_axis_scale.setLogBase(5);

// Set axis's tick mark.
IAxis category_axis_tick = shape.getChart().getAxes().item(AxisType.Category);
category_axis_tick.getFormat().getLine().getColor().setRGB(Color.GetGreen());
category_axis_tick.setMajorTickMark(TickMark.Inside);
category_axis_tick.setMinorTickMark(TickMark.Cross);
category_axis_tick.setTickMarkSpacing(2);

// Configure axis title
category_axis.setHasTitle(true);
category_axis.getAxisTitle().setText("CategoryAxisTitle");
category_axis.getAxisTitle().getFont().setSize(18);
category_axis.getAxisTitle().getFont().getColor().setRGB(Color.GetOrange());
```

## Configure Chart Axis

GcExcel Java provides you with several options that help you in configuring chart axis.

The following axes elements in the chart inserted in your spreadsheet are customizable:

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

### Axis title

Users can set custom style for the axis title using the **getAxisTitle** ('**getAxisTitle Method** in the on-line documentation) of the **IAxis** ('**IAxis Interface** in the on-line documentation) interface.

In order to configure the layout of the title of the chart axis, refer to the following example code.

#### Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

IAxis category_axis = shape.getChart().getAxes().item(AxisType.Category);
category_axis.setHasTitle(true);
category_axis.getAxisTitle().setText("CategoryAxisTitle");
category_axis.getAxisTitle().getFont().setSize(18);
category_axis.getAxisTitle().getFont().getColor().setRGB(Color.GetOrange());
```

## Gridlines

Users can also customize the style of major and minor gridlines in a chart axis using the **getMajorGridlines** ('**getMajorGridlines Method**' in the on-line documentation) method, **getMinorGridlines** ('**getMinorGridlines Method**' in the on-line documentation) method, **setHasMajorGridlines** ('**setHasMajorGridlines Method**' in the on-line documentation) method and **setHasMinorGridlines** ('**setHasMinorGridlines Method**' in the on-line documentation) method of the **IAxis** ('**IAxis Interface**' in the on-line documentation) interface.

In order to set major and minor gridlines' style, refer to the following example code.

Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(
    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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

IAxis value_axis = shape.getChart().getAxes().item(AxisType.Value);
value_axis.setHasMajorGridlines(true);
value_axis.setHasMinorGridlines(true);
value_axis.getMajorGridlines().getFormat().getLine().getColor().setRGB(Color.GetGray());
value_axis.getMajorGridlines().getFormat().getLine().setWeight(1);
value_axis.getMinorGridlines().getFormat().getLine().getColor().setRGB(Color.GetLightGray());
value_axis.getMinorGridlines().getFormat().getLine().setWeight(0.75);
value_axis.setMajorUnit(40);
value_axis.setMinorUnit(8);
value_axis.getMinorGridlines().getFormat().getLine().setStyle(LineStyle.ThickThin);
```

## Display unit label

Users can customize the display unit labels in the chart axis via configuring the display unit for the axis along with its label style using the **setDisplayUnit** ('**setDisplayUnit Method**' in the on-line documentation) method, **getDisplayUnitLabel** ('**getDisplayUnitLabel Method**' in the on-line documentation) method, **setDisplayUnitCustom** ('**setDisplayUnitCustom Method**' in the on-line documentation) method and **setHasDisplayUnitLabel** ('**setHasDisplayUnitLabel Method**' in the on-line documentation) method of the **IAxis** ('**IAxis Interface**' in the on-line documentation) interface.

In order to configure display unit for the axis and set custom label style, refer to the following example code.

Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(
    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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

IAxis value_axis = shape.getChart().getAxes().item(AxisType.Value);
value_axis.setDisplayUnit(DisplayUnit.Custom);
value_axis.setDisplayUnitCustom(100);
value_axis.setHasDisplayUnitLabel(true);
value_axis.getDisplayUnitLabel().getFont().getColor().setRGB(Color.GetCornflowerBlue());
value_axis.getDisplayUnitLabel().getFormat().getFill().getColor().setRGB(Color.GetOrange());
value_axis.getDisplayUnitLabel().getFormat().getLine().getColor().setRGB(Color.GetCornflowerBlue());
```

## Tick labels

Users can customize tick labels in chart axis via configuring the position and layout of the tick-mark labels using the **setTickLabelPosition** ('**setTickLabelPosition Method**' in the **on-line documentation**) method, **getTickLabels** ('**getTickLabels Method**' in the **on-line documentation**) method, **setTickLabelSpacing** ('**setTickLabelSpacing Method**' in the **on-line documentation**) method of the **IAxis** ('**IAxis Interface**' in the **on-line documentation**) interface.

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

### Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:D6").setValue(
    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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);

IAxis value_axis = shape.getChart().getAxes().item(AxisType.Value);
IAxis category_axis = shape.getChart().getAxes().item(AxisType.Category);

category_axis.setTickLabelPosition(TickLabelPosition.NextToAxis);
category_axis.setTickLabelSpacing(2);
category_axis.getTickLabels().getFont().getColor().setRGB(Color.GetDarkOrange());
category_axis.getTickLabels().getFont().setSize(12);
category_axis.getTickLabels().setNumberFormat("#,##0.00");
value_axis.getTickLabels().setNumberFormat("#,##0;[Red]#,##0");
```

You can also configure the text angle of tick-mark labels by using the **setOrientation** method of **ITickLabels** interface. The text angle can also be exported or imported to JSON or PDF document.

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

### Java

```
//create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

//add chart
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:B5").setValue(new Object[][]
{
    {null, "S1", "S2", "S3"},
    {1, -25, 25, 25},
    {2, 51, 36, 27},
    {3, 52, 80, 30},
    {4, 22, -20, 65},
    {5, 23, 69, 69}
});

shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);
IAxis category_axis = shape.getChart().getAxes().item(AxisType.Category);

//config tick label's angle
category_axis.getTickLabels().setOrientation(45);
```

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

## Floor

In GcExcel Java, floor represents the floor of a three-dimensional chart. Using floor as the charting object, you can format the area of a 3-D chart quickly and efficiently.

Users can set the line and fill format of the floor along with its thickness.

To set the floor format in a chart, refer to the following example code.

```
Java

IShape shape = worksheet.getShapes().addChart(ChartType.Column3D, 250, 20, 350, 250);
worksheet.getRange("A1:D6").setValue(
    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.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns,
true, true);

shape.getChart().getFloor().setThickness(5);
shape.getChart().getFloor().getFormat().getFill().getColor().setRGB(Color.GetYellow());
shape.getChart().getFloor().getFormat().getLine().getColor().setRGB(Color.GetRed());
```

## Data Label

GcExcel Java enables users to configure data labels so as to ensure the information depicted in a chart can be interpreted and visualized easily and quickly.

You can insert data labels in a chart using the methods of the **ISeries ('ISeries Interface' in the on-line documentation)** interface.

In order to configure data labels in a chart and set the data label text, refer to the following example code.

```
Java

IShape shape1 = worksheet.getShapes().addChart(ChartType.ColumnClustered, 200, 20, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);

// Set Series's all data labels and specific data label's format.
shape1.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true,
true);
ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);
```

```
// Set series1's all data label's format.
series1.getDataLabels().getFormat().getFill().getColor().setRGB(Color.GetGreen());
series1.getDataLabels().getFormat().getLine().getColor().setRGB(Color.GetRed());
series1.getDataLabels().getFormat().getLine().setWeight(3);

// set series1's specific data label's format.
series1.getDataLabels().get(2).getFormat().getFill().getColor().setRGB(Color.GetYellow());
series1.getPoints().get(2).getDataLabel().getFormat().getLine().getColor().setRGB(Color.GetBlue());
series1.getPoints().get(2).getDataLabel().getFormat().getLine().setWeight(5);

// Customize data label's text.
IShape shape2 = worksheet.getShapes().addChart(ChartType.ColumnClustered, 200, 20, 300, 300);
shape2.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true,
true);
ISeries series2 = shape.getChart().getSeriesCollection().get(0);
series2.setHasDataLabels(true);

// customize data label's text.
series2.getDataLabels().setShowCategoryName(true);
series2.getDataLabels().setShowSeriesName(true);
series2.getDataLabels().setShowLegendKey(true);
```

You can also configure the text angle for data labels by using the **setOrientation** method of **IDataLabel** interface. The text angle can also be exported or imported to JSON.

Refer to the following example code to set text angle for data label.

#### Java

```
//create a new workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

//add chart
IShape shape = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
worksheet.getRange("A1:B5").setValue(new Object[][]
{
{null, "S1"},
{"Item1", -20},
{"Item2", 30},
{"Item3", 50 },
{"Item3", 40 }
});
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:B5"), RowCol.Columns, true,
true);

ISeries series1 = shape.getChart().getSeriesCollection().get(0);
series1.setHasDataLabels(true);

//set series1's all data labels' angle
series1.getDataLabels().setOrientation(45);

//set series1's specific data label's angle
```

```
series1.getDataLabels().get(2).setOrientation(-45);

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

## Legends

Legends are the visual charting elements (keys associated with the data plotted on a chart) that automatically appear in spreadsheets when a user finishes the process of adding a chart.

Generally, legends help in quick interpretation of the charted data and are located at the right side of the chart. Also, they allow end users to figure out the series and series points representing different groups of data in a worksheet.

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

In GcExcel Java, you can 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 methods of the **ILegend** ('**ILegend Interface**' in the on-line documentation) interface, **ILegendEntries** ('**ILegendEntries Interface**' in the on-line documentation) interface and the **ICart** ('**ICart Interface**' in the on-line documentation) interface.

In order to configure some useful legend settings in your chart, refer to the following example code.

### Java

```
IShape shape = worksheet.getShapes().addChart(ChartType.Column3D, 200, 30, 300, 300);
Object[][] data = 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 } };
worksheet.getRange("A1:D6").setValue(data);
shape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns,
true, true);

shape.getChart().setHasLegend(true);
ILegend legend = shape.getChart().getLegend();

// position.
legend.setPosition(LegendPosition.Left);

// font.
legend.getFont().getColor().setRGB(Color.GetRed());
legend.getFont().setItalic(true);

// format.
legend.getFormat().getFill().getColor().setRGB(Color.GetPink());
legend.getFormat().getLine().getColor().setRGB(Color.GetBlue());

// Config legend entry font style.
ILegendEntry legendentry = legend.getLegendEntries().get(0);
legendentry.getFont().getColor().setRGB(Color.GetRed());
```



```
legendentry.getFont().setSize(15);
```

In case you want to delete the legend or a specific legend entry from your chart, refer to the following example code.

#### Java

```
// Delete legend.
IShape shape1 = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
shape1.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);
shape1.getChart().setHasLegend(true);
ILegend legend1 = shape1.getChart().getLegend();
legend1.delete();

// Delete legend entry.
IShape shape2 = worksheet.getShapes().addChart(ChartType.ColumnClustered, 250, 20, 360, 230);
shape2.getChart().getSeriesCollection().add(worksheet.getRange("A1:D6"), RowCol.Columns, true, true);
shape2.getChart().setHasLegend(true);
ILegend legend2 = shape2.getChart().getLegend();
ILegendEntry legendentry2 = legend2.getLegendEntries().get(0);
legendentry2.delete();
```

## Chart Types

**GcExcel** supports a wide range of chart types such as Area, Column, Line, Pie, Bar, Combo, Stock, Surface, Scatter, Radar, Statistical and Specialized charts. It also supports new Excel 2016 statistical and specialized chart types like Sunburst, Pareto, Treemap, Histogram, WaterFall, Box and Whisker, and Funnel. The new chart types represent and analyze hierarchical data better than conventional charts.

This topic gives a quick snapshot of all major chart types and their use cases.








Chart Type	Chart Snapshot	Use Case
<b>Area Charts</b> <ul style="list-style-type: none"><li>Area</li><li>Area3D</li><li>AreaStacked</li><li>AreaStacked100</li><li>AreaStacked1003D</li><li>AreaStacked3D</li></ul>		An Area chart is used to represent data that follows a time-series relationship. This type of chart is ideal when you need to show the plot change over time and depict the total value across a trend by showing the sum of the plotted values.





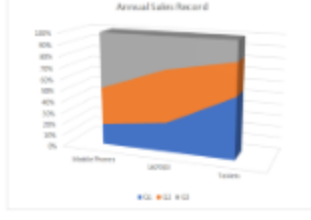
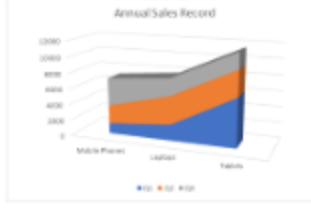
Chart Type	Chart Snapshot	Use Case
<b>Bar Charts</b> <ul style="list-style-type: none"> <li>BarClustered</li> <li>BarClustered3D</li> <li>BarStacked</li> <li>BarStacked100</li> <li>BarStacked1003D</li> <li>BarStacked3D</li> </ul>		<p>Bar charts are used for showing patterns and trends across different categories. In these charts, each horizontal bar corresponds to a category and its length corresponds to the value or measure of that category.</p>
<b>Column Charts</b> <ul style="list-style-type: none"> <li>Column3D</li> <li>ColumnClustered</li> <li>ColumnClustered3D</li> <li>ColumnStacked</li> <li>ColumnStacked100</li> <li>ColumnStacked1003D</li> <li>ColumnStacked3D</li> </ul>		<p>Unlike bar charts, Column charts use vertical columns/bars for representing data. These charts are generally used to plot data easily on X-axis.</p>
<b>Combo Chart</b> <ul style="list-style-type: none"> <li>Combo</li> </ul>		<p>The combo of two or more different charts can be used in the same plot area to compare the different data sets that are related to each other.</p>
<b>Line Charts</b> <ul style="list-style-type: none"> <li>Line</li> <li>Line3D</li> <li>LineMarkers</li> <li>LineMarkersStacked</li> <li>LineMarkersStacked100</li> <li>LineStacked</li> <li>LineStacked100</li> </ul>		<p>Line charts are used to plot continuously changing data against an interval of time. They can also be used to plot data against other continuous periodic values such as temperature, distance, humidity, share price, earnings per share etc.)</p>
<b>Pie Charts</b> <ul style="list-style-type: none"> <li>Pie</li> <li>Pie3D</li> <li>PieExploded</li> <li>PieExploded3D</li> <li>PieOfPie</li> <li>BarOfPie</li> <li>Doughnut</li> <li>DoughnutExploded</li> </ul>		<p>Pie charts are used to represent the relative contribution of various categories. It is one of the most commonly used charts and makes it easy to compare proportions by displaying the contribution of each value (slice) to a total (pie).</p>

Chart Type	Chart Snapshot	Use Case
<b>Stock Charts</b> <ul style="list-style-type: none"> <li>• StockHLC</li> <li>• StockOHLC</li> <li>• StockVHLC</li> <li>• StockVOHLC</li> </ul>		A Stock chart is used to illustrate fluctuations in data. It can represent fluctuations for stock, daily rainfall, or annual temperatures. Typically, this chart is ideal for analyzing financial data and visualizing stock information.
<b>Surface Charts</b> <ul style="list-style-type: none"> <li>• Surface</li> <li>• SurfaceTopView</li> <li>• SurfaceTopViewWireframe</li> <li>• SurfaceWireframe</li> </ul>		Surface charts are used to find the optimum combinations between two sets of data. As in a topographic map, the colors and patterns indicate the areas that are in the same range of values.
<b>XY (Scatter) Charts</b> <ul style="list-style-type: none"> <li>• XYScatter</li> <li>• XYScatterLines</li> <li>• XYScatterLinesNoMarkers</li> <li>• XYScatterSmooth</li> <li>• XYScatterSmoothNoMarkers</li> <li>• Bubble</li> <li>• Bubble3DEffect</li> </ul>		An XY chart (also called scatter diagram) is a two-dimensional chart that shows the relationship between two variables. In a scatter graph, both horizontal and vertical axes are value axes that plot numeric data to show the correlation between two variables.
<b>Radar Charts</b> <ul style="list-style-type: none"> <li>• Radar</li> <li>• Radar Filled</li> <li>• Radar Markers</li> </ul>		Radar charts are radial charts that help in visualizing comparison of two or more groups of values against various features or characteristics. These charts represent each variable on a separate axis, which are arranged radially at equal distances from each other.
<b>Statistical Charts</b> <ul style="list-style-type: none"> <li>• Box and Whisker</li> <li>• Histogram</li> <li>• Waterfall</li> <li>• Pareto</li> </ul>		Statistical charts help summarize and add visual meaning to key characteristics of data, including range, distribution, mean and median. It can also be used to in present and interpret statistical data in graphical format.
<b>Specialized Charts</b> <ul style="list-style-type: none"> <li>• Sunburst</li> <li>• Treemap</li> <li>• Funnel</li> </ul>		Specialized chart types provided by GcExcel have unique data representation to show hierarchies and relationships. Such visual comparisons allow users to analyze the data thoroughly.

## Area Chart

An **Area Chart** can be used to represent the change in one or more data quantities over time. It is similar to a line graph. In area charts, the data points are plotted and connected by line segments. This helps in showing the magnitude of the value at different times. Unlike in line charts, the area between the line and x-axis is filled with color or shading in area charts.

GcExcel supports the following types of area charts.

Chart Type	Chart Snapshot	Use Case
Area		Area chart is used to depict the data series as colored regions that help in comparing the values of multiple series for the same data point. This chart shows trends over time.
Area3D		Area3D chart is used to represent the chart demonstration in 3D, which is a modification of 2D Area chart. It does not have a third dimension, it only looks volumetric in appearance.
AreaStacked		AreaStacked chart is used to depict data series as stacked regions with different colors that help in performing comparisons between multiple series for the same data point. This chart shows the trend of the contribution of each value over time or other categorical data.
AreaStacked100		AreaStacked100 chart is used to depict the series of data points with positive and negative values shown over time to reveal values of multiple series for the same data point. This chart shows the percentage that each value contributes over time or other categorical data.
AreaStacked1003D		AreaStacked1003D is used to represent the AreaStacked100 chart in 3D, which looks volumetric in appearance.
AreaStacked3D		AreaStacked3D chart is used to represent AreaStacked chart in 3D, which is a modification of the 2D Area chart.

## Using Code

Refer to the following example code to add Area Stacked Chart:

Java

```
private static void AreaCharts() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);
    // Prepare data for chart
    worksheet.getRange("A1:D4")
        .setValue(new Object[][] {
            { null, "Q1", "Q2", "Q3" },
            { "Mobile Phones", 1330, 2345, 3493 },
            { "Laptops", 2032, 3632, 2197 },
            { "Tablets", 6233, 3270, 2030 } });
    worksheet.getRange("A:D").getColumns().autoFit();
    // Add Area Chart
    IShape areaChartShape = worksheet.getShapes().addChart(ChartType.Area3D, 250, 20,
        360, 230);

    // Adding series to SeriesCollection
    areaChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D4"),
        RowCol.Columns, true, true);



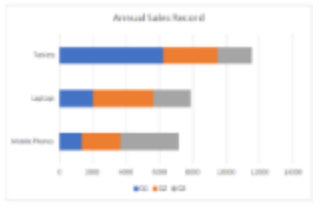



    // Configure Chart Title
    areaChartShape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs()
        .add("Annual Sales Record");

    // Saving workbook to Xlsx
    workbook.save("18-AreaChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Bar Chart

**Bar charts** compare categorical data through horizontal bars, where length of each bar represents the value of the corresponding category. In bar charts, categories are organized along the vertical axis and data values along the horizontal axis. For example, sales of various product categories can be presented through a bar chart.

GcExcel supports the following types of bar charts.

Chart Type	Chart Snapshot	Use Case
BarClustered		BarClustered Chart can be used to display the comparisons of values across different categories.
BarClustered3D		BarClustered3D chart is used to display the chart demonstration in 3D, which is a modification of 2D BarClustered chart. It does not have a third dimension, it only looks volumetric in appearance.
BarStacked		BarStacked chart is used to display the relationship of each item/category to the whole in two-dimensional and three-dimensional rectangles.
BarStacked3D		BarStacked3D chart is used to represent the BarStacked chart demonstration in 3D, which looks volumetric in appearance.
BarStacked100		BarStacked100 chart is used to display the comparisons of percentage that each of the values contribute to the total across different categories.
BarStacked1003D		BarStacked1003D chart is used to represent the BarStacked100 chart demonstration in 3D, which is a modification of 2D chart in appearance.

## Using Code

Refer to the following example code to add Bar Stacked Chart:

Java

```
private static void BarCharts() {
```

```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Prepare data for chart
worksheet.getRange("A1:D4")
    .setValue(new Object[][] {
        { null, "Q1", "Q2", "Q3" },
        { "Mobile Phones", 1330, 2345, 3493 },
        { "Laptops", 2032, 3632, 2197 },
        { "Tablets", 6233, 3270, 2030 } });
worksheet.getRange("A:D").getColumns().autoFit();
// Add BarStaked Chart
IShape areaChartShape = worksheet.getShapes().addChart(ChartType.BarStacked, 250,
20, 360, 230);

// Adding series to SeriesCollection
areaChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D4"),
RowCol.Columns, true, true);

// Configure Chart Title
areaChartShape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs()
    .add("Annual Sales Record");

// Saving workbook to Xlsx
workbook.save("19-BarChart.xlsx", SaveFileFormat.Xlsx);
```

## Column Chart

**Column charts** are vertical versions of bar charts and use x-axis as a category axis. Column charts are preferred where number of values is too large to be used on an x-axis, while bar charts are preferred where long category titles are difficult to fit on an x-axis. For example, population share of different countries across the globe can be represented using a column chart.

GcExcel supports the following types of column charts.







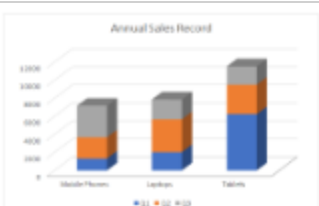
Chart Type	Chart Snapshot	Use Case
Column3D		Column3D chart is used to display the chart demonstration in 3D which is a modification of 2D Column chart. It does not have a third dimension, it only looks volumetric in appearance.

Chart Type	Chart Snapshot	Use Case
ColumnClustered		Column clustered chart is used to compare different values across different categories and show them in two-dimensional or three-dimensional vertical rectangles. This chart can be stacked normally in a regular way just like any other chart.
ColumnClustered3D		Column clustered chart to represent the ColumnClustered chart demonstration in 3D, which looks volumetric in appearance.
ColumnStacked		ColumnStacked chart is used to display the relationship of specific items to the whole across different categories and plot values in two-dimensional or three-dimensional vertical rectangles. This chart stacks the data series vertically (in a vertical direction).
ColumnStacked100		ColumnStacked100 chart is used to perform comparisons of percentages that each of the values are contributing to the total, across all your categories in the spreadsheet. This chart stacks the data series vertically and also equalizes the plotted values to meet 100%. The plotted values are displayed in two-dimensional and three-dimensional rectangles.
ColumnStacked1003D		ColumnStacked1003D is used to represent the ColumnStacked100 chart demonstration in 3D, which is a modification of 2D chart in appearance.
ColumnStacked3D		ColumnStacked3D chart is used to represent the ColumnStacked chart demonstration in 3D, which looks volumetric in appearance.

## Using Code

Refer to the following example code to add Column Stacked 3D Chart:

Java

```
private static void ColumnCharts() {
```



```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Prepare data for chart
worksheet.getRange("A1:D4")
    .setValue(new Object[][] {
        { null, "Q1", "Q2", "Q3" },
        { "Mobile Phones", 1330, 2345, 3493 },
        { "Laptops", 2032, 3632, 2197 },
        { "Tablets", 6233, 3270, 2030 } });
worksheet.getRange("A:D").getColumns().autoFit();
// Add Column Chart
IShape areaChartShape = worksheet.getShapes().addChart(ChartType.Column3D, 250, 20,
360, 230);


// Adding series to SeriesCollection
areaChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D4"),
RowCol.Columns, true, true);

// Configure Chart Title
areaChartShape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs()
    .add("Annual Sales Record");

// Saving workbook to Xlsx
workbook.save("20-ColumnChart.xlsx", SaveFileFormat.Xlsx);
```

## Combo Chart

**Combo chart** is a combination of two or more chart types in a single plot area. For instance, a bar and line chart in a single plot. Combination charts are best used to compare the different data sets that are related to each other, such as actual and target values, total revenue and profit, temperature and precipitation etc. Note that these charts may require multiple axes to cater different scales.

Chart Type	Chart Snapshot	Use Case
Combo		Combo chart can be used to interpret and understand different type of data that is completely unrelated (for instance: price and volume) or to plot one or more data series on the secondary axis.

### Using Code

Refer to the following example code to add Combo Chart:

Java

```
private static void ComboCharts() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:C17")
        .setValue(new Object[][] {
            { "Mobile Phones", "Laptops", "Tablets" },
            { 1350, 120, 75 },
            { 1500, 90, 35 },
            { 1200, 80, 50 },
            { 1300, 80, 80 },
            { 1750, 90, 100 },
            { 1640, 120, 130 },
            { 1700, 120, 95 },
            { 1100, 90, 80 },
            { 1350, 120, 75 },
            { 1500, 90, 35 },
            { 1200, 80, 50 }, });
    worksheet.getRange("A:C").getColumns().autoFit();

    // Add Combination Chart
    IShape comboChartShape = worksheet.getShapes().addChart(ChartType.Combo, 250, 20,
360, 230);
    // Adding series to SeriesCollection
    comboChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:C17"),
RowCol.Columns);

    // Configure Chart Title
    comboChartShape.getChart().getChartTitle().setText("Annual Sales Record-Combination
Chart");
    ISeries series1 = comboChartShape.getChart().getSeriesCollection().get(0);
    ISeries series2 = comboChartShape.getChart().getSeriesCollection().get(1);
    ISeries series3 = comboChartShape.getChart().getSeriesCollection().get(2);

    // Change series type to make it Combination chart of different ChartTypes
    series1.setChartType(ChartType.Area);
    series2.setChartType(ChartType.ColumnStacked);
    series3.setChartType(ChartType.Line);

    // Set axis group
    series2.setAxisGroup(AxisGroup.Secondary);
}
```

```

series3.setAxisGroup(AxisGroup.Secondary);

// Configure axis scale and unit
IAxis value_axis = comboChartShape.getChart().getAxes().item(AxisType.Value);
IAxis value_second_axis = comboChartShape.getChart().getAxes().item(AxisType.Value,
AxisGroup.Secondary);
value_axis.setMaximumScale(1800);
value_axis.setMajorUnit(450);
value_second_axis.setMaximumScale(300);
value_second_axis.setMajorUnit(75);

// Saving workbook to Xlsx
workbook.save("24-ComboChart.xlsx", SaveFileFormat.Xlsx);

```

## Line Chart

**Line charts** are the most basic charts that are created by connecting the data points with straight lines. These charts are used to visualize a trend in data by comparing values against periodic intervals such as time, temperature etc. Some examples that can be well depicted using line charts are closing prices of a stock in a given time frame and monthly average sale of a product.

GcExcel supports the following types of line charts.


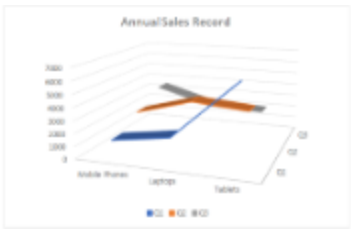





Chart Type	Chart Snapshot	Use Case
Line		Line chart is used to depict the data values plotted over time to display the trends. It shows continuous data over time on an evenly scaled Axis.
Line3D		Line3D chart is used to display the chart demonstration in 3D, which is a modification of 2D Line chart.
LineMarkers		LineMarkers chart is used to display data values shown with markers. It is ideal to use this chart when there are many categories or approximate values.
LineMarkersStacked		LineMarkersStacked is used to display data values with markers, typically showing the trend of contribution of each value over time or evenly spaced categories.

Chart Type	Chart Snapshot	Use Case
LineMarkersStacked100		LineMarkersStacked100 chart is used to display individual data values with markers, typically showing the trend of the percentage each value that has been contributed over time or evenly spaced categories. It is ideal to use this chart when there are many categories or approximate values.
LineStacked		LineStacked chart is used to display stacked line to depict the trend of contribution of each data value or ordered category over different time intervals.
LineStacked100		LineStacked100 chart is used to display displays trends in terms of the percentage that each data value or ordered category has contributed (to the whole) over different time intervals.

## Using Code

Refer to the following example code to add LineStacked100 chart:

### Java

```
private static void LineCharts() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);
    // Prepare data for chart
    worksheet.getRange("A1:D4")
        .setValue(new Object[][] {
            { null, "Q1", "Q2", "Q3" },
            { "Mobile Phones", 1330, 2345, 3493 },
            { "Laptops", 2032, 3632, 2197 },
            { "Tablets", 6233, 3270, 2030 } });
    worksheet.getRange("A:D").getColumns().autoFit();
    // Add Line Chart
    IShape areaChartShape = worksheet.getShapes().addChart(ChartType.LineMarkers, 250,
20, 360, 230);

    // Adding series to SeriesCollection
    areaChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D4"),
```

```
RowCol.Columns, true, true);

    // Configure Chart Title









areaChartShape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs()
    .add("Annual Sales Record");

    // Saving workbook to Xlsx
workbook.save("21-LineChart.xlsx", SaveFileFormat.Xlsx);
```

## Pie Chart

**Pie charts**, the most common tools used for data visualization, are circular graphs that display the proportionate contribution of each category, which is represented by a pie or a slice. The magnitude of the dependent variable is proportional to the angle of the slice. These charts can be used for plotting just one series with non-zero and positive values.

GcExcel supports the following types of pie charts.

Chart Type	Chart Snapshot	Use Case
Pie		Pie chart is used to display a single data series in a circle-type structure, with each sector representing a different category.
Pie3D		Pie3D chart is used to display the chart demonstration in 3D which is a modification of 2DPie chart in terms of appearance.
PieExploded		PieExploded chart is used to pull all of the slices out of a pie chart and view the sectors separately in pieces.
PieExploded3D		PieExploded 3D chart is used display the chart demonstration in 3D which is a modification of 2DPieExploded chart.
PieOfPie		PieofPie chart is used to separate the slices from the main pie chart and display them in an additional pie chart.
BarOfPie		BarofPie chart is used to separate the slices from the main pie chart and display them in an additional stacked bar chart.
Doughnut		Doughnut chart is used to display multiple data series concurrently, with each ring depicting a single data series.
DoughnutExploded		DoughnutExploded is used to pull all slices out of a DoughnutExploded chart and view the sectors separately in pieces.

### Using Code

Refer to the following code to add Doughnut Exploded chart:

```
Java
private static void PieCharts() {
```

```

// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Prepare data for chart
worksheet.getRange("A1:D4")
    .setValue(new Object[][] {
        { null, "Q1", "Q2", "Q3" },
        { "Mobile Phones", 1330, 2345, 3493 },
        { "Laptops", 2032, 3632, 2197 },
        { "Tablets", 6233, 3270, 2030 } });
worksheet.getRange("A:D").getColumns().autoFit();
// Add Pie Chart
IShape areaChartShape = worksheet.getShapes().addChart(ChartType.Pie3D, 250, 20,
360, 230);

// Adding series to SeriesCollection
areaChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D4"),
RowCol.Columns, true, true);

// Configure Chart Title
areaChartShape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs()
    .add("Annual Sales Record");





// Saving workbook to Xlsx
workbook.save("22-PieChart.xlsx", SaveFileFormat.Xlsx);

```

## Stock Chart

**Stock chart** is used to illustrate fluctuations in data over a time. It can represent fluctuations in stock, rainfall, or annual temperatures. The data arranged in columns or rows of a worksheet can be plotted in a Stock chart.

GcExcel supports the following types of Stock charts.

Chart Type	Chart Snapshot	Use Case
StockHLC		A high-low-close chart displays the data values organized in the order: high, low, close with the close value lying in between the high and low values.
StockOHLC		An open-high-low-close chart displays the data values organized in the order: open, high, low and close.
StockVHLC		A volume-high-low-close chart displays the data values organized in the order: volume, high, low and close.
StockVOHLC		A volume-open-high-low-close chart displays the data values organized in the order : volume, open, high, low and close.

## Using Code

Refer the following code to add StockVOHLC chart:

Java

```
private static void StockCharts() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:D17")
        .setValue(new Object[][] { { null, "High", "Low", "Close" },
            { new GregorianCalendar(2019, 9, 1), 105.76, 92.38, 100.94 },
            { new GregorianCalendar(2019, 9, 2), 102.45, 90.14, 93.45 },
            { new GregorianCalendar(2019, 9, 3), 102.11, 85.01, 99.89 },
            { new GregorianCalendar(2019, 9, 4), 106.01, 94.04, 99.45 },
            { new GregorianCalendar(2019, 9, 5), 108.23, 98.16, 104.33 },
            { new GregorianCalendar(2019, 9, 8), 107.7, 91.02, 102.17 },
            { new GregorianCalendar(2019, 9, 9), 110.36, 101.62, 110.07 },
            { new GregorianCalendar(2019, 9, 10), 115.97, 106.89, 112.39 },
            { new GregorianCalendar(2019, 9, 11), 120.32, 112.15, 117.52 },
            { new GregorianCalendar(2019, 9, 12), 122.03, 114.67, 114.75 },
            { new GregorianCalendar(2019, 9, 15), 120.46, 106.21, 116.85 },
            { new GregorianCalendar(2019, 9, 16), 118.08, 113.55, 116.69 },
            { new GregorianCalendar(2019, 9, 17), 128.23, 110.91, 117.25 },
            { new GregorianCalendar(2019, 9, 18), 120.55, 108.09, 112.52 },
            { new GregorianCalendar(2019, 9, 19), 112.58, 105.42, 109.12 },
            { new GregorianCalendar(2019, 9, 22), 115.23, 97.25, 101.56 },
        });
    worksheet.getRange("A:D").getColumns().autoFit();

    // Add Stock Chart
    IShape stockChartshape = worksheet.getShapes().addChart(ChartType.StockHLC, 350, 20,
        360, 230);

    // Adding series to SeriesCollection
    stockChartshape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D17"),
        RowCol.Columns);

    // Configure Chart Title
    stockChartshape.getChart().getChartTitle().setText("Market Data Analysis");

    // Configure value axis
    IAxis valueAxis = stockChartshape.getChart().getAxes().item(AxisType.Value);
```

```

valueAxis.setMinimumScale(80);
valueAxis.setMaximumScale(140);
valueAxis.setMajorUnit(15);

// Configure category axis
IAxis categoryAxis = stockChartshape.getChart().getAxes().item(AxisType.Category);
categoryAxis.setCategoryType(CategoryType.CategoryScale);
categoryAxis.setMajorTickMark(TickMark.Outside);
categoryAxis.setTickLabelSpacingIsAuto(false);
categoryAxis.setTickLabelSpacing(5);

// Configure Close Series Style
ISeries series_close = stockChartshape.getChart().getSeriesCollection().get(2);
series_close.setMarkerStyle(MarkerStyle.Diamond);
series_close.setHas3DEffect(true);





// Saving workbook to Xlsx
workbook.save("23-StockChart.xlsx", SaveFileFormat.Xlsx);

```

## Surface Chart

**Surface charts** are useful when you want to find the optimum combinations between two data sets. As in a topographic map, the colors and patterns indicate the areas that are in the same range of values. A surface chart plots data on a three-dimensional surface, in a similar way that topographic maps plots elevation. The colors and patterns represent values within the same range. This chart type is especially useful for finding the optimum results when comparing two or more sets of data.

GcExcel supports the following types of Surface charts.

Chart Type	Chart Snapshot	Purpose
Surface		Surface chart is a chart with a 3-D visual effect.
SurfaceTopView		SurfaceTopView chart depicts surface chart viewed from above.
SurfaceTopViewWireframe		SurfaceTopViewWireframe chart depicts surface chart viewed from above with no fill color.
SurfaceWireframe		SurfaceWireframe chart depicts surface chart with a 3-D visual effect and no fill color.

### Using Code

Refer to the following code to add SurfaceWireframe chart.

```

Java

private static void SurfaceCharts() {
    // Initialize workbook

```



```
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Prepare data for chart
worksheet.getRange("A1:D4")
    .setValue(new Object[][] {
        { null, "Q1", "Q2", "Q3" },
        { "Mobile Phones", 1330, 2345, 3493 },
        { "Laptops", 2032, 3632, 2197 },
        { "Tablets", 6233, 3270, 2030 } });
worksheet.getRange("A:D").getColumns().autoFit();
// Add Surface Chart
IShape areaChartShape = worksheet.getShapes().addChart(ChartType.Surface, 250, 20,
360, 230);

// Adding series to SeriesCollection
areaChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D4"),
RowCol.Columns, true, true);

// Configure Chart Title
areaChartShape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs()
    .add("Annual Sales Record");

// Saving workbook to Xlsx
workbook.save("25-SurfaceChart.xlsx", SaveFileFormat.Xlsx);
```

## XY (Scatter) chart

**Scatter chart** is used to illustrate relationships between individual items or categories. This chart is ideal for showing comparisons for scientific, statistical and engineering data. The data arranged in columns or rows of a worksheet can be plotted in a Scatter chart.

Unlike other charts, a scatter chart displays the actual values of the x and y variables in horizontal axis and vertical axis in the plot area. Typically, this chart combines the x and y values into single data points and displays them at irregular intervals. Also, this chart does not make use of the category axis because both horizontal axis (primary axis) and vertical axes (secondary axis) are value axes.

GcExcel supports the following types of Scatter charts.




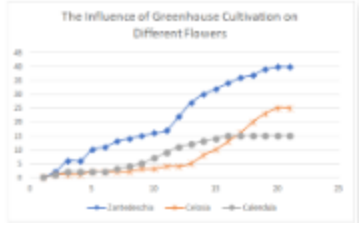
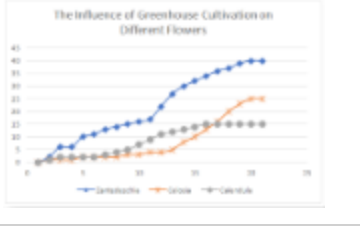


Chart Type	Chart Snapshot	Use Case
XYScatter		A clustered Scatter chart displays the data points based on a selected data range. This helps the users to analyze and determine the relationship between x and y variables.
XYScatterLines		A scatter chart with straight lines displays a straight connecting line between data points in a particular series

Chart Type	Chart Snapshot	Use Case
		without showing the individual points.
XYScatterLinesNoMarkers		A scatter chart with straight lines and no data markers displays a smooth curve that connects all the data points in a particular series.
XYScatterSmooth		A scatter chart with smooth lines displays a connecting line between data points in a particular series without showing the individual points.
XYScatterSmoothNoMarkers		A scatter chart with smooth lines and no data markers displays a smooth curve that connects all the data points in a particular series.
Bubble		A bubble chart is ideal for financial data analysis. It displays the variations of a scatter chart where data points are replaced with bubbles and a third dimension is represented (Z axis) in the size of the bubbles. This chart plots z(size) values as well as x values and y values. Typically, this chart can be used when you want to plot three data series. The size of the bubbles is determined by the values in the third data series.
Bubble3DEffect		Bubble3DEffect chart can be used to display the chart demonstration in 3D, which is a modification of 2DBubble chart. It does not have a third dimension, it only looks volumetric in appearance.

## Using Code

Refer to the following code to add a XY Scatter chart:

Java

```
private static void ScatterCharts() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:D22").setValue(new Object[][] { { "Index", "Zantedeschia",
"Celosia", "Calendula" },
```

```

        { 0, 0, 0, 0 }, { 1, 2, 1, 1 }, { 2, 6, 1, 2 }, { 3, 6, 1, 2 }, { 4, 10, 2,
2 }, { 5, 11, 2, 2 },
        { 6, 13, 2, 3 }, { 7, 14, 2, 4 }, { 8, 15, 3, 5 }, { 9, 16, 3, 7 }, { 10,
17, 4, 9 }, { 11, 22, 4, 11 },
        { 12, 27, 5, 12 }, { 13, 30, 8, 13 }, { 14, 32, 10, 14 }, { 15, 34, 13, 15
}, { 16, 36, 16, 15 },
        { 17, 37, 20, 15 }, { 18, 39, 23, 15 }, { 19, 40, 25, 15 }, { 20, 40, 25, 15
} });
    worksheet.getRange("A:D").getColumns().autoFit();
    // Add XYScatter Chart
    IShape xyScatterChartshape =
worksheet.getShapes().addChart(ChartType.XYScatterLines, 250, 20, 360, 230);

    // Adding series to SeriesCollection

xyScatterChartshape.getChart().getSeriesCollection().add(worksheet.getRange("B1:B22"),
RowCol.Columns);

xyScatterChartshape.getChart().getSeriesCollection().add(worksheet.getRange("C1:C22"),
RowCol.Columns);

xyScatterChartshape.getChart().getSeriesCollection().add(worksheet.getRange("D1:D22"),
RowCol.Columns);

    // Configure Chart Title
xyScatterChartshape.getChart().getChartTitle()
    .setText("The Influence of Greenhouse Cultivation on Different Flowers");

    // Configure Markers style
ISeries series1 = xyScatterChartshape.getChart().getSeriesCollection().get(0);
series1.setMarkerStyle(MarkerStyle.Diamond);
series1.setMarkerSize(7);
ISeries series2 = xyScatterChartshape.getChart().getSeriesCollection().get(1);
series2.setMarkerStyle(MarkerStyle.Star);
series2.setMarkerSize(7);
ISeries series3 = xyScatterChartshape.getChart().getSeriesCollection().get(2);
series3.setMarkerStyle(MarkerStyle.Circle);
series3.setMarkerSize(7);

    // Saving workbook to Xlsx
workbook.save("26-ScatterChart.xlsx", SaveFileFormat.Xlsx);




```

## Radar Chart

A **Radar chart** is used to display circular visual representation of a 2-dimensional data. One can think of it as a circular XY chart. These charts represent each variable on a separate axis, which are arranged radially at equal distances from each other. Each of these axes share the same tick marks and scale. The data for each observation is plotted along these axis

and then joined to form a polygon. Radar charts are generally used for analyzing performance or comparing values such as revenue and expense.

GcExcel supports the following types of radar charts.

Chart Type	Chart Snapshot	Use Case
Radar		Radar chart type can be used to represent multivariate data plotted in rows and columns in the graphical format.
RadarFilled		RadarFilled chart type can be used to display radar chart with areas highlighted by different colored regions for each value.
RadarMarkers		RadarMarkers chart can be used to display radar chart with markers representing data for each value along with areas highlighted by different line colors.

## Using Code

Refer to the following code to add RadarMarkers chart:

Java

```
private static void RadarCharts() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:D7")
        .setValue(new Object[][] {
            { null, "Lisa", "Tim", "Jim" },
            { "Mathematics", 87, 64, 79 },
            { "English", 79, 58, 78 },
            { "History", 62, 70, 82 },
            { "Biology", 85, 63, 54 },
            { "Geography", 64, 85, 75 },
            { "Zoology", 62, 79, 94 } });
    worksheet.getRange("A:D").getColumns().autoFit();
    // Add Radar Chart
    IShape radarChartShape = worksheet.getShapes().addChart(ChartType.Radar, 250, 20,
        360, 230);

    // Adding series to SeriesCollection
    radarChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D7"),
        RowCol.Columns, true, true);

    // Configure Chart Title
    radarChartShape.getChart().getChartTitle().getTextFrame().getTextRange().getParagraphs()
```




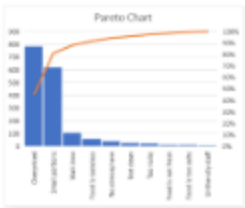
```
.add("Test Score Analysis");

// Saving workbook to Xlsx
workbook.save("27-RadarChart.xlsx", SaveFileFormat.Xlsx);
```

## Statistical Chart

**Statistical charts** can be used to present and interpret statistical data in graphical format. GcExcel supports statistical chart types like Box and Whisker, Histogram, Waterfall and Pareto. Such chart types add visual meaning to the represented data.

GcExcel supports the following types of Statistical chart types:

Chart Type	Chart Snapshot	Use Case
Box&Whisker		Box and Whisker charts are often used in Marketing Analysis, Statistical Analysis and General Analysis.
Histogram		Histogram is a common chart used in statistics. It can be used in scenarios, such as analysis of distribution/sales of books in a book store.
Waterfall Chart		Waterfall charts finds application in analyzing project gains including the number of contracts carried forwarded each year, contracts cancelled, tasks completed etc.
Pareto Chart		Pareto charts graphically summarize the process problems in ranking order from the most frequent to the least one.

## Box Whisker

**BoxWhisker** charts are statistical charts that display the distribution of numerical data through quartiles, means and outliers. As the name suggests, these values are represented using boxes and whiskers, where boxes show the range of quartiles (lower quartile, upper quartile and median), while whiskers indicate the variability outside the upper and lower quartiles. Any point outside the whiskers is said to be an outlier. These charts are useful for comparing distributions between many groups or data sets. For instance, you can easily display the variation in monthly temperature of two cities.

### Using Code

Refer to the following code to add Box and Whisker chart:

```
Java

private static void BoxWhiskerChart() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
```

```
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Prepare data for chart
worksheet.getRange("A1:D16").setValue(new Object[][] {
    { "Course", "SchoolA", "SchoolB", "SchoolC" },
    { "English", 78, 72, 45 },
    { "Physics", 61, 55, 65 },
    { "English", 63, 50, 65 },
    { "Math", 62, 73, 83 },
    { "English", 46, 64, 75 },
    { "English", 58, 56, 67 },
    { "Math", 60, 51, 67 },
    { "Math", 62, 53, 66 },
    { "English", 63, 54, 64 },
    { "English", 90, 52, 67 },
    { "Physics", 70, 82, 64 },
    { "English", 60, 56, 67 },
    { "Math", 73, 56, 75 },
    { "Math", 63, 58, 74 },
    { "English", 73, 84, 45 } });
worksheet.getRange("A:D").getColumns().autoFit();
// Add BoxWhisker chart
IShape boxWhiskerChartshape = worksheet.getShapes().addChart(ChartType.BoxWhisker,
300, 20, 300, 200);

boxWhiskerChartshape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D16"));

// Configure Chart Title
boxWhiskerChartshape.getChart().getChartTitle().setText("Box & Whisker Chart");

// Config value axis's scale
IAxis value_axis = boxWhiskerChartshape.getChart().getAxes().item(AxisType.Value,
AxisGroup.Primary);
value_axis.setMinimumScale(40);
value_axis.setMaximumScale(70);

// Configure the display of box&whisker plot
ISeries series = boxWhiskerChartshape.getChart().getSeriesCollection().get(0);
series.setShowInnerPoints(true);
series.setShowOutlierPoints(false);
series.setShowMeanMarkers(false);
series.setShowMeanLine(true);
series.setQuartileCalculationInclusiveMedian(true);

// Saving workbook to Xlsx
workbook.save("28-BoxWhiskerChart.xlsx", SaveFileFormat.Xlsx);
```

## Histogram

**Histograms** are visual representation of data distribution over a continuous interval or certain time period. These charts comprise vertical bars to indicate the frequency in each interval or bin created by dividing the raw data values into a series of consecutive and non-overlapping intervals. Hence, histograms help in estimating the range where maximum values fall as well as in knowing the extremes and gaps in data values, if there are any. For instance, histogram can help you find the range of height in which maximum students of a particular age group fall.

### Using Code

Refer to the following code to add a Histogram chart.

Java

```
private static void HistogramChart() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:B11")
        .setValue(new Object[][] {
            { "Complaint", "Count" },
            { "Too noisy", 27 },
            { "Overpriced", 789 },
            { "Food is tasteless", 65 },
            { "Food is not fresh", 19 },
            { "Food is too salty", 15 },
            { "Not clean", 30 },
            { "Unfriendly staff", 12 },
            { "Wait time", 109 },
            { "No atmosphere", 45 },
            { "Small portions", 621 } });
    worksheet.getRange("A:B").getColumns().autoFit();
    // Add Histogram Chart
    IShape histogramchartShape = worksheet.getShapes().addChart(ChartType.Histogram, 300, 30,
    300, 250);

    // Set range"A1:B11" as the histogram chart series
    histogramchartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:B11"));

    // Sets bins type by category

    histogramchartShape.getChart().getChartGroups().get(0).setBinsType(BinsType.BinsTypeCategorical);

    // Configure Chart Title
    histogramchartShape.getChart().getChartTitle().setText("Histogram Chart");

    // Saving workbook to Xlsx
    workbook.save("29-HistogramChart.xlsx", SaveFileFormat.Xlsx);
}
```

## Waterfall Chart

A **waterfall chart** shows the aggregate of values as they are added or subtracted. This type of chart is useful to understand how the initial value is affected by a series of positive and negative values. Waterfall charts can be used for viewing fluctuations in product earnings, net income or profit analysis.

### Using Code

Refer the following code for adding Waterfall chart.

Java

```
private static void WaterfallChart() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:B8")
        .setValue(new Object[][] {
            { "Starting Amt", 130 },
            { "Measurement 1", 25 },
            { "Measurement 2", -75 },
            { "Subtotal", 80 },
            { "Measurement 3", 45 },
            { "Measurement 4", -65 },
            { "Measurement 5", 80 },
            { "Total", 140 } });
    worksheet.getRange("A:A").getColumns().autoFit();

    // Add Waterfall Chart
    IShape waterfallChartShape = worksheet.getShapes().addChart(ChartType.Waterfall,
        300, 20, 300, 250);

    waterfallChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:B8"));

    // Configure Chart Title
    waterfallChartShape.getChart().getChartTitle().setText("Waterfall Chart");

    // Set subtotal & total points
    IPoints points =
        waterfallChartShape.getChart().getSeriesCollection().get(0).getPoints();
    points.get(3).setIsTotal(true);
    points.get(7).setIsTotal(true);

    // Connector lines are not shown
    ISeries series = waterfallChartShape.getChart().getSeriesCollection().get(0);
```



```
series.setShowConnectorLines(false);

// Saving workbook to Xlsx
workbook.save("30-WaterfallChart.xlsx", SaveFileFormat.Xlsx);
```

## Pareto Chart

GcExcel supports Pareto chart, also known as Pareto distribution diagram. It is a vertical bar graph in which values are plotted left to right, in decreasing order of relative frequency. Pareto charts are useful for task prioritizing. The chart gives a hint about the variables that have the greatest effect on a given system.

Pareto chart can be used to highlight the most important factor from a given set of factors. For example, quality control, inventory control, and customer grievance handling are some areas where Pareto chart analysis can be used.

### Using code

Refer the following code to add Pareto chart:

Java

```
private static void ParetoChart() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:B11")
        .setValue(new Object[][] {
            { "Complaint", "Count" },
            { "Too noisy", 27 },
            { "Overpriced", 789 },
            { "Food is tasteless", 65 },
            { "Food is not fresh", 19 },
            { "Food is too salty", 15 },
            { "Not clean", 30 },
            { "Unfriendly staff", 12 },
            { "Wait time", 109 },
            { "No atmosphere", 45 },
            { "Small portions", 621 } });
    worksheet.getRange("A:B").getColumns().autoFit();
    // Add Pareto Chart
    IShape paretochartShape = worksheet.getShapes().addChart(ChartType.Pareto, 300, 30,
    300, 250);

    // Set range"A1:B11" as the pareto chart series
    paretochartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:B11"));




    // Configure Chart Title
```

```
paretochartShape.getChart().getChartTitle().setText("Pareto Chart");

// Saving workbook to Xlsx
workbook.save("31-ParetoChart.xlsx", SaveFileFormat.Xlsx);
```

## Specialized Chart

GcExcel supports the following types of Specialized chart types such as Sunburst, Treemap and Funnel. The following table covers the different specialized chart types, chart snapshots and their use-cases.

Chart Type	Chart Snapshot	Use Case
Sunburst		Sunburst charts can be used to break down data into different entities for identifying and visualizing multilevel parent child relationships in different business scenarios quickly and efficiently.
Treemap		Treemap charts can be used to display large amount of hierarchical data without any space constraints. You can plot more than tens of thousands of data points.
Funnel		Funnel charts help in visualizing the sequential stages in a linear process such as recruitment process, order fulfillment cycles and promotional campaigns.

## Sunburst

**Sunburst**, also known as a multi-level pie chart, is ideal for visualizing multi-level hierarchical data depicted by concentric circles. The circle in the center represents the root node surrounded by the rings representing different levels of hierarchy. Rings are divided based on their relationship with the parent slice with each of them either divided equally or proportional to a value. This type of chart helps users in breaking down data into different entities for identifying and visualizing multilevel parent child relationships in different business scenarios quickly and efficiently.

### Using Code

Refer to the following code to add a Sunburst chart:

Java

```
private static void SunburstChart() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:D16")
        .setValue(new Object[][] {
            { "Region", "Subregion", "Country", "Population" },
            { "Asia", "Southern", "India", 1354051854 },
```

```
{ null, null, "Pakistan", 200813818 },
{ null, null, "Bangladesh", 166368149 },
{ null, null, "Others", 170220300 },
{ null, "Eastern", "China", 1415045928 },
{ null, null, "Japan", 127185332 },
{ null, null, "Others", 111652273 },
{ null, "South-Eastern", null, 655636576 },
{ null, "Western", null, 272298399 },
{ null, "Central", null, 71860465 },
{ "Africa", "Eastern", null, 433643132 },
{ null, "Western", null, 381980688 },
{ null, "Northern", null, 237784677 },
{ null, "Others", null, 234512021 },
{ "Europe", null, null, 742648010 },
{ "Others", null, null, 1057117703 } }));

worksheet.getRange("A:D").getColumns().autoFit();
// Add Sunburst Chart
IShape sunburstChartShape = worksheet.getShapes().addChart(ChartType.Sunburst, 250,
20, 360, 330);

// Adding series to SeriesCollection

sunburstChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D16"),
RowCol.Columns, true,
true);

// Configure Chart Title
sunburstChartShape.getChart().getChartTitle().setText("World Population");

// Saving workbook to Xlsx
workbook.save("32-SunburstChart.xlsx", SaveFileFormat.Xlsx);
```

## Treemap

**Treemap** is a chart type used to display hierarchical data as a set of nested rectangles. Treemap charts are used to represent hierarchical data in a tree-like structure. Data, organized as branches and sub-branches, is depicted with the help of rectangles. With Treemap charts, you can easily drill down huge data to an unlimited number of levels.

### Using Code

Refer to the following code to add Treemap chart:

```
Java

private static void TreemapChart() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
```

```

IWorksheet worksheet = workbook.getWorksheets().get(0);

// Prepare data for chart
worksheet.getRange("A1:D16")
    .setValue(new Object[][] {
        { "Region", "Subregion", "Country", "Population" },
        { "Asia", "Southern", "India", 1354051854 },
        { null, null, "Pakistan", 200813818 },
        { null, null, "Bangladesh", 166368149 },
        { null, null, "Others", 170220300 },
        { null, "Eastern", "China", 1415045928 },
        { null, null, "Japan", 127185332 },
        { null, null, "Others", 111652273 },
        { null, "South-Eastern", null, 655636576 },
        { null, "Western", null, 272298399 },
        { null, "Central", null, 71860465 },
        { "Africa", "Eastern", null, 433643132 },
        { null, "Western", null, 381980688 },
        { null, "Northern", null, 237784677 },
        { null, "Others", null, 234512021 },
        { "Europe", null, null, 742648010 },
        { "Others", null, null, 1057117703 } });
worksheet.getRange("A:D").getColumns().autoFit();
// Add Treemap Chart
IShape treeMapChartShape = worksheet.getShapes().addChart(ChartType.Treemap, 250,
20, 360, 330);

// Adding series to SeriesCollection
treeMapChartShape.getChart().getSeriesCollection().add(worksheet.getRange("A1:D16"),
RowCol.Columns, true,
    true);

// Configure Chart Title
treeMapChartShape.getChart().getChartTitle().setText("World Population");

// Saving workbook to Xlsx
workbook.save("33-TreemapChart.xlsx", SaveFileFormat.Xlsx);

```

## Funnel

**Funnel** charts help in visualizing sequential stages in a linear process such as order fulfillment. In such processes, each stage represents a proportion (percentage) of the total. Therefore, the chart takes the funnel shape with the first stage being the largest and each subsequent stage smaller than the predecessor. The Funnel charts can be used to represent stages in a sales process and represent the amount of potential revenue for each stage. This type of chart is useful in finding potential problem areas in an organization's sales processes. For instance, with Funnel charts, a user can plot the order fulfillment process that tracks number of orders getting across a stage.

## Using Code

Refer to the following code to add a Funnel chart:

Java

```
private static void FunnelChart() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    // Prepare data for chart
    worksheet.getRange("A1:B9")
        .setValue(new Object[][] {
            { null, "Sales" },
            { "Consultation", 140000 },
            { "Prospects", 120000 },
            { "Qualified", 100000 },
            { "Negotiations", 80000 },
            { "Prototype", 60000 },
            { "Closing", 40000 },
            { "Won", 20000 },
            { "Finalized", 10000 } });
    worksheet.getRange("A:B").getColumns().autoFit();

    // Add Funnel Chart
    IShape funnelChartshape = worksheet.getShapes().addChart(ChartType.Funnel, 300, 20,
    300, 200);
    funnelChartshape.getChart().getSeriesCollection().add(worksheet.getRange("A1:B9"));

    // Configure Chart Title
    funnelChartshape.getChart().getChartTitle().setText("Sales Pipeline");

    // Configure Axis
    IAxis axis = funnelChartshape.getChart().getAxes().item(AxisType.Category,
    AxisGroup.Primary);
    axis.setVisible(true);

    // Saving workbook to Xlsx
    workbook.save("34-FunnelChart.xlsx", SaveFileFormat.Xlsx);
}
```


## Chart Sheet

Sometimes, users find it difficult to accommodate both data and charts in the same worksheet. For this reason, GcExcel now lets users add the chart to a separate sheet, called the 'Chart sheet'. Unlike Worksheets, Chart sheets can contain only the chart. This helps avoid the usual clutter of data and embedded charts in the same worksheet. Also, using chart sheets,

users will be able to read the chart in detail and change the sheet page orientation while printing.

A Chart sheet can be created in a Workbook using the **IWorksheets.Add(SheetType.Chart)** ('**add(SheetType) Method**' in the on-line documentation) method. Further, you can add a chart to the Chart sheet by using **IShapes.AddChart** ('**addChart Method**' in the on-line documentation) method. The user may note that each chart sheet should have a chart, else it can throw an exception while saving the file.

The methods and properties associated with chart sheets in GcExcel are listed in the table below:

Methods/Properties	Description
<b>Add(SheetType.Chart)</b>	The <b>Add</b> (' <b>add Method</b> ' in the on-line documentation) method in <b>IWorksheets</b> (' <b>IWorksheets Interface</b> ' in the on-line documentation) interface has an overload with ' <b>SheetType</b> (' <b>add(SheetType) Method</b> ' in the on-line documentation)'. Hence, for adding a Chart sheet, you need to use <b>SheetType.Chart</b> (' <b>add(SheetType) Method</b> ' in the on-line documentation).
<b>AddChart</b>	<p>The <b>AddChart</b> ('<b>addChart Method</b>' in the on-line documentation) method in <b>IShapes</b> ('<b>IShapes Interface</b>' in the on-line documentation) interface adds a chart for the Chart sheet.</p> <div>  <b>Note:</b> Each Chart sheet should have a chart. Otherwise, it will throw an exception while saving the file. </div>
<b>AddShape</b>	The <b>AddShape</b> (' <b>addShape Method</b> ' in the on-line documentation) method in <b>IShape</b> (' <b>IShape Interface</b> ' in the on-line documentation) interface adds multiple shapes for the Chart sheet. User Shapes supported are shape, chart, picture, connector etc. In this case, the first chart is the main Chart, and the other shapes will be discarded when saving the file.
<b>SheetType</b>	The <b>SheetType</b> Property in <b>IWorksheet</b> (' <b>IWorksheet Interface</b> ' in the on-line documentation) interface gets the type of current sheet (Worksheet or Chart sheet).
<b>Delete</b>	The <b>Delete</b> (' <b>delete Method</b> ' in the on-line documentation) method in <b>IShape</b> (' <b>IShape Interface</b> ' in the on-line documentation) interface deletes the Chart from the Chart sheet, or deletes the user shape from the Chart.
<b>Copy</b>	The <b>Copy</b> (' <b>copy Method</b> ' in the on-line documentation) method in <b>IWorksheet</b> (' <b>IWorksheet Interface</b> ' in the on-line documentation) interface copies a new Chart sheet.
<b>Move</b>	The <b>Move</b> (' <b>move Method</b> ' in the on-line documentation) method in <b>IWorksheet</b> (' <b>IWorksheet Interface</b> ' in the on-line documentation) interface moves the chart sheet to a new location in the current workbook or a new workbook.

The following sections discuss in detail about chart sheet operations in a workbook.

### Add Chart Sheet

To add a chart sheet, refer the following code:

Java

```
private static Workbook AddChartSheet() {
    // Initialize workbook
    Workbook workbook = new Workbook();
    // Fetch default worksheet
    IWorksheet worksheet = workbook.getWorksheets().get(0);

    worksheet.getRange("A1:E5")
        .setValue(new Object[][] {
            { "Region", "Q1", "Q2", "Q3", "Q4" },
            { "North", 100, 300, 200, 600 },
            { "East", 400, 200, 500, 800 },
            { "South", 300, 500, 100, 400 },
            { "West", 400, 200, 600, 100 },
        });

    // Add a Chart Sheet
    IWorksheet chartSheet = workbook.getWorksheets().add(SheetType.Chart);

    // Add the main chart for the chart sheet
    IShape mainChart = chartSheet.getShapes().addChart(ChartType.ColumnClustered, 100,
100, 200, 200);
    mainChart.getChart().getChartTitle().setText("Sales 2018-2019");
    mainChart.getChart().getSeriesCollection().add(worksheet.getRange("A1:E5"));

    // Add a user shape for the main chart.
    IShape shape = mainChart.getChart().addShape(AutoShapeType.Rectangle, 50, 20, 100,
100);
    shape.getTextFrame().getTextRange()
        .add("This chart displays the regional quarterly sales for the year 2018-
2019");

    // Saving workbook to Xlsx
    workbook.save("381-AddChartSheet.xlsx", SaveFileFormat.Xlsx);
    return workbook;
}
```

### Copy and Move Chart Sheet

To copy and move a chart sheet, refer the following example code:

Java

```
private static void CopyMoveChartSheet() {
    Workbook workbook = AddChartSheet();

    // Add additional worksheets
    workbook.getWorksheets().add(SheetType.Worksheet);
    workbook.getWorksheets().add(SheetType.Worksheet);
}
```

```
// Access ChartSheet
IWorksheet chartSheet = workbook.getWorksheets().get(1);

// Copies the chart sheet to the end of the workbook and save it
chartSheet.copy();
// Saving workbook to Xlsx
workbook.save("382-CopyChartsheet.xlsx", SaveFileFormat.Xlsx);

// Moves the chart sheet to the end of the workbook and save it
chartSheet.move();
// Saving workbook to Xlsx
workbook.save("382-MoveChartsheet.xlsx", SaveFileFormat.Xlsx);
```

## Delete Chart Sheet

To delete a chart sheet, refer the following example code:

Java

```
private static void DeleteChartSheet() {
    Workbook workbook = AddChartSheet();

    // Access ChartSheet
    IWorksheet chartSheet = workbook.getWorksheets().get(1);

    // Deletes the chart sheet
    chartSheet.delete();

    // Saving workbook to Xlsx
    workbook.save("384-NoChartsheet.xlsx", SaveFileFormat.Xlsx);
}
```

## Table

GcExcel Java enables users to manage, manipulate and analyse relational data sets easily and quickly with the help of tables.

Basically, a table comprises rows and columns (range of cells) in a spreadsheet that can be formatted and managed by users in a worksheet. GcExcel Java supports the use of tables in worksheets by enabling users to perform different tasks on a table that help them in handling large chunks of data quickly and efficiently.

In GcExcel Java, you can use table in the following ways:

- [Create and Delete Tables](#)
- [Modify Tables](#)
- [Table Sort](#)
- [Table Filters](#)
- [Add and Delete Table Columns and Rows](#)
- [Table Style](#)



## Create and Delete Tables

In GcExcel Java, you can create and delete tables in spreadsheets using the **add** ('**add Method**' in the **on-line documentation**) method of the **ITables** ('**ITables Interface**' in the **on-line documentation**) interface and the **delete** ('**delete Method**' in the **on-line documentation**) method of the **ITable** ('**ITable Interface**' in the **on-line documentation**) interface, or simply transform a cell range into a table by specifying the existing data lying in a worksheet.

In order to create and delete tables in a worksheet, refer to the following example code.

Java

```
//Create workbook and access the worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Add first table
ITable table1 = worksheet.getTables().add(worksheet.getRange("A1:E5"), true);

// Add second table
ITable table2 = worksheet.getTables().add(worksheet.getRange("N1:R5"), true);

// Delete first table
worksheet.getTables().get(0).delete();
```

## Modify Tables

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

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

### Modify table range

You can modify the table range of your worksheet using the **resize** ('**resize Method**' in the **on-line documentation**) method of the **ITable** ('**ITable Interface**' in the **on-line documentation**) interface.

In order to modify table range, refer to the following example code.

Java

```
// Modify table range
table.resize(worksheet.getRange("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 **getHeaderRange** ('**getHeaderRange Method**' in the **on-line documentation**) method, **getDataRange** ('**getDataRange Method**' in the **on-line documentation**) method and **getTotalsRange** ('**getTotalsRange Method**' in the **on-line documentation**) method of the **ITable** ('**ITable Interface**' in the **on-line documentation**) interface.

In order to modify table areas in your worksheet, refer to the following example code.

Java

```
ITable table = worksheet.getTables().add(worksheet.getRange("A1:E5"), true);
table.setShowTotals(true);

// Populate table values

worksheet.getRange("A2").setValue(3);
worksheet.getRange("A3").setValue(4);
worksheet.getRange("A4").setValue(2);
worksheet.getRange("A5").setValue(1);
worksheet.getRange("B2").setValue(32);
worksheet.getRange("B3").setValue(41);
worksheet.getRange("B4").setValue(12);
worksheet.getRange("B5").setValue(16);
worksheet.getRange("C2").setValue(3);
worksheet.getRange("C3").setValue(4);
worksheet.getRange("C4").setValue(15);
worksheet.getRange("C5").setValue(18);

// Table second column name set to "Age".
worksheet.getTables().get(0).getHeaderRange().get(0, 1).setValue("Age");

// "Age" Column's second row's value set to 23.
worksheet.getTables().get(0).getDataRange().get(1, 1).setValue(23);

// "Age" column's total row function set to average.
worksheet.getTables().get(0).getTotalsRange().get(0, 1).setFormula("=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 **setTotalsCalculation** ('**setTotalsCalculation Method**' in the on-line documentation) method of the **ITableColumn** ('**ITableColumn Interface**' in the on-line documentation) interface.

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

Java

```
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getTables().add(worksheet.getRange("A1:E5"), true);

// Populate table values

worksheet.getRange("A2").setValue(3);
worksheet.getRange("A3").setValue(4);
worksheet.getRange("A4").setValue(2);
worksheet.getRange("A5").setValue(1);
worksheet.getRange("B2").setValue(32);
worksheet.getRange("B3").setValue(41);
worksheet.getRange("B4").setValue(12);
worksheet.getRange("B5").setValue(16);
```

```
worksheet.getRange("C2").setValue(3);
worksheet.getRange("C3").setValue(4);
worksheet.getRange("C4").setValue(15);
worksheet.getRange("C5").setValue(18);

// First table column's total row calculation fuction will be "=SUBTOTAL(101,[Column1])"
worksheet.getTables().get(0).getColumns().get(0).setTotalsCalculation(TotalsCalculation.Average);
worksheet.getTables().get(0).setShowTotals(true);
```

## Table Sort

With GcExcel Java, you can choose to apply sorting on a specific table in the worksheet. For executing the sort operation, you can use the **getSort** ('**getSort Method**' in the on-line documentation) method of the **ITable** ('**ITable Interface**' in the on-line documentation) interface.

The apply method is used to apply the selected sort state and display the results. In order to apply table sorting in a worksheet, refer to the following example code.

### Java

```
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:E5"), true);

// Assign values to range
worksheet.getRange("A2").setValue(3);
worksheet.getRange("A3").setValue(4);
worksheet.getRange("A4").setValue(2);
worksheet.getRange("A5").setValue(1);
worksheet.getRange("B2").setValue(1);
worksheet.getRange("B3").setValue(2);
worksheet.getRange("B4").setValue(3);
worksheet.getRange("B5").setValue(4);
worksheet.getRange("F2").setValue("aaa");
worksheet.getRange("F3").setValue("bbb");
worksheet.getRange("F4").setValue("ccc");
worksheet.getRange("F5").setValue("ddd");

worksheet.getRange("B2:B5").getFormatConditions().addIconSetCondition();

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

table.getSort().getSortFields().add(key1);
table.getSort().getSortFields().add(key2);
table.getSort().apply();
```

## Table Filters

While dealing with bulk data in spreadsheets, you can create as many tables on a worksheet as you want and apply separate filters on columns of each of the table to manage essential information in an effective manner.

Using GcExcel Java, you can apply table filters while setting up worksheets for ensuring improved and quick data analysis.

To apply filters on tables in worksheets created, you need to first get the table range and then use the **autoFilter** ('**autoFilter Method**' in the on-line documentation) method of the **IRange** ('**IRange Interface**' in the on-line documentation) interface to filter the table.

In order to set table filters in a worksheet, refer to the following example code.

Java

```
//Add table
ITable table = worksheet.getTables().add(worksheet.getRange("A1:E5"), true);
System.out.println(table);

//Assign values to the range
worksheet.getRange("A2").setValue(3);
worksheet.getRange("A3").setValue(4);
worksheet.getRange("A4").setValue(2);
worksheet.getRange("A5").setValue(1);

//Apply table filter
worksheet.getTables().get(0).getRange().autoFilter(0, ">2");
```

## Add and Delete Table Columns and Rows

GcExcel Java provides users with the ability to insert and delete columns and rows of a table using the methods of the following interfaces:

- **ITableColumns** ('**ITableColumns Interface**' in the on-line documentation) interface - Represents the collection of columns in a table.
- **ITableRows** ('**ITableRows Interface**' in the on-line documentation) interface - Represents the collection of rows in a table.
- **ITableColumn** ('**ITableColumn Interface**' in the on-line documentation) interface - Represents a specific column in a table.
- **ITableRow** ('**ITableRow Interface**' in the on-line documentation) interface - Represents a specific row in a table.

### Add and delete table columns

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

In order to add and delete table columns, refer to the following example code.

Java

```
//Create first table
ITable table = worksheet.getTables().add(worksheet.getRange("D3:I6"), true);

// Insert a table column before first column.
table.getColumns().add(0);

// Insert a table column before second column.
table.getColumns().add(1);

// Delete the first table column.
worksheet.getTables().get(0).getColumns().get(0).delete();

// Delete the second table column.
worksheet.getTables().get(0).getColumns().get(1).delete();
```

### Add and delete table rows

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

In order to add and delete table rows, refer to the following example code.

Java

```
//Create first table
ITable table = worksheet.getTables().get(0);

// insert table row in last row.
table.getRows().add();

// delete second table row.
table.getRows().get(1).delete();
```

## Table Style

GcExcel Java enables users to create custom table style elements and apply them to a worksheet using the **ITableStyle ('ITableStyle Interface' in the on-line documentation)** interface. Also, users can format a table using any of the predefined table styles as per their preferences.

Generally, each workbook stores both built-in and custom table styles. If you want to insert a custom table style, you use the **add ('add Method' in the on-line documentation)** method of the **ITables ('ITables Interface' in the on-line documentation)** interface, which returns the **IStyle** object representing the corresponding table style instance.

In order to apply table style in a worksheet, refer to the following example code.

Java

```
// Use table style name get one build in table style.
ITableStyle tableStyle = workbook.getTableStyles().get("TableStyleLight11");
worksheet.getTables().add(worksheet.getRange(0, 0, 2, 2), true);

// Set built-in table style to table.
worksheet.getTables().get(0).setTableStyle(tableStyle);
```

## Modify Table with Custom Style

GcExcel Java allows users to modify tables with custom style.

By default, a workbook possesses a collection of built-in table styles to enables users to apply formatting to tables. These default table styles represent that no formatting is applied to the tables. However, when a custom table style is created, it automatically gets added to the table style collection of a workbook and can be reused as and when required.

While managing one or more table styles in your workbook, users can modify the built-in table style with custom table style. Each table style element represents the formatting for a particular element of the table. When a custom style for a table is defined, users first need to access the existing table style element in order to customize the table borders, set custom fill for the table, style row stripes or column stripes etc.

In order to change the table style, you can use the `setTableStyle` method. In case you want to delete the applied table style, you can use the `delete` method.

Refer to the following example code to modify table with custom style.

### Java

```
// Add one custom table style.
ITableStyle style = workbook.getTableStyles().add("test");

// Set WholeTable element style.
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getFont().setItalic(true);
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getFont().setThemeColor(ThemeColor.Accent6);
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getFont().setStrikethrough(true);
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getBorders().setLineStyle(BorderLineStyle.Dotted);
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getBorders().setThemeColor(ThemeColor.Accent2);
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getInterior().setColor(Color.FromArgb(24, 232, 192));

// Set FirstColumnStripe element style.
style.getTableStyleElements().get(TableStyleElementType.FirstColumnStripe).getFont().setBold(true);
style.getTableStyleElements().get(TableStyleElementType.FirstColumnStripe).getFont().setColor(Color.FromArgb(255, 0, 0));
style.getTableStyleElements().get(TableStyleElementType.FirstColumnStripe).getBorders().setLineStyle(BorderLineStyle.Thick);
style.getTableStyleElements().get(TableStyleElementType.FirstColumnStripe).getBorders().setThemeColor(ThemeColor.Accent5);
style.getTableStyleElements().get(TableStyleElementType.FirstColumnStripe).getInterior().setColor(Color.FromArgb(255, 255, 0));
style.getTableStyleElements().get(TableStyleElementType.FirstColumnStripe).setStripeSize(2);

// Set SecondColumnStripe element style.
style.getTableStyleElements().get(TableStyleElementType.SecondColumnStripe).getFont().setColor(Color.FromArgb(255, 0, 255));
style.getTableStyleElements().get(TableStyleElementType.SecondColumnStripe).getBorders().setLineStyle(BorderLineStyle.DashDot);
style.getTableStyleElements().get(TableStyleElementType.SecondColumnStripe).getBorders().setColor(Color.FromArgb(42, 105, 162));
style.getTableStyleElements().get(TableStyleElementType.SecondColumnStripe).getInterior().setColor(Color.FromArgb(204, 204, 255));

ITable table = worksheet.getTables().get(0);

// Set custom table style to table.
table.setTableStyle(style);
table.setShowTableStyleColumnStripes(true);
```

## Modify Table Layout

GcExcel Java provides users with the ability to modify table layout as per their choice.

Table Layout mode enables users to divide an area of a group into several rows and columns and then place controls into the 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.

In order to modify table layout in GcExcel Java, refer to the following example code.

## Java

```
ITable table = worksheet.getTables().add(worksheet.getRange("A1:B2"));

// Show table header row.
table.setShowHeaders(true);

// To make "first row stripe" and "second row stripe" table style element's
// style effective.
table.setShowTableStyleRowStripes(false);

// Hide auto filter drop down button.
table.setShowAutoFilterDropDown(false);

// To make "first column" table style element's style effective.
table.setShowTableStyleFirstColumn(true);

// Show table total row.
table.setShowTotals(true);

// To make "last column" table style element's style effective.
table.setShowTableStyleLastColumn(true);

// To make "first column stripe" and "second column stripe" table style
// element's style effective.
table.setShowTableStyleColumnStripes(true);

// Unfilter table column filters, and hide auto filter drop down button.
table.setShowAutoFilter(false);
```

## Pivot Table

GcExcel Java allows users to display aggregated data in a spreadsheet using pivot tables.

A pivot table is a data summarization tool that can perform complex information analytics for the data stored in cells. This facilitates users to explore, analyze and manipulate bulk data in a worksheet.

Inserting pivot tables not only helps in categorizing data in a worksheet but also helps in computing the totals and average of the values in the cells based on the summary functions defined in the built-in functions list.

In order to work with pivot tables in spreadsheets, refer to the the following tasks:

- [Create Pivot Table](#)
- [Pivot Table Settings](#)
- [Pivot Table Style](#)

## Create Pivot Table

GcExcel Java enables users to create pivot tables in a worksheet. The **IPivotCache** (**'IPivotCache Interface' in the on-line documentation**) and the **IPivotCaches** (**'IPivotCaches Interface' in the on-line documentation**) interface stores all the pivot caches in the workbook.

You can create pivot tables in worksheets using any of the following ways:

- Use the **createPivotTable** (**'createPivotTable Method' in the on-line documentation**) method of the **IPivotCache** (**'IPivotCache Interface' in the on-line documentation**) interface.
- Use the **add** (**'add Method' in the on-line documentation**) method of the **IPivotTables** (**'IPivotTables Interface' in the on-line documentation**) interface.

In order to create pivot table in a worksheet using the add method, refer to the following example code.

Java

```
// Source data for PivotCache
Object sourceData = new Object[][]
{
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United States" },
    { 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United Kingdom" },
    { 3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States" },
    { 4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada" },
    { 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany" },
    { 6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United States" },
    { 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11), "Australia" },
    { 8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand" },
    { 9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France" },
    { 10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada" },
    { 11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany" },
    { 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18), "United States" },
    { 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20), "Germany" },
    { 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22), "Canada" },
    { 15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France" },
};

// Initialize the workbook and fetch the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
```



```
// Assigning data to the range
worksheet.getRange("A1:F16").setValue(sourceData);
worksheet.getRange("A:F").setColumnWidth(15);

// Creating pivot
IPivotCache pivotcache = workbook.getPivotCaches().create(worksheet.getRange("A1:F16"));
IPivotTable pivottable = worksheet.getPivotTables().add(pivotcache,
worksheet.getRange("H7"), "pivottable1");
worksheet.getRange("D2:D16").setNumberFormat("$#,##0.00");
worksheet.getRange("I9:O11").setNumberFormat("$#,##0.00");
worksheet.getRange("H:O").setColumnWidth(12);
```

## Pivot Table Settings

You can modify the settings of the pivot table added in a worksheet by referring to the following tasks:

- **Configure Pivot table fields**
- **Add field function**
- **Filter Pivot table**
- **Manage Pivot field level**
- **Manage Grand Total Visibility Settings**
- **Change Row Axis Layout**
- **Change Pivot Table Layout**
- **Rename Pivot Table Fields**
- **Refresh Pivot table**
- **Modify Pivot table**
- **Apply Different Calculations on a Pivot Field**
- **Defer Layout Update**
- **Use Pivot Table Options**
- **Sort Pivot Table Fields**
- **Retrieve Pivot Table Ranges**
- **Set Conditional Formatting**

### Configure pivot table fields

The fields of a pivot table can be configured using the methods of the **IPivotCaches** ('**IPivotCaches Interface**' in the **on-line documentation**) interface and **IPivotTables** ('**IPivotTables Interface**' in the **on-line documentation**) interface, as shown in the example code shared below.

Java

```
// config pivot table's fields
IPivotField field_Category = pivottable.getPivotFields().get("Category");
field_Category.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Product = pivottable.getPivotFields().get("Product");
field_Product.setOrientation(PivotFieldOrientation.ColumnField);
```

```
IPivotField field_Amount = pivottable.getPivotFields().get("Amount");
field_Amount.setOrientation(PivotFieldOrientation.DataField);

IPivotField field_Country = pivottable.getPivotFields().get("Country");
field_Country.setOrientation(PivotFieldOrientation.PageField);
```

### Add field function

In order to add field function in a pivot table, refer to the following example code.

#### Java

```
// Change or set data field's summarize function.
field_Amount.setFunction(ConsolidationFunction.Average);
```

### Filter Pivot Table

In order to execute the filter operation on a pivot table, refer to the following example code.

#### Java

```
IPivotField field_Product = pivottable.getPivotFields().get(1);
field_Product.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Amount = pivottable.getPivotFields().get(3);
field_Amount.setOrientation(PivotFieldOrientation.DataField);

IPivotField field_Country = pivottable.getPivotFields().get(5);
field_Country.setOrientation(PivotFieldOrientation.PageField);

// Apply row field filter.
field_Product.getPivotItems().get("Apple").setVisible(false);
field_Product.getPivotItems().get("Beans").setVisible(false);
field_Product.getPivotItems().get("Orange").setVisible(false);

// Apply page filter.
field_Country.getPivotItems().get("United States").setVisible(false);
field_Country.getPivotItems().get("Canada").setVisible(false);
```

### Manage Pivot Field Level

In order to manage the field level of a pivot table, refer to the following example code.

#### Java

```
// Product in level 1.
IPivotField field_product = pivottable.getPivotFields().get("Product");
field_product.setOrientation(PivotFieldOrientation.RowField);

// Category in level 2.
```

```
IPivotField field_category = pivottable.getPivotFields().get("Category");
field_category.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Amount = pivottable.getPivotFields().get(3);
field_Amount.setOrientation(PivotFieldOrientation.DataField);

// Category will be in level 1 and product will be in level 2.
field_product.setPosition(1);
field_category.setPosition(0);
```

### Manage Grand Total Visibility Settings

The Grand total in pivot table helps in analyzing the total sum of the data in the pivot table. You can display or hide the grand total for the row or column field by setting the visibility of **ColumnGrand** and **RowGrand** properties of the **IPivotTable** interface. These properties take boolean values and are set to true by default. For example, if you want to display the grand total only for rows, then set the RowGrand method to true and ColumnGrand to false.

Refer to the following example code to manage the visibility settings of the grand total field.

Java

```
// Set the PivotTable report to show grand totals for columns & rows
pivottable.setColumnGrand(true);
pivottable.setRowGrand(true);
```

### Change Row Axis Layout

The display of pivot table can be changed to any desired layout using the **LayoutRowType** enumeration. The following options are provided by this enumeration:

- **CompactRow** (default layout)
- **OutlineRow**
- **TabularRow**



**Note:** The **SubtotalLocationType** enumeration can only be set to **Bottom** if the **LayoutRowType** is set to **TabularRow**.

Refer to the following example code to set the row axis layout of the pivot table to TabularRow.

Java

```
// Set the PivotTable LayoutRowType to Tabular Row
pivottable.setRowAxisLayout(LayoutRowType.TabularRow);
```

### Change Pivot Table Layout

The different layouts of a pivot table makes it more flexible and convenient to analyse its data. GcExcel supports the following pivot table layouts:

- Compact form
- Outline form
- Tabular form

In addition to these, you can also choose to insert blank rows, set the position of subtotals, show all items or to repeat any item in the pivot table layouts.

Refer to the following example code to set the layout of pivot table and additional options.

#### Java

```
// Set pivot table layout
field_Category.setLayoutForm(LayoutFormType.Tabular);
field_Category.setLayoutBlankLine(true);

field_Country.setLayoutForm(LayoutFormType.Outline);
field_Country.setLayoutCompactRow(false);

// Set subtotal location
field_Country.setLayoutSubtotalLocation(SubtotalLocationType.Bottom);
field_Country.setShowAllItems(true);
```

### Rename Pivot Table Fields

Sometimes, the pivot table fields are not easily comprehensible and hence can be renamed to meaningful and easily understandable names.

Refer to the following example code to rename the pivot table fields.

#### Java

```
// config pivot table's fields
IPivotField field_Date = pivottable.getPivotFields().get("Date");
field_Date.setOrientation(PivotFieldOrientation.PageField);

IPivotField field_Category = pivottable.getPivotFields().get("Category");
field_Category.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Product = pivottable.getPivotFields().get("Product");
field_Product.setOrientation(PivotFieldOrientation.ColumnField);

IPivotField field_Amount = pivottable.getPivotFields().get("Amount");
field_Amount.setOrientation(PivotFieldOrientation.DataField);
field_Amount.setNumberFormat("$#,##0.00");

IPivotField field_Country = pivottable.getPivotFields().get("Country");
field_Country.setOrientation(PivotFieldOrientation.RowField);

// Renaming DataField "Sum of Amount" to "Amount Total"
pivottable.getDataFields().get(0).setName("Amount Total");
```

### Refresh Pivot Table

In order to refresh a pivot table, refer to the following example code.

## Java

```
IPivotField field_Product = pivottable.getPivotFields().get("Product");
field_Product.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Amount = pivottable.getPivotFields().get(3);
field_Amount.setOrientation(PivotFieldOrientation.DataField);

// change pivot cache's source data.
worksheet.getRange("D8").setValue(3000);

// sync cache's data to pivot table.
worksheet.getPivotTables().get(0).refresh();
```

## Modify Pivot Table

In order to modify a pivot table, refer to the following example code.

## Java

```
// Defining source data
Object sourceData = new Object[][]
{
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United States" },
    { 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United Kingdom"
},
    { 3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States" },
    { 4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada" },
    { 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany" },
    { 6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United States" },
    { 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11), "Australia" },
    { 8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand" },
    { 9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France" },
    { 10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada" },
    { 11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany" },
    { 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18), "United States"
},
    { 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20), "Germany" },
    { 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22), "Canada" },
    { 15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France" },
};

// Initialize workbook and fetch the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Assigning data to the range
worksheet.getRange("A1:F16").setValue(sourceData);
```

```
// Creating pivot table and modifying it
IPivotCache pivotcache = workbook.getPivotCaches().create(worksheet.getRange("A1:F16"));
IPivotTable pivottable = worksheet.getPivotTables().add(pivotcache,
worksheet.getRange("I2"), "pivottable1");

worksheet.getRange("D2:D16").setNumberFormat("$#,##0.00");
worksheet.getRange("J4:J17, J9:J33").setNumberFormat("$#,##0.00");

// Configure pivot table's fields
IPivotField field_Product = pivottable.getPivotFields().get(1);
field_Product.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Category = pivottable.getPivotFields().get(2);
field_Category.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Amount = pivottable.getPivotFields().get(3);
field_Amount.setOrientation(PivotFieldOrientation.DataField);

// Modify subtotals for pivot field.
field_Category.setSubtotals(EnumSet.of(SubtotalType.Sum, SubtotalType.Count,
SubtotalType.Average, SubtotalType.Max, SubtotalType.Min, SubtotalType.CountNums,
SubtotalType.StdDev, SubtotalType.StdDevP, SubtotalType.Var, SubtotalType.VarP));

worksheet.getRange("E:E").setColumnWidth(12);
worksheet.getRange("J:J").setColumnWidth(20);
```

### Apply Different Calculations on a Pivot Field

In GcExcel, you can add a pivot table field to a pivot table multiple times by applying various calculation functions on it. These functions include sum, average, min, max, count etc. The final pivot table output will contain multiple data fields based on the calculations applied over the pivot table field.

Refer to the following example code to add a pivot table field as multiple data fields by applying different calculation functions.

#### Java

```
// Config pivot table's fields
IPivotField field_Category = pivottable.getPivotFields().get("Category");
field_Category.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Product = pivottable.getPivotFields().get("Product");
field_Product.setOrientation(PivotFieldOrientation.RowField);

// Sum function on Amount field
IPivotField field_Amount = pivottable.getPivotFields().get("Amount");
pivottable.addDataField(field_Amount, "sum amount", ConsolidationFunction.Sum);

// Count function on Amount field
IPivotField field_Amount2 = pivottable.getPivotFields().get("Amount");
```

```
pivottable.addDataField(field_Amount2, "count amount", ConsolidationFunction.Count);
```

The output of above example code when viewed in Excel, looks like below:

Row Labels	sum amount	count amount
<b>Fruit</b>	<b>44561</b>	<b>8</b>
Apple	16794	3
Banana	24157	4
Orange	3610	1
<b>Vegetables</b>	<b>35936</b>	<b>7</b>
Beans	2626	1
Broccoli	27137	4
Carrots	6173	2
<b>Grand Total</b>	<b>80497</b>	<b>15</b>
		count amount
		Value: 15
		Row: Grand Total
		Column: count amount

### Defer Layout Update

In case of huge amount of data, the performance of a pivot table might get affected while updating its layout by adding or moving fields in the different areas of a pivot table.

GcExcel provides **setDeferLayoutUpdate** method which improves the performance of a pivot table by deferring its layout updates. When set to true, the pivot table is recalculated only after all the fields are added or moved instead of getting recalculated after each change. You can choose to update the pivot table output after making all the changes by calling the **update** method.

Refer to the following example code to defer layout updates to a pivot table.

Java

```
// Defer layout update
pivottable.setDeferLayoutUpdate(true);

// Config pivot table's fields
IPivotField field_Category = pivottable.getPivotFields().get("Category");
field_Category.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Product = pivottable.getPivotFields().get("Product");
field_Product.setOrientation(PivotFieldOrientation.ColumnField);

IPivotField field_Amount = pivottable.getPivotFields().get("Amount");
field_Amount.setOrientation(PivotFieldOrientation.DataField);

// Update the pivottable.
pivottable.update();
```

### Use Pivot Table Options

GcExcel supports the following layout and formatting options in a pivot table:

- Merging cells with outer-row item, column item, subtotal and grand total labels
- Indentation of Pivot table items when compact row layout form is set
- Ordering page fields in pivot table layout. It can be either **DownThenOver** (default value) or **OverThenDown**.
- Defining number of page fields in each column or row in the pivot table output
- Displaying custom string in cells which contain errors
- Displaying custom string in cells which contain null values

Refer to the following example code to set various layout and format options in a pivot table.

Java

```
pivottable.setPageFieldOrder(Order.OverThenDown);
pivottable.setPageFieldWrapCount(2);
pivottable.setCompactRowIndent(2);

pivottable.setErrorString("Error");
pivottable.setNullString("Empty");

pivottable.setDisplayErrorString(true);
pivottable.setDisplayNullString(true);
```

### Sort Pivot Table Fields

GcExcel supports sorting data fields in a pivot table by using **autoSort** method and defining ascending or descending as its sort order.

You can also retrieve the name of data field used to sort the specified PivotTable field by using **autoSortField** method and its sorting order by using **autoSortOrder** method. The position of an item in its field can also be set or retrieved by using the **setPosition** or **getPosition** method of **IPivotItem** interface.

Refer to the following example code to sort 'Product' field in a pivot table.

Java

```
// Sort the product items
field_Product.autoSort(SortOrder.Descending);
```

### Retrieve Pivot Table Ranges

The structure of a pivot table report is comprised of different ranges. In order to retrieve a specific range of pivot table, it is important to understand the structure of a pivot table.



Sum of Amount	Column Labels		
Row Labels	2018 Qtr1 Jan	2019 Qtr1 Jan	Grand Total
Consumer Electronics	\$28,515.00	\$10,904.00	\$39,419.00
Bose 785593-0050	\$4,270.00	\$1,903.00	\$6,173.00
Canon EOS 1500D	\$11,063.00		\$11,063.00
Haier 394L 4Star	\$6,946.00	\$617.00	\$7,563.00
IFB 6.5 Kg FullyAuto		\$8,384.00	\$8,384.00
Mi LED 40inch	\$2,626.00		\$2,626.00
Sennheiser HD 4.40-BT	\$3,610.00		\$3,610.00
Mobile	\$32,016.00	\$9,062.00	\$41,078.00
Iphone XR		\$9,062.00	\$9,062.00
OnePlus 7Pro	\$15,156.00		\$15,156.00
Redmi 7	\$9,429.00		\$9,429.00
Samsung S9	\$7,431.00		\$7,431.00
Grand Total	\$60,531.00	\$19,966.00	\$80,497.00

As can be observed from the above screenshot, the structure of a pivot table can be explained as:

- **Pivot Row Axis:** The row axis area of a pivot table contains fields which group the table's data by rows
- **Pivot Column Axis:** The column axis area of a pivot table contains fields which break the table's data into different categories by columns.
- **Pivot Cell:** Any cell in a pivot table
- **Row PivotLine:** Any row in the row axis area of a pivot table
- **Column PivotLine:** Any column in the column axis area of a pivot table

GcExcel provides API to retrieve the detailed ranges of a pivot table to apply any operation or style on them to make the result more readable and distinguishable. Detailed pivot table ranges which can be retrieved are:

- Different types of pivot cells like subtotals, grand totals, data fields, pivot fields, values, blank cells
- Different types of pivot lines like subtotal, grand total, regular or blank line
- Entire row or column axis
- Whole page area
- Entire pivot table report including page fields
- A value in any range of pivot table
- The position of any element or pivot line

Refer to the following example code to get a specific range and set its style in a pivot table report.

Java

```
// Get detail range and set style.
for (IPivotLine item : pivottable.getPivotRowAxis().getPivotLines()) {
    if (item.getLineType() == PivotLineType.Subtotal) {


item.getPivotLineCells().get(0).getRange().getInterior().setColor(Color.GetGreenYellow());
    }
}
```

```
}

```

The output of above code example when viewed in Excel, looks like below:

Row Labels	Sum of Amount
<b>Consumer Electronics</b>	
Bose 785593-0050	\$6,173.00
Canon EOS 1500D	\$11,063.00
Haier 394L 4Star	\$7,563.00
IFB 6.5 Kg FullyAuto	\$8,384.00
Mi LED 40inch	\$2,626.00
Sennheiser HD 4.40-BT	\$3,610.00
<b>Consumer Electronics Total</b>	<b>\$39,419.00</b>
<b>Mobile</b>	
Iphone XR	\$9,062.00
OnePlus 7Pro	\$15,156.00
Redmi 7	\$9,429.00
Samsung S9	\$7,431.00
<b>Mobile Total</b>	<b>\$41,078.00</b>
<b>Grand Total</b>	<b>\$80,497.00</b>

 **Note:** Style applied to a pivot table is lost if the pivot table is changed in any way.


## Set Conditional Formatting

Refer to the following example code to set conditional formatting in last row of a pivot table report by setting cell color when the values are above average.

Java

```
// set conditional format to the last row
int rowCount = pivottable.getDataBodyRange().getRowCount();
IAboveAverage averageCondition = pivottable.getDataBodyRange().getRows().get(rowCount - 1).getFormatConditions().addAboveAverage();
averageCondition.setAboveBelow(AboveBelow.AboveAverage);
averageCondition.getInterior().setColor(Color.GetPink());

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

 **Note:** Conditional formatting applied to a pivot table is lost if the pivot table is changed in any way.

## Pivot Table Style

GcExcel Java allows users to apply built-in and custom styles to the pivot table.

With the help of this feature, users will be able to save pivot tables with different styles (with respect to the pivot table layout and pivot table fields). Users can customize how their pivot table is displayed including the pivot table's orientation, page size, pivot table fields and many other characteristics as per their custom display preferences. Further, users can also refer to the topic [Export Pivot Table Styles and Format](#) in order export spreadsheets with different pivot table styles in PDF format.

Usually, when users add a pivot table to the worksheet, a default pivot table style is applied automatically. Users can modify the default style of the pivot table added to the worksheet by either copying an existing style (also called built-in style) or creating a custom pivot table style right from the scratch. In order to apply style to the pivot table, you can refer to the following sections:

- **Apply Built-In Pivot Table Style**
- **Apply Custom Style**

### Apply Built-In Pivot Table Style

You can change the default appearance of the pivot table by applying any of the built-in styles. In order to apply built-in style to the pivot table, users can either use the **setStyle()** ('**setStyle Method**' in the **on-line documentation**) method or use the **setTableStyle()** ('**setTableStyle Method**' in the **on-line documentation**) method of the **IPivotTable** ('**IPivotTable Interface**' in the **on-line documentation**) interface.

The image shared below depicts a pivot table with built-in style.

Date	(All)						
Sum of Amount	Column Labels	Banana	Beans	Broccoli	Carrots	Orange	Grand Total
Row Labels	Apple						
<b>Fruit</b>	<b>\$9,848.00</b>	<b>\$24,157.00</b>				<b>\$3,610.00</b>	<b>\$37,615.00</b>
Canada	\$7,431.00	\$8,384.00					\$15,815.00
France	\$2,417.00						\$2,417.00
Germany		\$8,250.00					\$8,250.00
New Zealand		\$6,906.00					\$6,906.00
United States		\$617.00				\$3,610.00	\$4,227.00
<b>Vegetables</b>			<b>\$2,626.00</b>	<b>\$24,313.00</b>	<b>\$6,173.00</b>		<b>\$33,112.00</b>
Australia				\$9,062.00			\$9,062.00
Germany			\$2,626.00		\$1,903.00		\$4,529.00
United Kingdom				\$8,239.00			\$8,239.00
United States				\$7,012.00	\$4,270.00		\$11,282.00
<b>Grand Total</b>	<b>\$9,848.00</b>	<b>\$24,157.00</b>	<b>\$2,626.00</b>	<b>\$24,313.00</b>	<b>\$6,173.00</b>	<b>\$3,610.00</b>	<b>\$70,727.00</b>

Refer to the following example code in order to apply built-in style to the pivot table.

Java

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

// Fetch default worksheet
```

```
IWorksheet worksheet = workbook.getWorksheets().get(0);

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

worksheet.getRange("A20:F33").setValue(sourceData);
worksheet.getRange("A:F").setColumnWidth(10);

// Add pivot table
IPivotCache pivotcache =
workbook.getPivotCaches().create(worksheet.getRange("A20:F33"));
IPivotTable pivottable =
worksheet.getPivotTables().add(pivotcache, worksheet
.getRange("A1"), "pivottable1");

// Setting number format for a field
worksheet.getRange("D21:D35").setNumberFormat("$#,##0.00");

// Configure pivot table's fields
IPivotField field_Date =
pivottable.getPivotFields().get("Date");
field_Date.setOrientation(PivotFieldOrientation.PageField);

IPivotField field_Category =
pivottable.getPivotFields().get("Category");
field_Category.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Product =
pivottable.getPivotFields().get("Product");
```

```
field_Product.setOrientation(PivotFieldOrientation.ColumnField);

IPivotField field_Amount =
pivottable.getPivotFields().get("Amount");
field_Amount.setOrientation(PivotFieldOrientation.DataField);
field_Amount.setNumberFormat("$#,##0.00");

IPivotField field_Country =
pivottable.getPivotFields().get("Country");
field_Country.setOrientation(PivotFieldOrientation.RowField);

// Set pivot style
pivottable.setTableStyle("PivotStyleMedium20");

worksheet.getPageSetup().setTopMargin(30);
worksheet.getPageSetup().setLeftMargin(30);

worksheet.getRange("A1:H16").getColumns().autoFit();

// Saving workbook to PDF
workbook.save("PivotBuiltInStyle.pdf", SaveFileFormat.Pdf);
```



**Note:** While applying built-in styles to the pivot table, it is important to note that if users apply a TableStyle whose **setShowAsAvailableTableStyle** ('**setShowAsAvailableTableStyle Method**' in the on-line **documentation**) method is true, then the `InvalidOperationException` is thrown.

### Apply Custom Style

If you don't want to apply any of the built-in styles, you can also create and apply your own custom style to the pivot table. This can be done using the **setStyle()** ('**setStyle Method**' in the on-line **documentation**) method of the **IPivotTable** ('**IPivotTable Interface**' in the on-line **documentation**) interface.

The image shared below depicts a pivot table with custom style.

Date	(All)					
Sum of Amount	Column Labels					
Row Labels	Apple	Banana	Beans	Broccoli	Carrots	Orange
Fruit	\$9,848.00	\$24,157.00				\$3,610.00
Canada	\$7,431.00	\$8,384.00				
France	\$2,417.00					
Germany		\$8,250.00				
New Zealand		\$6,906.00				
United States		\$617.00				\$3,610.00
Vegetables			\$2,626.00	\$24,313.00	\$6,173.00	
Australia				\$9,062.00		
Germany			\$2,626.00		\$1,903.00	
United Kingdom				\$8,239.00		
United States				\$7,012.00	\$4,270.00	
Grand Total	\$9,848.00	\$24,157.00	\$2,626.00	\$24,313.00	\$6,173.00	\$3,610.00
						\$70,727.00

Refer to the following example code in order to apply custom style to the pivot table.

#### Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Create PivotTable
Object sourceData = new Object[][] {
{ "Order ID", "Product", "Category", "Amount", "Date", "Country" },
{ 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2012, 1, 6), "United States" },
{ 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2012, 1, 7), "United Kingdom" },
{ 3, "Banana", "Fruit", 617, new GregorianCalendar(2012, 1, 8), "United States" },
{ 4, "Banana", "Fruit", 8384, new GregorianCalendar(2012, 1, 10), "Canada" },
{ 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2012, 1, 10), "Germany" },
{ 6, "Orange", "Fruit", 3610, new GregorianCalendar(2012, 1, 11), "United States" },
{ 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2012, 1, 11), "Australia" },
{ 8, "Banana", "Fruit", 6906, new GregorianCalendar(2012, 1, 16), "New Zealand" },
{ 9, "Apple", "Fruit", 2417, new GregorianCalendar(2012, 1, 16), "France" },
{ 10, "Apple", "Fruit", 7431, new GregorianCalendar(2012, 1, 16), "Canada" },
{ 11, "Banana", "Fruit", 8250, new GregorianCalendar(2012, 1, 16), "Germany" },
{ 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2012, 1, 18), "United States" },
{ 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2012, 1, 20), "Germany" },
```

```
{ 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2012, 1, 22), "Canada" },
{ 15, "Apple", "Fruit", 6946, new GregorianCalendar(2012, 1, 24), "France" }, };

worksheet.getRange("A20:F33").setValue(sourceData);
worksheet.getRange("A:F").setColumnWidth(10);

// Add pivot table
IPivotCache pivotcache =
workbook.getPivotCaches().create(worksheet.getRange("A20:F33"));
IPivotTable pivottable =
worksheet.getPivotTables().add(pivotcache, worksheet.getRange("A1"), "pivottable1");

// Setting number format for a field
worksheet.getRange("D21:D35").setNumberFormat("$#,##0.00");

// Configure pivot table's fields
IPivotField field_Date = pivottable.getPivotFields().get("Date");
field_Date.setOrientation(PivotFieldOrientation.PageField);

IPivotField field_Category = pivottable.getPivotFields().get("Category");
field_Category.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Product = pivottable.getPivotFields().get("Product");
field_Product.setOrientation(PivotFieldOrientation.ColumnField);

IPivotField field_Amount = pivottable.getPivotFields().get("Amount");
field_Amount.setOrientation(PivotFieldOrientation.DataField);
field_Amount.setNumberFormat("$#,##0.00");

IPivotField field_Country = pivottable.getPivotFields().get("Country");
field_Country.setOrientation(PivotFieldOrientation.RowField);

// Create pivot style with name "CustomPivotstyle"
ITableStyle pivotStyle = workbook.getTableStyles().add("CustomPivotstyle");

// Set table style as pivot table style
pivotStyle.setShowAsAvailablePivotStyle(true);

pivotStyle.getTableStyleElements()
    .get(TableStyleElementType.PageFieldLabels).getInterior()
    .setColor(com.grapecity.documents.excel.Color.GetLightGreen());
pivotStyle.getTableStyleElements()
    .get(TableStyleElementType.PageFieldValues).getInterior()
    .setColor(com.grapecity.documents.excel.Color.GetLightGreen());

pivotStyle.getTableStyleElements()
    .get(TableStyleElementType.GrandTotalColumn).getInterior()
    .setColor(com.grapecity.documents.excel.Color.GetPowderBlue());
```

```
pivotStyle.getTableStyleElements()  
.get(TableStyleElementType.GrandTotalRow).getInterior()  
.setColor(com.grapecity.documents.excel.Color.GetPowderBlue());  
  
pivotStyle.getTableStyleElements()  
.get(TableStyleElementType.HeaderRow).getInterior()  
.setColor(com.grapecity.documents.excel.Color.GetMistyRose());  
pivotStyle.getTableStyleElements()  
.get(TableStyleElementType.FirstColumn).getInterior()  
.setColor(com.grapecity.documents.excel.Color.GetLightPink());  
  
pivotStyle.getTableStyleElements()  
.get(TableStyleElementType.FirstRowStripe).getInterior()  
.setColor(com.grapecity.documents.excel.Color.GetSteelBlue());  
pivotStyle.getTableStyleElements()  
.get(TableStyleElementType.SecondRowStripe).getInterior()  
.setColor(com.grapecity.documents.excel.Color.GetNavajoWhite());  
  
// Set ShowTableStyleRowStripes as true  
pivottable.setShowTableStyleRowStripes(true);  
  
// Set pivot table style  
pivottable.setStyle(pivotStyle);  
worksheet.getRange("A1:H16").getColumns().autoFit();  
worksheet.getPageSetup().setTopMargin(30);  
worksheet.getPageSetup().setLeftMargin(30);  
worksheet.getRange("A1:H16").getColumns().autoFit();  
  
// Saving workbook to PDF  
workbook.save("PivotTableCustomStyle.pdf", SaveFileFormat.Pdf);
```



**Note:** While applying custom styles to the pivot table, it is important to note that if users apply a `TableStyle` whose **setShowAsAvailablePivotStyle** ('**setShowAsAvailablePivotStyle Method**' in the on-line documentation) method is false, then the `InvalidOperationException` is thrown.

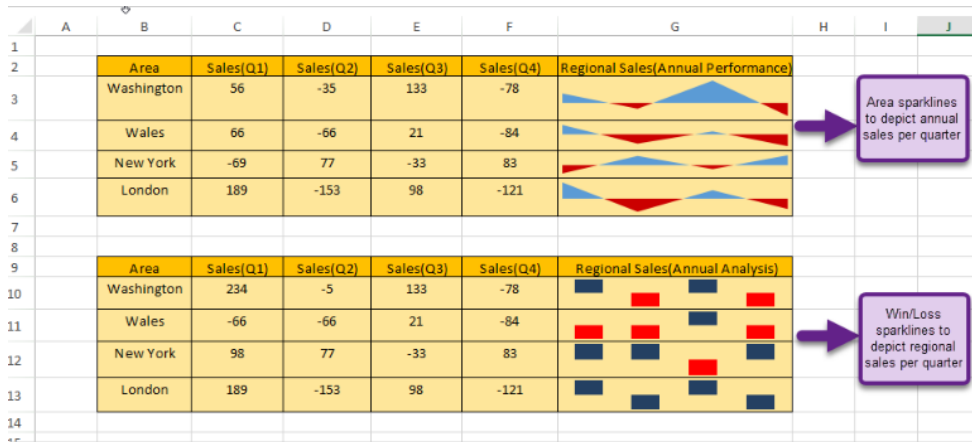
## Sparkline

GcExcel Java enables users to visualize data variations over time with Sparklines. Basically, a sparkline refers to a small, lightweight chart that can be drawn in a cell to rapidly visualize information for enhanced analytics.

Sparklines are useful especially when business analysts and managers need to work with analytical dashboards, presentations, business reports etc. A sparkline displays the most recent value as the rightmost data point and compares it with earlier values on a scale, enabling you to view general changes in data over time.

Sparklines fetch data from a range of cells and put it in the form of a small graph that fits inside a cell. Users can mark data values with different colors to denote various values (like low, high, last, first and negative) based on their specific preferences.





Refer to the following tasks in order to use sparklines.

- 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 insert 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** ('add Method' in the on-line documentation) method of the **ISparklineGroups** ('ISparklineGroups Interface' in the on-line documentation) interface and **getSparklineGroups** ('getSparklineGroups Method' in the on-line documentation) method of the **IRange** ('IRange Interface' in the on-line documentation) interface.

In order to insert a group of new sparklines, refer to the following example code.

Java

```
// Create workbook and access its first worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Defining data in the range
Object[][] data = new Object[][]
{
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 },
    { 10, 11, 12 }
};

// Add a group of new sparklines
worksheet.getRange("A1:C4").setValue(data);
worksheet.getRange("D1:D4").getSparklineGroups().add(SparkType.Line, "A1:C4");
```

#### Clear sparkline

You can remove a sparkline from your worksheet via specifying the data range and then using the **clear** ('clear Method' in the on-line documentation) method of the **ISparklineGroups** ('ISparklineGroups Interface' in the on-line documentation) interface.

In order to clear sparkline, refer to the following example code.

Java

```
// Defining data in the range
Object[][] data = new Object[][]
{
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 },
    { 10, 11, 12 }
};

worksheet.getRange("A1:C4").setValue(data);
```

```
worksheet.getRange("D1:D4").getSparklineGroups().add(SparkType.Line, "A1:C4");
worksheet.getRange("F1:H4").setValue(data);

// Add a group of new sparklines
worksheet.getRange("J1:J4").getSparklineGroups().add(SparkType.Line, "F1:H4");

// Clear D2 and J1 cell's sparkline.
worksheet.getRange("D2,J1").getSparklineGroups().clear();
```

### Clear sparkline groups

You can remove a group of sparklines (added for a row or column) from the spreadsheet via specifying the data range and then using the **clearGroups** ('clearGroups Method' in the on-line documentation) method of the **ISparklineGroups** ('ISparklineGroups Interface' in the on-line documentation) interface.

In order to clear sparkline groups, refer to the following example code.

```
Java

// Defining data in the range
Object[][] data = new Object[][]
{
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 },
    { 10, 11, 12 }
};
worksheet.getRange("A1:C4").setValue(data);

// Add a group of new sparklines
worksheet.getRange("D1:D4").getSparklineGroups().add(SparkType.Line, "A1:C4");
worksheet.getRange("F1:H4").setValue(data);
worksheet.getRange("J1:J4").getSparklineGroups().add(SparkType.Line, "F1:H4");

// Clear sparkline groups
worksheet.getRange("D2,J1").getSparklineGroups().clearGroups();
```

### Create a group of existing sparklines

You can create a group of existing sparklines in your worksheet via specifying the data range and then using the methods of the **ISparklineGroups** ('ISparklineGroups Interface' in the on-line documentation) interface.

In order to create a group of existing sparklines, refer to the following example code.

```
Java

// Defining data in the range
Object[][] data = new Object[][]
{
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 },
    { 10, 11, 12 }
};
worksheet.getRange("A1:C4").setValue(data);
worksheet.getRange("D1:D4").getSparklineGroups().add(SparkType.Line, "A1:C4");
worksheet.getRange("F1:H4").setValue(data);

// Add a group of new sparklines
worksheet.getRange("J1:J4").getSparklineGroups().add(SparkType.Column, "F1:H4");

// Create a new group, according to Range["J2"]'s sparkline group setting.
worksheet.getRange("A1:J4").getSparklineGroups().group(worksheet.getRange("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 methods of the **ISparklineGroups** ('ISparklineGroups Interface' in the on-line documentation) interface.

In order to add group of new sparkline with date axis, refer to the following example code.

```
Java

// Defining data in the range
```

```

Object[][] data = new Object[][]
{
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 },
    { 10, 11, 12 }
};
worksheet.getRange("A1:C4").setValue(data);

// Add a group of new sparklines
worksheet.getRange("D1:D4").getSparklineGroups().add(SparkType.Line, "A1:C4");
Object[] date_data = new Object[]
{
    new GregorianCalendar(2011, 11, 16),
    new GregorianCalendar(2011, 11, 17),
    new GregorianCalendar(2011, 11, 18)
};
worksheet.getRange("A7:C7").setValue(date_data);

// Set horizontal axis's Date range.
worksheet.getRange("D1").getSparklineGroups().get(0).setDateRange("A7:C7");
worksheet.getRange("D1").getSparklineGroups().get(0).getAxes().getHorizontal().getAxis().setVisible(true);
worksheet.getRange("D1").getSparklineGroups().get(0).getAxes().getHorizontal().getAxis().getColor().setColor(Color.GetGreen());
worksheet.getRange("D1").getSparklineGroups().get(0).getAxes().getVertical().setMinScaleType(SparkScale.SparkScaleCustom);
worksheet.getRange("D1").getSparklineGroups().get(0).getAxes().getVertical().setMaxScaleType(SparkScale.SparkScaleCustom);
worksheet.getRange("D1").getSparklineGroups().get(0).getAxes().getVertical().setCustomMinScaleValue(-2);
worksheet.getRange("D1").getSparklineGroups().get(0).getAxes().getVertical().setCustomMaxScaleValue(8);

```

### Configure layout of sparkline

You can configure the layout of the sparkline by using the methods of the **ISparklineGroup** ('**ISparklineGroup Interface**' in the on-line documentation) interface.

In order to configure the layout of the sparkline, refer to the following example code.

```

Java

// Defining data in the range
Object[][] data = new Object[][]
{
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 },
    { 10, 11, 12 }
};
worksheet.getRange("A1:C4").setValue(data);

// Adding sparkline
worksheet.getRange("D1:D4").getSparklineGroups().add(SparkType.Line, "A1:C4");

// Defining source data
Object[] date_data = new Object[]
{
    new GregorianCalendar(2011, 11, 16),
    new GregorianCalendar(2011, 11, 17),
    new GregorianCalendar(2011, 11, 18)
};
worksheet.getRange("A7:C7").setValue(date_data);

// Configuring the layout
ISparklineGroup sparklinegroup = worksheet.getRange("D1").getSparklineGroups().get(0);
sparklinegroup.setLineWeight(2.5);
sparklinegroup.getPoints().getMarkers().getColor().setColor(Color.GetRed());
sparklinegroup.getPoints().getMarkers().setVisible(true);
sparklinegroup.getSeriesColor().setColor(Color.GetPurple());

```

## Slicer

With GcExcel Java, you can execute quick data filtration in tables and pivot tables by inserting slicers in worksheets.

In order to work with slicers in a worksheet, refer to the following tasks:

- [Add Slicer in Table](#)
- [Add Slicer in Pivot Table](#)
- [Slicer Style](#)
- [Auto-Filter Table with Slicer](#)
- [Configure Slicer Layout](#)
- [Cut or Copy Slicer](#)
- [Duplicate Slicer](#)
- [Use Do Filter Operation](#)
- [Export Slicers](#)

## Add Slicer in Table

You can add slicers in tables and pivot tables using the methods of the **ISlicer** ('**ISlicer Interface**' in the **on-line documentation**) interface, **ISlicerCache** ('**ISlicerCache Interface**' in the **on-line documentation**) interface, and **ISlicerCaches** ('**ISlicerCaches Interface**' in the **on-line documentation**) interface.

The **add** ('**add Method**' in the **on-line documentation**) method of the **ISlicerCaches** interface allows users to create a new slicer cache for a table.

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

Java

```
// Defining source data
Object sourceData = new Object[][]
{
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United States" },
    { 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United Kingdom" },
    { 3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States" },
    { 4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada" },
    { 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany" },
    { 6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United States" },
    { 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11), "Australia" },
    { 8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand" },
    { 9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France" },
    { 10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada" },
    { 11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany" },
    { 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18), "United States" },
    { 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20), "Germany" },
    { 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22), "Canada" },
    { 15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France" },
}
```

```

};

// Initialize the workbook and fetch the default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("A:F").setColumnWidth(15);

// Adding data to the table
worksheet.getRange("A1:F16").setValue(sourceData);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");

// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add slicers for table
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"cate1",
"Category", 30, 550, 100, 200);
ISlicer slicer2 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"cate2",
"Category", 30, 700, 100, 200);

```

## Add Slicer in Pivot Table

Gcexcel enables users to efficiently organize data in pivot tables and multi pivot tables via slicers.

The methods of the **ISlicerCaches** ('**ISlicerCaches Interface**' in the on-line documentation) interface, the **ISlicerCache** ('**ISlicerCache Interface**' in the on-line documentation) interface, the **IPivotCache** ('**IPivotCache Interface**' in the on-line documentation) interface, **IPivotCaches** ('**IPivotCaches Interface**' in the on-line documentation) interface, **IPivotField** ('**IPivotField Interface**' in the on-line documentation) interface, **IPivotFields** ('**IPivotFields Interface**' in the on-line documentation) interface, **IPivotTable** ('**IPivotTable Interface**' in the on-line documentation) interface, **IPivotTables** ('**IPivotTables Interface**' in the on-line documentation) interface and the **IPivotItem** ('**IPivotItem Interface**' in the on-line documentation) interface can be used to insert slicers in pivot tables.

In order to insert slicer in a pivot table, you can use the **add** ('**add Method**' in the on-line documentation) method of the **ISlicerCaches** interface to create a new slicer cache for a pivot table, as shown in the example code shared below.

Java

```

// Defining source data
Object sourceData = new Object[][]
{
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United States" },
    { 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United Kingdom" },
    { 3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States" },
}

```

```
{ 4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada" },
{ 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany" },
{ 6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United States" },
{ 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11), "Australia"
},
{ 8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand" },
{ 9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France" },
{ 10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada" },
{ 11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany" },
{ 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18), "United
States" },
{ 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20), "Germany" },
{ 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22), "Canada" },
{ 15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France" },
};

// Initialize the workbook and fetch the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Adding data to the pivot table
worksheet.getRange("A1:F16").setValue(sourceData);
worksheet.getRange("A:F").setColumnWidth(15);

// Create pivot cache.
IPivotCache pivotcache = workbook.getPivotCaches().create(worksheet.getRange("A1:F16"));

// Create pivot tables.
IPivotTable pivottable1 = worksheet.getPivotTables().add(pivotcache,
worksheet.getRange("K5"), "pivottable1");
IPivotTable pivottable2 = worksheet.getPivotTables().add(pivotcache,
worksheet.getRange("N3"), "pivottable2");
worksheet.getRange("D2:D16").setNumberFormat("$#,##0.00");

// Configure pivot fields
IPivotField field_product1 = pivottable1.getPivotFields().get(1);
field_product1.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Amount1 = pivottable1.getPivotFields().get(3);
field_Amount1.setOrientation(PivotFieldOrientation.DataField);

IPivotField field_product2 = pivottable2.getPivotFields().get(5);
field_product2.setOrientation(PivotFieldOrientation.RowField);

IPivotField field_Amount2 = pivottable2.getPivotFields().get(2);
field_Amount2.setOrientation(PivotFieldOrientation.DataField);
field_Amount2.setFunction(ConsolidationFunction.Count);
```

```
// Create slicer cache and the slicers base. The slicer cache controls pivot table1
ISlicerCache cache = workbook.getSlicerCaches().add(pivottable1, "Product");
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"p1", "Product", 30, 550, 100, 200);

// Add pivot table2 for slicer cache. Slicer cache will control pivot table1 and pivot
table2
cache.getPivotTables().addPivotTable(pivottable2);
```

In order to add slicer in a multi pivot table, refer to the following example code.

#### Java

```
ISlicerCache cache = workbook.getSlicerCaches().add(pivottable1, "Product");
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"), "p1",
"Product",
20, 20, 100,200);
cache.getPivotTables().addPivotTable(pivottable2);

// Set slicer style to built-in style
slicer1.setStyle(workbook.getTableStyles().get("SlicerStyleLight2"));
```

## Slicer Style

GcExcel Java enables users to apply style to the slicers. While adding a slicer, users first need to create a slicer cache and then use the slicer cache created base on the column of the table or the pivot table.

The **ISlicerCaches** (**'ISlicerCaches Interface' in the on-line documentation**) interface in GcExcel Java holds all the slicer caches in the workbook.

### Set slicer to built-in style

You can set a slicer to built-in style by using the **setStyle** (**'setStyle Method' in the on-line documentation**) method of the **ISlicer** (**'ISlicer Interface' in the on-line documentation**) interface.

In order to set slicer to built-in style, refer to the following example code.

#### Java

```
// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add slicer
ISlicer slicer = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"), "catel",
"Category", 30, 550, 100, 200);

// Set slicer style to built-in style.
slicer.setStyle(workbook.getTableStyles().get("SlicerStyleLight2"));
```

## Modify Slicer with Custom Style

In GcExcel Java, 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.

Java

```
// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add slicer
ISlicer slicer = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"cate2", "Category", 30, 550, 100, 200);

// Create custom slicer style.
ITableStyle slicerStyle = workbook.getTableStyles().add("test");

// Set ShowAsAvailableSlicerStyle to true, the style will be treated as slicer style.
slicerStyle.setShowAsAvailableSlicerStyle(true);
slicerStyle.getTableStyleElements().get(TableStyleElementType.WholeTable).getFont().setName("Arial");
slicerStyle.getTableStyleElements().get(TableStyleElementType.WholeTable).getFont().setBold(false);
slicerStyle.getTableStyleElements().get(TableStyleElementType.WholeTable).getFont().setItalic(false);
slicerStyle.getTableStyleElements().get(TableStyleElementType.WholeTable).getFont()
.setColor(Color.GetWhite());
slicerStyle.getTableStyleElements().get(TableStyleElementType.WholeTable).getBorders()
.setColor(Color.GetLightPink());
slicerStyle.getTableStyleElements().get(TableStyleElementType.WholeTable).getInterior()
.setColor(Color.GetLightGreen());

// Set slicer style to custom style.
slicer.setStyle(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 **setRowHeight** ('**setRowHeight Method**' in the **on-line documentation**) method and **setDisplayHeader** ('**setDisplayHeader Method**' in the **on-line documentation**) method of the **ISlicer** ('**ISlicer Interface**' in the **on-line documentation**) interface.

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

Java

```
// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Product", "productCache");

// Add slicer for the table
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"product1", "Product", 30, 550, 100, 200);

// Configure slicer layout.
```



```
slicer1.setNumberOfColumns(2);
slicer1.setRowHeight(25);
slicer1.setDisplayHeader(false);
```

## Auto-Filter Table with Slicer

In GcExcel Java, you can automatically filter tables with slicers via using the methods of the **ISlicerCaches ('ISlicerCaches Interface' in the on-line documentation)** interface. This helps in creating a new slicer cache for your table.

In order to auto-filter table with slicer, refer to the following example code.

Java

```
// Defining source data
Object sourceData = new Object[][]
{
    {"Order ID", "Product", "Category", "Amount", "Date", "Country"},
    {1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United States"},
    {2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United Kingdom"},
    {3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States"},
    {4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada"},
    {5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany"},
    {6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United States"},
    {7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11), "Australia"},
    {8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand"},
    {9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France"},
    {10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada"},
    {11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany"},
    {12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18), "United States"},
    {13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20), "Germany"},
    {14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22), "Canada"},
    {15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France"},
};

// Initialize the workbook and fetch the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Adding data to the table
worksheet.getRange("A:F").setColumnWidth(15);
worksheet.getRange("A1:F16").setValue(sourceData);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");

// Create slicer cache for table.
```

```
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add slicer for table
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"catel",
"Category", 20, 550, 100, 200);

// Apply table filter
// This synchronizes automatically to the slicer. The slicer1's selected item is
"Fruit".
worksheet.getRange("A1:F16").autoFilter(2, "Fruit");
```

## Configure Slicer Layout

GcExcel Java enables users to configure slicer layout using the methods of the **ISlicerCaches** ('**ISlicerCaches Interface**' in **the on-line documentation**) interface that creates a new slicer cache for a table.

In order to configure slicer layout for a table, refer to the following example code.

### Java

```
// Defining source data
Object sourceData = new Object[][] { { "Order ID", "Product", "Category", "Amount",
"Date", "Country" },
{ 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United
States" },
{ 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United
Kingdom" },
{ 3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States"
},
{ 4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada" },
{ 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany"
},
{ 6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United
States" },
{ 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11),
"Australia" },
{ 8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand"
},
{ 9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France" },
{ 10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada" },
{ 11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany" },
{ 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18),
"United States" },
{ 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20),
"Germany" },
{ 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22),
"Canada" },
```

```

        { 15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France" }, };

// Initialize the workbook and fetch the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Adding data to the table
worksheet.getRange("A:F").setColumnWidth(15);
worksheet.getRange("A1:F16").setValue(sourceData);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");

// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Product", "productCache");

// Add slicer for the table
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"product1", "Product", 30, 550, 100, 200);

// Configure slicer layout.
slicer1.setNumberOfColumns(2);
slicer1.setRowHeight(25);
slicer1.setDisplayHeader(false);

```

## Cut or Copy Slicer

You can cut or copy slicer in a table using the methods of the **ISlicerCaches** (**'ISlicerCaches Interface' in the on-line documentation**) interface.

In order to copy slicer in table, refer to the following example code.

### Java

```

// Defining source data
Object sourceData = new Object[][]
{
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United States" },
    { 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United Kingdom" },
    { 3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States" },
    { 4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada" },
    { 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany" },
    { 6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United States" },
    { 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11), "Australia" },
    { 8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand" },
    { 9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France" },

```

```

{ 10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada" },
{ 11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany" },
{ 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18), "United
States" },
{ 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20), "Germany" },
{ 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22), "Canada" },
{ 15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France" },
};

// Initialize the workbook and fetch the default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("A:F").setColumnWidth(15);

// Adding data to the table
worksheet.getRange("A1:F16").setValue(sourceData);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");

// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add slicer, slicer's range is Range["H3:J16"]
ISlicer slicer = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"), "catel",
"Category", 30, 550, 100, 200);

// Range["H3:J16"] must contain slicer's range, copy a new shape to Range["K3:M16"]
worksheet.getRange("H3:J16").Copy(worksheet.getRange("K3"));

```

In order to cut slicer in table, refer to the following example code.

#### Java

```

Object sourceData = new Object[][]
{
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United States"
},
    { 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United
Kingdom" },
    { 3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States" },
    { 4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada" },
    { 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany" },
    { 6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United States" },
    { 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11), "Australia" },
    { 8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand" },
    { 9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France" },
    { 10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada" },
    { 11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany" },

```

```

    { 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18), "United States" },
    { 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20), "Germany" },
    { 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22), "Canada" },
    { 15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France" },
};

IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("A:F").setColumnWidth(15);
worksheet.getRange("A1:F16").setValue(sourceData);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");

// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add slicer, slicer's range is Range["H3:J16"]
ISlicer slicer = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"), "catel",
"Category", 30, 550, 100, 200);

// Range["H3:J16"] must contain slicer's range, cut a new shape to Range["K3:M16"]
worksheet.getRange("H3:J16").Cut(worksheet.getRange("K3"));

```

## Duplicate Slicer

You can add duplicate slicer using GcExcel Java with the help of the **add ('add Method' in the on-line documentation)** method of the **ISlicerCaches ('ISlicerCaches Interface' in the on-line documentation)** interface.

In order to add duplicate slicer in the worksheet, refer to the following example code.

### Java

```

// Defining source data
Object sourceData = new Object[][]
{
    { "Order ID", "Product", "Category", "Amount", "Date", "Country" },
    { 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2018, 0, 6), "United States" },
},
    { 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2018, 0, 7), "United Kingdom" },
    { 3, "Banana", "Fruit", 617, new GregorianCalendar(2018, 0, 8), "United States" },
    { 4, "Banana", "Fruit", 8384, new GregorianCalendar(2018, 0, 10), "Canada" },
    { 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2018, 0, 10), "Germany" },
    { 6, "Orange", "Fruit", 3610, new GregorianCalendar(2018, 0, 11), "United States" },
    { 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2018, 0, 11), "Australia" },
    { 8, "Banana", "Fruit", 6906, new GregorianCalendar(2018, 0, 16), "New Zealand" },
    { 9, "Apple", "Fruit", 2417, new GregorianCalendar(2018, 0, 16), "France" },
    { 10, "Apple", "Fruit", 7431, new GregorianCalendar(2018, 0, 16), "Canada" },

```

```
{ 11, "Banana", "Fruit", 8250, new GregorianCalendar(2018, 0, 16), "Germany" },
{ 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2018, 0, 18), "United
States" },
{ 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2018, 0, 20), "Germany" },
{ 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2018, 0, 22), "Canada" },
{ 15, "Apple", "Fruit", 6946, new GregorianCalendar(2018, 0, 24), "France" },
};

// Initialize the workbook and fetch the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Adding data to the table
worksheet.getRange("A:F").setColumnWidth(15);
worksheet.getRange("A1:F16").setValue(sourceData);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");

// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add slicer for table
ISlicer slicer = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"), "catel",
"Category", 30, 550, 100, 200);

// Duplicate slicer
IShape newShape = slicer.getShape().duplicate();
```

## Use Do Filter Operation

GcExcel Java enables users to apply filters to slicers, thus enabling users to analyse bulk information in a spreadsheet quickly and effectively.

### Use slicer do-filter operation

In order to use slicer to perform do-filter operation, refer to the following example code.

#### Java

```
// Adding data to the table
worksheet.getRange("A:F").setColumnWidth(15);
worksheet.getRange("A1:F16").setValue(sourceData);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");

// Create slicer cache for the table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");
```

```
// Add slicer for table
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"catel",
"Category", 30, 550, 100, 200);

// Execute the do filter operation. Here we are filtering vegetables.
slicer1.getSlicerCache().getSlicerItems().get("Vegetables").setSelected(false);
```

### Clear slicer filter

In order to clear slicer filter, refer to the following example code.

#### Java

```
// Adding data to the table
worksheet.getRange("A:F").setColumnWidth(15);
worksheet.getRange("A1:F16").setValue(sourceData);
ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");

// Create slicer cache for table.
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add slicer for table
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
"catel",
"Category", 30, 550, 100, 200);

// Execute the do filter operation. Here we are filtering vegetables.
slicer1.getSlicerCache().getSlicerItems().get("Vegetables").setSelected(false);

// Clear the slicer filter.
slicer1.getSlicerCache().clearAllFilters();
```

## Print Settings

GcExcel Java provides users with several Page Setup options to facilitate users in executing the print operations swiftly and effectively.

While printing, you can use the Page Setup feature to personalize the layout of your page. This includes the page size, page orientation (landscape and portrait), header and footer, etc. along with many essential paper settings that can be configured while printing.

- [Configure Page Header and Footer](#)
- [Configure Page Settings](#)
- [Configure Page Breaks](#)
- [Configure Paper Settings](#)
- [Configure Print Area](#)
- [Configure Columns to Repeat at Left and Right](#)

- [Configure Rows to Repeat at Top and Bottom](#)
- [configure-drafts](#)
- [Configure Print Page Range](#)
- [Configure Sheet Print Settings](#)

## Configure Page Header and Footer

In GcExcel Java, you can use the **setLeftHeader** ('setLeftHeader Method' in the on-line documentation), **setRightHeader** ('setRightHeader Method' in the on-line documentation), **setLeftFooter** ('setLeftFooter Method' in the on-line documentation), **setRightFooter** ('setRightFooter Method' in the on-line documentation), **setCenterHeader** ('setCenterHeader Method' in the on-line documentation) and **setCenterFooter** ('setCenterFooter Method' in the on-line documentation) methods of the **IPageSetup** ('IPageSetup Interface' in the on-line documentation) interface in order to configure header and footer for a page.

Java

```
// Configure PageHeader and PageFooter
// Set header for the page
worksheet.getPageSetup().setLeftHeader("&\\"Arial,Italic\\"LeftHeader");
worksheet.getPageSetup().setCenterHeader("&P");

// Set header and footer graphic for the page
worksheet.getPageSetup().setCenterFooter("&G");
InputStream stream = ClassLoader.getResourceAsStream("logo.png");
worksheet.getPageSetup().getCenterFooterPicture().setGraphicStream(stream, ImageType.PNG);
```

For special settings, you can also perform the following tasks in order to customize the configuration of the header and footer of your page:

1. **Configure first page header and footer**
2. **Configure even page header and footer**

### Configure first page header and footer

If you want a different header and footer in your first page, you first need to set the **setDifferentFirstPageHeaderFooter** ('setDifferentFirstPageHeaderFooter Method' in the on-line documentation) method of the **IPageSetup** interface to true. When this is done, you can use the method of the **IPageSetup** interface in order to configure the first page header and footer.

Java

```
//Set first page header and footer
worksheet.getPageSetup().setDifferentFirstPageHeaderFooter(true);
worksheet.getPageSetup().getFirstPage().getCenterHeader().setText("&T");
worksheet.getPageSetup().getFirstPage().getRightFooter().setText("&D");

//Set first page header and footer graphic
worksheet.getPageSetup().getFirstPage().getLeftFooter().setText("&G");
InputStream stream = ClassLoader.getResourceAsStream("logo.png");
worksheet.getPageSetup().getFirstPage().getLeftFooter().getPicture().setGraphicStream(stream, ImageType.PNG);
worksheet.getPageSetup().getFirstPage().getLeftFooter().getPicture().setWidth(100);
worksheet.getPageSetup().getFirstPage().getLeftFooter().getPicture().setHeight(13);
```



### Configure even page header and footer

If you want a different header and footer for all the even pages, you first need to set the **setOddAndEvenPagesHeaderFooter** ('**setOddAndEvenPagesHeaderFooter Method**' in the **on-line documentation**) method to true. When this is done, you can use other methods of the `IPageSetup` interface in order to configure the even page header and footer.

Java

```
// Set even page header and footer
worksheet.getPageSetup().setOddAndEvenPagesHeaderFooter(true);
worksheet.getPageSetup().getEvenPage().getCenterHeader().setText("&T");
worksheet.getPageSetup().getEvenPage().getRightFooter().setText("&D");

// Set even page header and footer graphic
worksheet.getPageSetup().getEvenPage().getLeftFooter().setText("&G");
InputStream stream = ClassLoader.getResourceAsStream("logo.png");
worksheet.getPageSetup().getEvenPage().getLeftFooter().getPicture().setGraphicStream(stream,
ImageType.PNG);
```

## Configure Page Settings

In GcExcel Java, you can use the methods of the **IPageSetup** ('**IPageSetup Interface**' in the **on-line documentation**) interface in order to configure page settings.

Configuring page settings involves the following tasks:


1. **Configure Page Margins**
2. **Configure Page Orientation**
3. **Configure Page Order**
4. **Configure Page Center**
5. **Configure First Page Number**

### Configure Page Margins

You can use the **setTopMargin** ('**setTopMargin Method**' in the **on-line documentation**) method, **setHeaderMargin** ('**setHeaderMargin Method**' in the **on-line documentation**) method, **setFooterMargin** ('**setFooterMargin Method**' in the **on-line documentation**) method, **setRightMargin** ('**setRightMargin Method**' in the **on-line documentation**) method, **setLeftMargin** ('**setLeftMargin Method**' in the **on-line documentation**) method and the **setBottomMargin** ('**setBottomMargin Method**' in the **on-line documentation**) method of the `IPageSetup` interface to configure margins for a page.

Java

```
// Set margins, in points.
worksheet.getPageSetup().setTopMargin(36);
worksheet.getPageSetup().setBottomMargin(36);
worksheet.getPageSetup().setLeftMargin(28.8);
worksheet.getPageSetup().setRightMargin(72);
worksheet.getPageSetup().setHeaderMargin(0);
worksheet.getPageSetup().setFooterMargin(93.6);
```

 **Note:** While you set margins for your page, it is necessary to ensure that it should not be less than Zero.

## Configure Page Orientation

You can use the **setOrientation** ('**setOrientation Method**' in the on-line documentation) method of the IPageSetup interface in order to set the orientation for a page to Portrait or Landscape as per custom preferences.

Java

```
// Set page orientation, default is portrait.
worksheet.getPageSetup().setOrientation(PageOrientation.Landscape);
```

## Configure Page Order

You can use the **setOrder** ('**setOrder Method**' in the on-line documentation) method of the IPageSetup interface in order to configure the order of the page as per your choice.

Java

```
// Set page order. The default value is DownThenOver.
worksheet.getPageSetup().setOrder(Order.OverThenDown);
```

## Configure Page Center

You can use the **setCenterHorizontally** ('**setCenterHorizontally Method**' in the on-line documentation) method and the **setCenterVertically** ('**setCenterVertically Method**' in the on-line documentation) method of the IPageSetup interface in order to configure the center of the page according to your custom preferences.

Java

```
// Set page center. The default value is false.
worksheet.getPageSetup().setCenterHorizontally(true);
worksheet.getPageSetup().setCenterVertically(true);
```

## Configure First Page Number

You can use the **setFirstPageNumber** ('**setFirstPageNumber Method**' in the on-line documentation) method of the IPageSetup interface in order to configure the number for your first page as per your choice.

Java

```
// Set first page number, default is p1.
worksheet.getPageSetup().setFirstPageNumber(3);
```

## Configure Page Breaks

GcExcel Java allows users to configure the vertical and horizontal page breaks by using the **getHPageBreaks** ('**getHPageBreaks Method**' in the on-line documentation) method and **getVPageBreaks** ('**getVPageBreaks Method**' in the on-line documentation) method of the **IWorksheet** ('**IWorksheet Interface**' in the on-line documentation) interface.

You can also determine whether to adjust the horizontal and vertical page breaks or keep them fixed (while performing the insert and delete operations on the rows and columns) by using the **getFixedPageBreaks** ('**getFixedPageBreaks Method**' in the on-line documentation) and the **setFixedPageBreaks** ('**setFixedPageBreaks Method**' in the on-line documentation) methods of the `IWorksheet` interface.

This feature is useful especially when users need to print different reports from Excel to a PDF file. With the option to choose whether to adjust page breaks or keep them fixed, users can specify whether each section appears on a separate page or starts from a new page whenever any rows and columns are inserted or deleted in a spreadsheet.

If the **setFixedPageBreaks** ('**setFixedPageBreaks Method**' in the on-line documentation) method is set to false (this is the default behavior), then:

- The horizontal and vertical page breaks are adjusted when the rows and columns are inserted or deleted from the worksheet.
- The row or column index of the page break is increased or decreased according to the inserted and deleted rows or columns based on the following conditions:
  - If the inserted or deleted rows or columns exist after the page break, the row or column index of the page break remains unchanged.
  - If the deleted rows or columns are present before the page break, the row or column index of the page break is decreased accordingly.
  - If the deleted rows or columns contain the page break, the page break will be removed.

If the **setFixedPageBreaks** ('**setFixedPageBreaks Method**' in the on-line documentation) method is set to true, the row or column index of page breaks are not changed even after inserting or deleting rows or columns. Further, the horizontal and vertical page breaks are considered "fixed" and the page breaks can't be adjusted in this scenario.

In order to configure page breaks for customized printing, refer to the following example code.

#### Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

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

// Set data
worksheet.getRange("A1:G9").setValue(data);

// Add a horizontal page break before the fourth row
IHPageBreak hPageBreak = worksheet.getHPageBreaks().add(worksheet.getRange("F4"));
```

```
// Add a vertical page break before the third column
IVPageBreak vPageBreak = worksheet.getVPageBreaks().add(worksheet.getRange("F3"));

// Saving workbook to xlsx
workbook.save("7-PageBreaks.xlsx", SaveFileFormat.Xlsx);

/* Delete rows and columns before the page breaks, the page breaks will be
   adjusted */
worksheet.getRange("1:2").delete(); // hPageBreak is before the 4th row
worksheet.getRange("B:C").delete(); // vPageBreak is before the 4th column

/* Set the page breaks are fixed, it will not be adjusted when inserting/
   deleting rows/ columns */
worksheet.setFixedPageBreaks(true);

// Saving workbook to xlsx
workbook.save("PageBreaks1.xlsx", SaveFileFormat.Xlsx);

/* Delete rows and columns after the page breaks, the page breaks will not be
   adjusted */
worksheet.getRange("1:2").delete(); // hPageBreak is still before the 4th row
worksheet.getRange("B:C").delete(); // vPageBreak is still before the 4th column

// Inserting rows after deleting row and column ranges
worksheet.getRange("A3:A5").getEntireRow().insert();

// Saving workbook to xlsx
workbook.save("PageBreaks2.xlsx", SaveFileFormat.Xlsx);
```

## Configure Paper Settings

In GcExcel Java, you can use the methods of the **IPageSetup** (**'IPageSetup Interface' in the on-line documentation**) interface in order to configure paper settings for customized printing.

Configuring paper settings involves the following tasks:

1. **Configure Paper Scaling**
2. **Configure Paper Size**

### Configure Paper Scaling

You can use the **getIsPercentScale** (**'getIsPercentScale Method' in the on-line documentation**) method in order to control how the spreadsheet is scaled; the **setIsPercentScale** (**'setIsPercentScale Method' in the on-line documentation**) method, the **setZoom** (**'setZoom Method' in the on-line documentation**) method and the **getPageSetup** (**'getPageSetup Method' in the on-line documentation**) method in order to adjust the size of the paper that will be used for printing.

Java

```
// Set paper scaling via percent scale
worksheet.getPageSetup().setIsPercentScale(true);

// Set paper scaling via zoom
worksheet.getPageSetup().setZoom(150);

// Set paper scaling via Fit to page's wide & tall
worksheet.getPageSetup().setIsPercentScale(false);
worksheet.getPageSetup().setFitToPagesWide(3);
worksheet.getPageSetup().setFitToPagesTall(4);
```

### Configure Paper Size

You can use the **getPageSetup** ('**getPageSetup Method**' in the on-line documentation) method and **setPaperSize** ('**setPaperSize Method**' in the on-line documentation) method in order to set the paper size for the paper that will be used for printing.

Java

```
// Set built-in paper size. Default is Letter Format - A4
worksheet.getPageSetup().setPaperSize(PaperSize.A4);
```

## Configure Print Area

At times, you may want to print only a specific area in a worksheet instead of printing the whole worksheet.

GcExcel Java supports customized printing by allowing users to select a range of cells in order to configure the desired print area in a worksheet. This can be done by using the **setPrintArea** ('**setPrintArea Method**' in the on-line documentation) method, **setPrintTitleRows** ('**setPrintTitleRows Method**' in the on-line documentation) method and **setPrintTitleColumns** ('**setPrintTitleColumns Method**' in the on-line documentation) method of the **IPageSetup** ('**IPageSetup Interface**' in the on-line documentation) interface.

Java

```
// Set print area & print titles in the worksheet
worksheet.getPageSetup().setPrintArea("$D$5:$G$10");
worksheet.getPageSetup().setPrintTitleRows("$5:$10");
worksheet.getPageSetup().setPrintTitleColumns("$D:$G");
```

## Configure Columns to Repeat at Left and Right

You can configure columns in a worksheet to repeat them at the left by using the **setPrintTitleColumns** ('**setPrintTitleColumns Method**' in the on-line documentation) method and at the right by using

the **setPrintTailColumns** ('**setPrintTailColumns Method**' in the on-line documentation) method of the **IPageSetup** ('**IPageSetup Interface**' in the on-line documentation) interface.

This feature is useful especially when you're using GcExcel Java to create reports wherein you want to repeat specific title columns and tail columns in the exported file. With support for repeating specific columns at the left and right side of the page; it becomes much easier and quicker to handle and visualize spreadsheets containing large number of columns.

While exporting a spreadsheet with repeating columns to a PDF file, the tail columns will be exported only when its index is larger than the page's last column's index. Otherwise, the tail column is ignored. For instance, if the Print Area is "A1:J200" and the right repeating column is "\$I:\$J"; it will print "\$I:\$J" repeatedly on each page. However, if users set the right repeating column to "\$K:\$L", then it will not print "\$K:\$L" (because the column index is larger than print area).

Refer to the following example code in order to configure columns to repeat at the right.

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Populating cells in worksheet
for (int i = 0; i < 200; i++)
    for (int j = 0; j < 8; j++) {
        worksheet.getRange(i, j).setValue(i);
        worksheet.getRange(i, 8).setValue("Row I");
        worksheet.getRange(i, 9).setValue("Row J");
    }

// Repeat Columns from I to J at the right of each page while saving pdf
worksheet.getPageSetup().setPrintTailColumns("$I:$J");

// Saving workbook to pdf
workbook.save("ConfigureTailColumns.pdf", SaveFileFormat.Pdf);
```

Refer to the following example code in order to configure columns to repeat at the left.

Java

```
// Set columns to repeat at left
worksheet.getPageSetup().setPrintTitleColumns("$D:$G");
```

## Configure Rows to Repeat at Top and Bottom

You can configure rows in a worksheet in order to repeat them at the top by using the **setPrintTitleRows()** ('**setPrintTitleRows Method**' in the on-line documentation) method and at the bottom by using the **setPrintTailRows()** ('**setPrintTailRows Method**' in the on-line documentation) method of the **IPageSetup** ('**IPageSetup Interface**' in the on-line documentation) interface.

While exporting a spreadsheet with repeating rows to a PDF file, the tail rows will be exported only when its index is larger

than the page's last row's index. Otherwise, the tail row is ignored. For instance, if the Print Area is "B5:H23" and the top repeating row is "\$3:\$3"; it will print "\$3:\$3" repeatedly on each page. However, if users set the top repeating row to "\$30:\$30", then it will not print "\$30:\$30" (because the row index is larger than print area).

Refer to the following example code in order to configure rows to repeat at the bottom.

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Populating cells in worksheet
for (int i = 0; i < 200; i++)
    for (int j = 0; j < 10; j++) {
        worksheet.getRange(i, j).setValue(i);
        worksheet.getRange(199, j).setValue("Row 199");
    }

// Repeat Row 200 at the bottom of each page while saving pdf
worksheet.getPageSetup().setPrintTailRows("$200:$200");

// Saving workbook to pdf
workbook.save("ConfigureTailRows.pdf", SaveFileFormat.Pdf);
```

In order to configure rows to repeat at top, refer to the following example code.

Java

```
// Set rows to repeat at top
worksheet.getPageSetup().setPrintTitleRows("$5:$10");
```

## configure-drafts

GcExcel Java allows users to save drafts of the worksheets while executing the print operation.

In order to configure drafts while printing, refer to the following example code.

Java

```
// Create a new workbook and access the default worksheet
Workbook workbook = new Workbook();
IWorksheet sheet = workbook.getWorksheets().get(0);

// Set text.
sheet.getRange("A1:G10").setValue("Text");

// Add picture in worksheet.
```

```
FileInputStream stream = null;
try
{
    stream = new FileInputStream("Pictures/logo.png");
}
catch (FileNotFoundException e)
{
    e.printStackTrace();
}
try
{
    IShape picture = sheet.getShapes()
        .addPicture(stream, ImageType.PNG, 20, 20, 395, 60);
}
catch (IOException ioe)
{
}

// Add header graphic.
FileInputStream stream1 = null;
try
{
    stream1 = new FileInputStream("Pictures/logo.png");
}
catch (FileNotFoundException e)
{
    e.printStackTrace();
}
sheet.getPageSetup().setCenterHeader("&G");
sheet.getPageSetup().getCenterHeaderPicture().setGraphicStream(stream1, ImageType.PNG);
sheet.getPageSetup().getCenterHeaderPicture().setWidth(100);
sheet.getPageSetup().getCenterHeaderPicture().setHeight(13);

// Set print without graphics in page content area.
sheet.getPageSetup().setDraft(true);

// Save to a pdf file
workbook.save("Draft.pdf", SaveFileFormat.Pdf);
```

## Configure Print Page Range

GcExcel Java allows users to specify the page range while printing a worksheet.

In order to configure page range for the print operation, refer to the following example code.



Java

```
// Create a new workbook and access the default worksheet
Workbook workbook = new Workbook();
IWorksheet sheet = workbook.getWorksheets().get(0);

// Set pages' data.
sheet.getRange("A1:J46").setValue("Page1");
sheet.getRange("A1:J46").getInterior().setColor(Color.GetLightGreen());

sheet.getRange("A47:J92").setValue("Page2");
sheet.getRange("A47:J92").getInterior().setColor(Color.GetLightYellow());

sheet.getRange("K1:T46").setValue("Page3");
sheet.getRange("K1:T46").getInterior().setColor(Color.GetOrangeRed());

sheet.getRange("K47:T92").setValue("Page4");
sheet.getRange("K47:T92").getInterior().setColor(Color.GetDarkOrange());

sheet.getRange("U1:AD46").setValue("Page5");
sheet.getRange("U1:AD46").getInterior().setColor(Color.GetLightBlue());

sheet.getRange("U47:AD92").setValue("Page6");
sheet.getRange("U47:AD92").getInterior().setColor(Color.GetIndianRed());
sheet.getPageSetup().setPrintHeadings(true);

// Set print page range, print p1, p3 to p5.
sheet.getPageSetup().setPrintPageRange("1,3-5");

// Save to a pdf file
workbook.save("PrintPageRange.pdf", SaveFileFormat.Pdf);
```

## Configure Sheet Print Settings

In order to configure the print settings for the sheet, you can use the **setPrintGridlines** ('setPrintGridlines Method' in the on-line documentation) method, **setPrintHeadings** ('setPrintHeadings Method' in the on-line documentation) method, **setBlackAndWhite** ('setBlackAndWhite Method' in the on-line documentation) method, **setPrintComments** ('setPrintComments Method' in the on-line documentation) method and **setPrintErrors** ('setPrintErrors Method' in the on-line documentation) method of the **IPageSetup** ('IPageSetup Interface' in the on-line documentation) interface as shown in the example code shared below.

Java

```
// Configure sheet print settings
worksheet.getPageSetup().setPrintGridlines(true);
worksheet.getPageSetup().setPrintHeadings(true);
```

```
worksheet.getPageSetup().setBlackAndWhite(true);  
worksheet.getPageSetup().setPrintComments(PrintLocation.InPlace);  
worksheet.getPageSetup().setPrintErrors(PrintErrors.Dash);
```

## Templates

Report generation is crucial for creating marketing strategies, project management, product development cycles, budgeting estimates and growth strategies. It is a common requirement of any business domain. Excel reports are periodically generated and consume a considerable amount of time and effort. However, the chances of manual error cannot be eliminated altogether. That's where the use of templates finds its place. GcExcel provides templates to create highly effective and well-designed Excel reports.

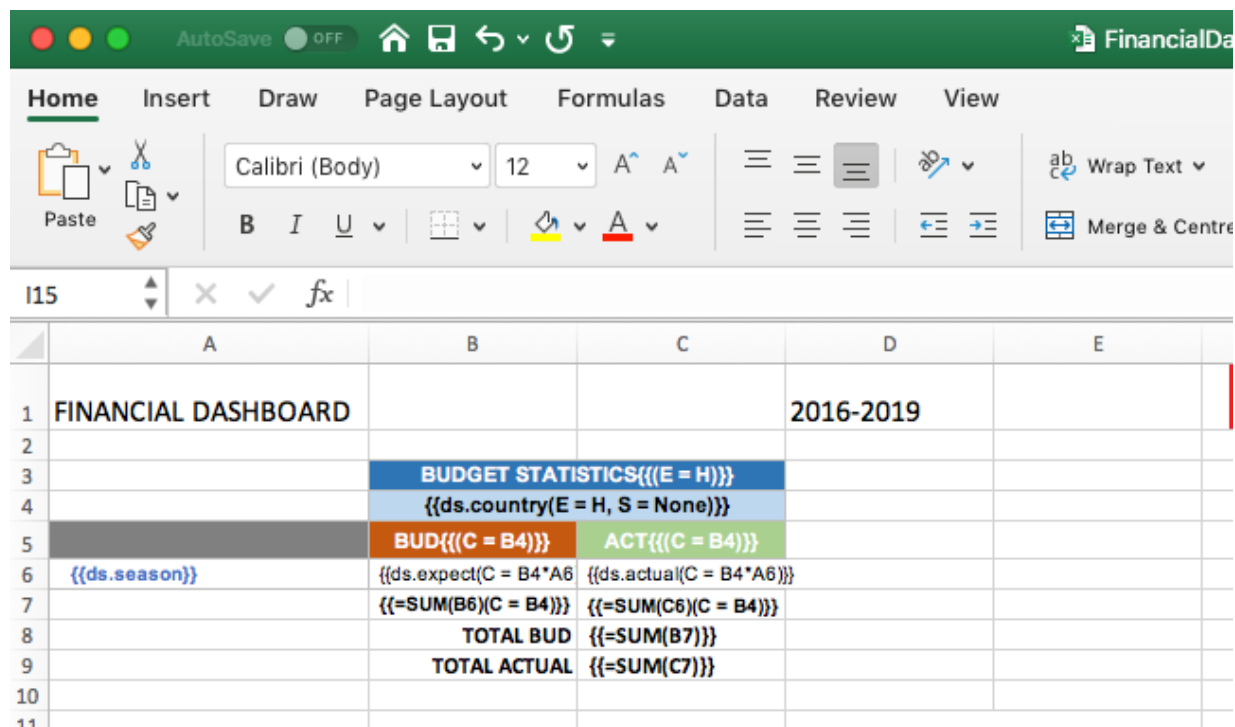
Some of the powerful features provided by GcExcel templates are as follows:

- **Flexible:** GcExcel templates have a highly flexible template syntax and API to bind Excel documents to data. It follows easy data population rules in fields.
- **Efficient:** GcExcel templates provide extended reusability. This means the templates can be used with minor modifications, or as it is time and again, saving both time and effort.
- **Multi-platform:** GcExcel templates are supported on Windows, Linux and macOS.
- **Multi-domain:** GcExcel templates cater to complex use-cases to create Excel reports for any scenario.

The GcExcel template is a pre-defined and formatted workbook. It can be used in the creation of final reports. In the following sections, you will find GcExcel templates used in three diverse use-cases.

### • Use Case 1 - Financial Statistics Report

In this use-case, we have created a Financial dashboard template to show the Budget statistics of different countries in different seasons or quarters. Here, in the template, the cell A1 contains the title of the template, and the cell D1 contains the year gap for which the financial data has been recorded. Note that here 'ds' is the data source that will populate the country names and quarterly seasons in the Excel report. The names such as BUDGET STATISTICS, BUD and ACT are other data fields of ds. The country field is expanded horizontally to add other countries. Various function fields are used to perform calculations on the Budget and Actual columns.



	A	B	C	D	E
1	FINANCIAL DASHBOARD			2016-2019	
2					
3		BUDGET STATISTICS{{{E = H}}}			
4		{{ds.country(E = H, S = None)}}			
5		BUD{{{C = B4}}}	ACT{{{C = B4}}}		
6	{{ds.season}}	{{ds.expect(C = B4*A6)}}	{{ds.actual(C = B4*A6)}}		
7		{{=SUM(B6)(C = B4)}}	{{=SUM(C6)(C = B4)}}		
8		TOTAL BUD	{{=SUM(B7)}}		
9		TOTAL ACTUAL	{{=SUM(C7)}}		
10					
11					

The Excel report generated from the Financial Dashboard template is given below:

	A	B	C	D	E	
1	FINANCIAL DASHBOARD			2016-2019		
2						
3						
4		USA		Japan		
5		BUD	AST	BUD	AST	
6	2016 Q1	\$ 2,36,047	\$ 3,28,554	\$ 3,50,156	\$ 3,70,834	\$
7	2016 Q2	\$ 3,73,060	\$ 2,38,136	\$ 3,69,399	\$ 2,47,324	\$
8	2016 Q3	\$ 2,24,132	\$ 3,00,822	\$ 2,78,834	\$ 2,37,385	\$
9	2016 Q4	\$ 2,69,305	\$ 3,15,337	\$ 2,64,277	\$ 2,45,048	\$
10	2017 Q1	\$ 2,65,397	\$ 2,79,008	\$ 2,03,006	\$ 2,95,389	\$
11	2017 Q2	\$ 2,14,079	\$ 2,06,019	\$ 2,76,987	\$ 2,15,804	\$
12	2017 Q3	\$ 3,70,191	\$ 2,38,294	\$ 3,30,315	\$ 3,30,443	\$
13	2017 Q4	\$ 2,66,843	\$ 2,42,323	\$ 3,07,477	\$ 2,62,512	\$
14		\$ 22,19,054	\$ 21,48,493	\$ 23,80,451	\$ 22,04,739	\$
15		TOTAL BUD	\$ 1,08,60,998			
16		TOTAL ACTUAL	\$ 1,12,50,382			
17						
18						
19						

- **Use Case 2 - Department & Budget Report**

In this template, the budget of each department is depicted based on the salaries of employees in that department. Here, 'ds' is the data source and it populates data fields with the names of departments, managers and employees. The department data fields are expanded horizontally to add more departments. Note that in each department, the static text fields 'Employee' and 'Salary' remains the same. The image below shows the budget report for two departments, Marketing and Sales.

B	C	D	
	<b>{{ds.dpt.name(E=H, R=C2:D6, G=list)}}</b>		
	MANAGER	{{ds.dpt.mgr}}	
	BUDGET	{{ds.dpt.bud}}	
	<b>Employee</b>	<b>Salary</b>	
	{{ds.dpt.emp.name(G=list)}}	{{ds.dpt.emp.salary}}	


The Excel report generated from the Department & Budget template is given below:

C	D	E	F
<b>Marketing</b>		<b>Sales</b>	
MANAGER	Carl Sommerset	MANAGER	Kelly Johnson
BUDGET	\$ 3,54,586	BUDGET	\$ 2,37,721
<b>Employee</b>	<b>Salary</b>	<b>Employee</b>	<b>Salary</b>
JoeKline	\$ 49,402	Liam Elmerston	\$ 61,892
Lisa Crane	\$ 81,337	Angela Sanderson	\$ 38,020
John Ryes	\$ 43,503	Blake Schwarz	\$ 55,701
Elli Davidson	\$ 67,334	Linda Barataz	\$ 82,108
Jack Reaze	\$ 68,314		
Ben Lam	\$ 44,696		


- **Use Case 3 - Sales Report**

This use case depicts a template for recording the E-commerce sales of electronic goods in different areas of a country. The data source used here is ds, and it populates the data in the final Excel report with categories, names, cities, sales etc.

The Excel report generated in this case displays the sales of electronic goods individually as well as with respect to their categories. The area and cities are expanded horizontally due to their expansion property. Various function fields are used to perform calculations on the sales and revenue numbers. The value in cell D14 is calculated, first by summing up the revenue in cell C14 and then summing up the values of the whole category (as A14 is its context)

	A	B	C	D
5				
6	Quarterly Results	Q4 Sales		
7				
8				
9	Business Name:	E-Commerce		
10				
11	Sales		Area {{{E=H}}}	Category's Sales
12			{{ds.Area(E=H)}}	
13			{{ds.City(E=H)}}	
14	{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C14)(C=A14)}}
15	Region's Sales		{{=Sum(C14)(C=C12)}}	{{=Sum(C15)}}
16				

The Excel report generated from the Sales template is given below:

Quarterly Results		Q4 Sales					
Business Name:		E-Commerce					
Sales		Area				Category's Sales	
		North America					
		Chicago	Minnesota	Medillin	Quito		
Consumer Electronics	Bose 785593-0050	\$92,800.00				\$42,06,891.00	
	Canon EOS 1500D	\$98,650.00	\$89,110.00				
	Haier 394L 4Star	\$3,67,050.00			\$7,29,100.00		
	IFB 6.5 Kg FullyAuto			\$82,910.00			
	Mi LED 40inch	\$5,50,010.00	\$17,84,702.00				
	Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00		
Mobile	Iphone XR		\$17,34,621.00			\$44,19,531.00	
	OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00		
	Redmi 7		\$81,650.00		\$2,76,390.00		
	Samsung S9		\$8,96,250.00		\$7,16,520.00		
Region's Sales		\$63,72,043.00		\$22,54,379.00		\$86,26,422.00	

## Template Configuration

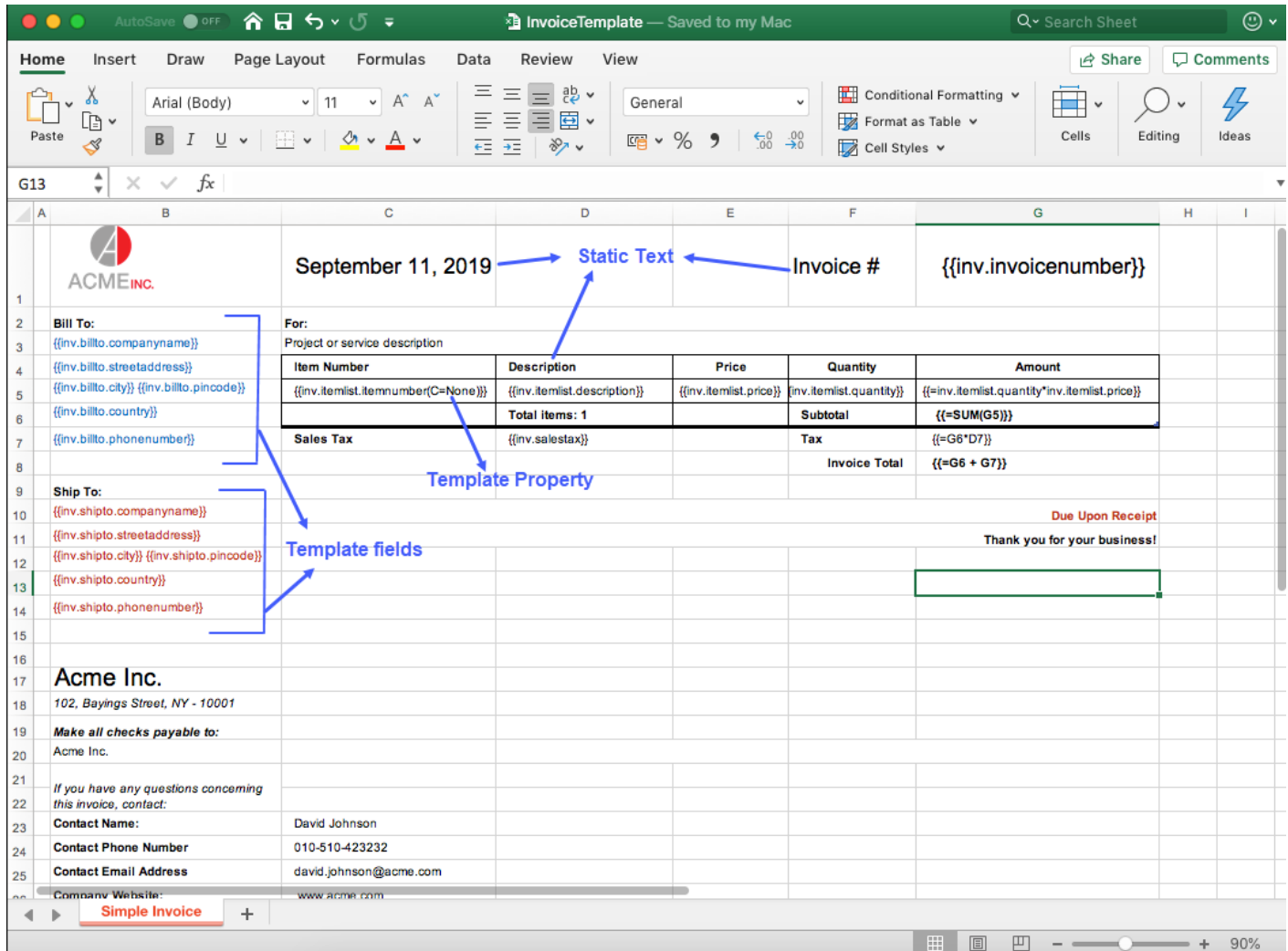
Template configuration includes all the features, fields and properties of Templates in GcExcel.

The first step to configure a template is to create a template layout in Excel, which is a pre requisite to generate Excel reports. This layout defines the outline of how the final report will look like and can include static text, data bound fields and other template properties.

Except static fields, all other fields follow syntax and are defined in mustache braces {{ }}. These fields can also include template properties like group, sort, pagebreak etc which are applied on the final Excel report when populated from the data, in datasource.

Apart from static text, a template layout in Excel is comprised of:

- [Template Fields](#)
- [Template Properties](#)



**Note:** The Excel formulas applied in template layout can be exported as formulas in Excel reports instead of just the cell values. The formula and its range can be viewed in the formula bar of Excel report by selecting the cell to which it has been applied. The Excel formula can be exported by using this syntax in template layout: **{{= formula }}**

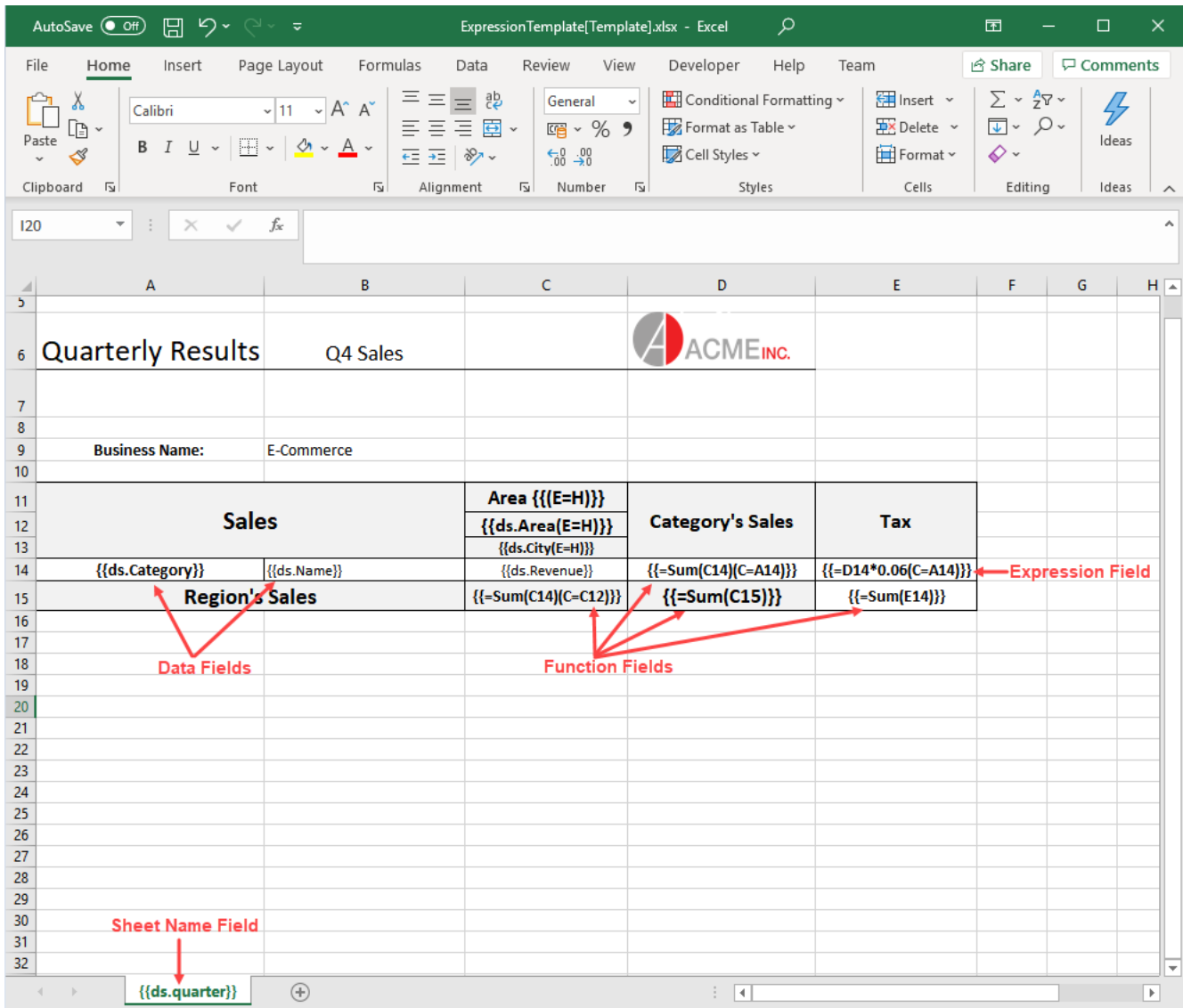
For example:

Lets say, **{{=SUM(A5)}}** formula is applied in a template layout. Now, in the generated Excel, formula displayed on clicking the formula cell will be **SUM(A5:A10)**, meaning that this is the range on which the formula is applied.

The formula must be syntactically correct and refer to the right range while defining in the template layout.

## Template Fields

A template layout can contain various data bound fields whose data is bound to the datasource. The below image shows different template fields:



## Data Fields

Data fields are the bound fields which are populated by the data in data source. These fields can be defined in different ways as shown in examples below:


- **Data field**

The data bound fields are defined as: `{{ds.FieldName}}`, where `ds` is the alias of the datasource. For example, `{{ds.grade}}`

- **Nested data field**

If the data is nested, you might want to arrange report as per an inner object field, it can be defined using nested data fields with the following syntax by separating the nested objects with a `'`



Student Report	
Student Name	{{report.student.name(R=A3:B9)}}
Father's Name	{{report.student.family.father.name(S=None,E=H)}}
Occupation	{{report.student.family.father.occupation(E=H)}}
Mother's Name	{{report.student.family.mother.name(E=H)}}
Occupation	{{report.student.family.mother.occupation(E=H)}}

- **Inline data field**

It is defined along with the constant text in a cell. For example, Date: {{task.dueDate}}

Also, you can define multiple inline data fields from different data sources as shown below:

Employee Name	Employee ID	Department	Department HOD Name	Department ID	Department Location
Hello: {{emp.name}}	{{emp.id(R=A4:F4, S=None)}}	Belongs to {{emp.department}} of our company	{{department.name.hodname}}	{{emp.department.id}}	{{emp.department.loc}} in US

## Function Fields

Function fields are used to perform calculations in your reports. A function can be applied over a cell or a data field. The standard Excel functions which are supported in the function field are Sum, Count, Average, Max, Min, Product, StdDev, StdDevp, Var and Varp. For example:

{{=SUM(F4)}}


{{=SUM(team.score)}}


{{=Count(student.name)}}



**Note:** Function field supports only one parameter

The function fields can also be calculated over a context. for example, in the below image cell D14 contains function field as well as the context property. The resultant value will be calculated, first by summing up the revenue in cell C14 and then summing up the values of the whole category (as A14 is its context)

	A	B	C	D
5				
6	Quarterly Results	Q4 Sales		
7				
8				
9	Business Name:	E-Commerce		
10				
11	Sales		Area {{{E=H}}}	Category's Sales
12			{{ds.Area(E=H)}}	
13			{{ds.City(E=H)}}	
14	{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C14)(C=A14)}}
15	Region's Sales		{{=Sum(C14)(C=C12)}}	{{=Sum(C15)}}
16				

Quarterly Results	Q4 Sales				
Business Name:	E-Commerce				
Sales		Area			
		North America			
		Chicago	Minnesota	Medillin	Quito
Consumer Electronics	Bose 785593-0050	\$92,800.00			
	Canon EOS 1500D	\$98,650.00	\$89,110.00		
	Haier 394L 4Star	\$3,67,050.00			\$7,29,100.00
	IFB 6.5 Kg FullyAuto			\$82,910.00	
	Mi LED 40inch	\$5,50,010.00	\$17,84,702.00		
	Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00
Mobile	Iphone XR		\$17,34,621.00		
	OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00
	Redmi 7		\$81,650.00		\$2,76,390.00
	Samsung S9		\$8,96,250.00		\$7,16,520.00
Region's Sales		\$63,72,043.00		\$22,54,379.00	
				\$86,26,422.00	

### Expression Fields

Expression fields can be used to perform calculations using operators '+', '-', '\*', '/' and '()'. An expression can be applied over cells or data fields. For example:

```
{{= ds.count*ds.price}}
```

```
{{= ds.price + ds.tax}}
```

```
{{=A18*0.05}}
```

```
{{=A18+D18-G18}}
```

```
{{=(A18+ A20)*0.3}}
```

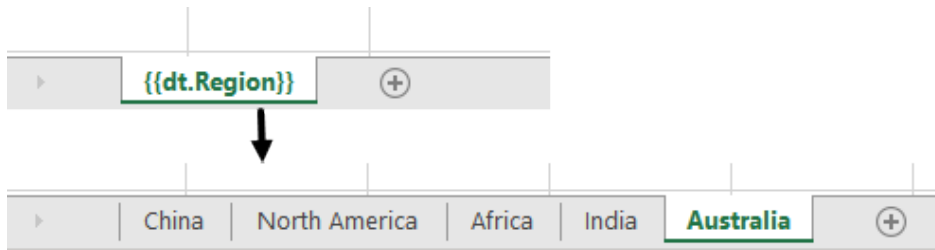


**Note: Template properties** are not supported in Expression fields. Function field and Expression field cannot be used together like `{{=Sum(A5+A6)}}` or `{{=Sum(A5)*0.01}}`, which is not supported.

### Sheet Name

GcExcel supports using bound field in sheet name, which means, the field value of sheet name is populated by the data in data source and multiple worksheets are created. Each worksheet contains data corresponding to its value.

For example, if we specify `{{dt.Region}}` as the sheet name and the data source contains data for 5 regions, the final report will consist of 5 worksheets and the individual sheet will contain data for a specific region.



**Note:** Only the Sort and Group [template properties](#) are supported for sheet name field.

## Template Properties

The template properties are defined along with template fields in round braces ( ) as can be seen in the below image:

The screenshot displays a GcExcel spreadsheet titled 'TravelTemplate'. The interface includes a ribbon with tabs: Home, Insert, Draw, Page Layout, Formulas, Data, Review, and View. The 'Home' tab is active, showing options for font (Calibri, size 12), bold, italic, underline, and text color. The spreadsheet content is organized into sections: 'CALIFORNIA' (a large teal header), 'TRANSPORT' (a table with columns: Carrier, Flight No., Date, From, Departure Time, To, Arrival Time, Reservation No.), 'LODGING' (a table with columns: Accommodations, Date, Concierge, Phone/Email, Address, Confirm. No., Days, Total Cost), 'Emergency Contacts' (a table with columns: Contact, Phone, Notes), and 'Travel Party' (a table with columns: Contact, Phone, Notes). Several cells contain template fields in round braces, such as `{{ds1.Carrier(G = list, S = None)}}`, `{{ds1.FlightNo}}`, `{{ds1.Date}}`, `{{ds1.From}}`, `{{ds1.DepartureTime}}`, `{{ds1.To}}`, `{{ds1.ArrivalTime}}`, `{{ds1.ReservationNo}}`, `{{ds2.Accommodations(G = list, S = None)}}`, `{{ds2.Date}}`, `{{ds2.Concierge}}`, `{{ds2.Phone}}`, `{{ds2.AddressPart1}}`, `{{ds2.AddressPart2}}`, `{{ds2.ConfirmNo}}`, `{{ds2.Days}}`, `{{ds2.TotalCost}}`, `{{ds3.Contact(R=A21:C21, G = list, S = None)}}`, `{{ds3.Phone}}`, `{{ds4.Contact(G = list, S = None)}}`, `{{ds4.Phone}}`, `{{ds4.Note(C=F21)}}`, and `{{=SUM(H14)}}`. Some of these fields are highlighted with blue boxes.

### Cell Context

The cell context property defines the relationship between cells depending on which the cells are grouped or filtered.

**Value:** Cell location or Data field

*Custom:* Cell context must be specified explicitly.

*Default* (default value): The adjacent cell on the left with E=V, or the adjacent cell on the top with E= H.

*None:* The cell has no context.

## Example

```
{{ds.field(C=A1, E=H)}}
```

```
Hello World! {{{C=A2}}}
```

```
{{=SUM(F4) (C=ds1.team)}}
```

```
{{=SUM(ds1.score) (C=ds1.team*ds1.season)}}
```

For more information about Cell Context, refer [Cell Context](#) topic.

## Cell Expansion

The cell expansion property describes the direction in which the cell values will expand.

**Value:** Enum

*E=N* (None)

*E=H* (Horizontal): Cell data is expanded from left to right.

*E=V* (Vertical-Default value): Cell data is expanded from top to bottom.

## Example

```
{{ds.field(C=A1, E=H)}}
```

For more information about Cell Expansion, refer [Cell Expansion](#) topic.

## Group

The group property allows you to group data in template.

**Value:** Enum

*G=Normal:* The group by field(s) value is not repeated for the corresponding records in the column; instead they are printed once per data group.

*G=Merge* (default value): The same behavior as for the normal parameter, except that it merges the cells in the group by field(s) for each group set.

*G=Repeat:* The group by field(s) value is repeated for the corresponding records.

*G=List:* The field(s) values are listed independently for the corresponding records.

## Example

```
{{ds.field(G=repeat)}}
```

```
{{ds.field(G=list)}}
```

The below image shows how to apply 'merge' grouping on repeating data:

Merge group(default)			Repeat group	
Team	Name		Team	Name
{{ds.Team}}	{{ds.Name}}		{{ds.Team(G=R)}}	{{ds.Name}}
Normal group			List group	
Team	Name		Team	Name
{{ds.Team(G=N)}}	{{ds.Name}}		{{ds.Team(G=L)}}	{{ds.Name}}

Merge group(default)			Repeat group	
Team	Name		Team	Name
New York	Hellen		New York	Hellen
	Hunter		New York	Hunter
	Jim		New York	Jim
	Phillip		New York	Phillip
Seattle	Bob		Seattle	Bob
	Jaguar		Seattle	Jaguar
	Tommy		Seattle	Tommy
Normal group			List group	
Team	Name		Team	Name
New York	Hellen		Seattle	Bob
	Hunter		Seattle	Tommy
	Jim		Seattle	Jaguar
	Phillip		New York	Phillip
Seattle	Bob		New York	Hunter
	Jaguar		New York	Hellen
	Tommy		New York	Jim

## Range

The range property specifies the fallback context for the fields in specified range. All the fields that are covered in the range which have no default nor explicit context, use the current cell in which the range is defined, as their context.

**Value:** Cell range

Default value: Null

### Example

```
{{ds.field(R= B3:F10)}}
```

The below image shows that the range is defined for a student name, specifying that the details will expand and group with respect to Student name:

Sales			
<b>Name:</b>	{{ds.Name(R=B11:F116)}}	<b>Revenue:</b>	{{ds.Revenue}}
<b>Area:</b>	{{ds.Area}}		
<b>City:</b>	{{ds.City}}		

↓

Sales			
<b>Name:</b>	Bose 785593-0050	<b>Revenue:</b>	\$ 92,800.00
<b>Area:</b>	North America		
<b>City:</b>	Chicago		

## Sort

The sort property specifies the type of sorting in template.

**Value:** Enum

*S=Asc* (default value): Ascending

*S=Desc* : Descending

*S=None*: None

### Example

```
{{ds.field(S=Desc)}}
```

The below image shows how the template fields are expanded based on their sorting type:

Sort Ascending(default)		Sort Descending		None sort
Name		Name		Name
{{ds.Name}}		{{ds.Name(S=desc)}}		{{ds.Name(S=none)}}

↓

Sort Ascending(default)		Sort Descending		None sort
Name		Name		Name
Bose 785593-0050		Sennheiser HD 4.40-BT		Bose 785593-0050
Canon EOS 1500D		Samsung S9		Canon EOS 1500D
Haier 394L 4Star		Redmi 7		Haier 394L 4Star
IFB 6.5 Kg FullyAuto		OnePlus 7Pro		IFB 6.5 Kg FullyAuto
Iphone XR		Mi LED 40inch		Mi LED 40inch
Mi LED 40inch		Iphone XR		Sennheiser HD 4.40-BT
OnePlus 7Pro		IFB 6.5 Kg FullyAuto		Iphone XR
Redmi 7		Haier 394L 4Star		OnePlus 7Pro
Samsung S9		Canon EOS 1500D		Redmi 7
Sennheiser HD 4.40-BT		Bose 785593-0050		Samsung S9

## Page Break

The page break property specifies whether to add a new page after a field or not.

- If the template cell is located in the first column, horizontal page break is added.
- If the template cell is located in the first row, vertical page break is added
- If the template cell is located in any other location than the first row and first column, both a horizontal and a vertical page break is added

**Value:** Boolean

*Pagebreak=True*


*Pagebreak=False* (Default value)

## Example

```
{{ds.field(Pagebreak=true)}}
```

The below image shows that a page break will be added after 'Occupation' field:

Category	Name
{{ds.Category(Pb=true)}}	{{ds.Name}}



Category	Name
Consumer Electronics	Bose 785593-0050
	Canon EOS 1500D
	Haier 394L 4Star
	IFB 6.5 Kg FullyAuto
	Mi LED 40inch
	Sennheiser HD 4.40-BT
Mobile	Iphone XR
	OnePlus 7Pro
	Redmi 7
	Samsung S9

## Image

The image property specifies whether to add an image or not and if yes, its height and width can also be specified.

The supported image data type is byte[] and base64 string.

The position of image in the cell can be controlled by setting the horizontal and vertical alignment style of cell. By default, the image is located in the center of the cell horizontally and vertically, both.


**Value:** Boolean

*Image = True*

*Image = False* (Default value)

Image.width=String value: Default value is cell width.

Image.height=String value: Default value is cell height.

 **Note:** Image property should be set to true in order to set its width or height.

## Example

```
{{ds.icon(Image=true)}}
```

```
{{ds.icon(Image=true, Image.width=150px)}}
```

```
{{ds.icon(Image=true, Image.height=150px)}}
```

The below image shows how an image can be added in the Excel report:





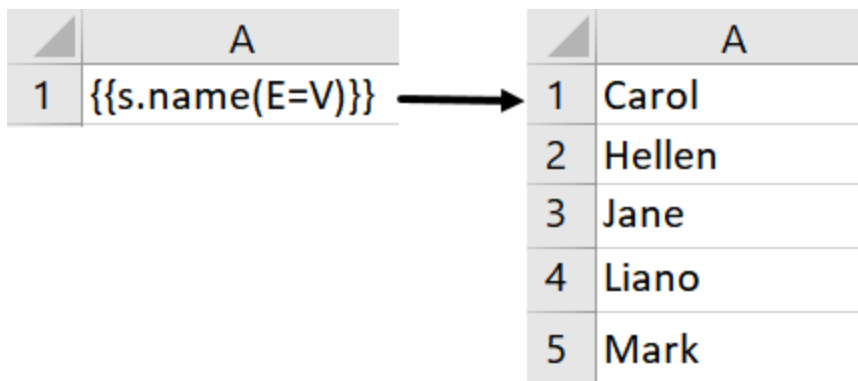
## Cell Expansion

The layout of a template in Excel consists of various fields, some of which are bound to a data source. The value of a bound field in a template, expands to several cells in report. For example, if you have created a field named 'Color' and bound it to the data source which contains 10 values for 'Color', the cell will expand to 10 values.

The expansion of a cell depends on the rules explained below:

### Vertical Expansion

The cell values will expand vertically if the expansion property of the cell is set to vertical, that is, "**E=V**", as shown below. The default expansion setting is vertical, which means if you do not specify any expansion property in the cell, the cell values will expand vertically.

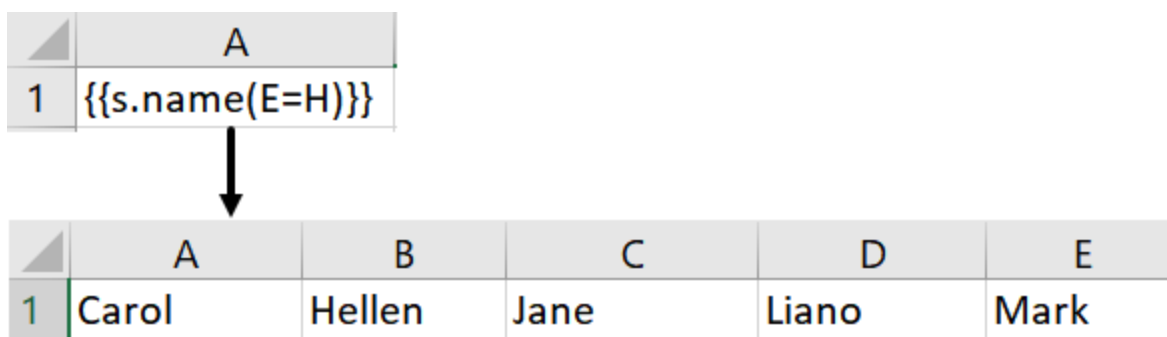


	A
1	{{s.name(E=V)}}

1	Carol
2	Hellen
3	Jane
4	Liano
5	Mark

### Horizontal Expansion

The cell values will expand horizontally if the expansion property of the cell is set to horizontal, that is, "**E=H**", as shown below:



	A
1	{{s.name(E=H)}}

	A	B	C	D	E
1	Carol	Hellen	Jane	Liano	Mark

## Cell Context

A template layout can contain multiple bound fields which depend on each other while expansion in the final Excel report.

For example, in the below image, the 'team' and 'name' are two bound fields in the template layout where team is the former cell and name is the latter cell. Now, the 'name' field will depend on the 'team' field to group or filter its values based on the team. Also, the direction of expansion of the 'name' field will be decided by the 'team' field. Here, team is the context cell of name.

	A	B
1	{{s.team}}	{{s.name}}

### Context Relationships

When multiple fields bound to a data source are defined in a template layout, a relationship is established between them which is called 'Context' relationship. The former cell is called the context cell of latter cell. Based on this relationship, the data is filtered or grouped while expansion in the final report.

There are two types of context relationships:

- Filtering Relationship:** The data in the cell is filtered using data of the context cell as the filter condition. For example, in the below image, the data in the 'name' cell is filtered corresponding to the data in its context cell:

	A	B
1	{{s.team}}	{{s.name}}

→

	A	B
1		Carol
2	Avengers	Hellen
3		Jane
4		Liano
5	Dominators	Mark

- Following Relationship:** The data in the cell is grouped according to the expansion direction of the data in context cell. For example, in the below image, the data in the 'name' cell is grouped and expanded horizontally depending on its context cell:

	A
1	{{s.team(E=H)}}
2	{{s.name}}

→

	A	B
1	Avengers	Dominators
2	Carol	Jane
3	Hellen	Liano
4		Mark
5		

### Context Cell

The context of a cell is defined using the 'C' property. The data in cells expand vertically or horizontally depending on their context. A cell's context can be set in the below ways:

- **None:** No cell context (C= None)

	A		A	B
1	{{s.team(E=H)}}	→	1	Avengers
2	{{s.name(C=None)}}		2	Carol
			3	Hellen
			4	Jane
			5	Liano
			6	Mark

- **Custom:** The cell context is specified explicitly using 'C' property

	A	B		A	B
1	{{s.team}}		→	1	Avengers
2				2	
3		{{s.name(C=A1)}}		3	Carol
				4	Hellen
				5	Dominators
				6	
				7	Jane
				8	Liano
				9	Mark

- **Default:** If no context is defined in the cell, the default context cell is the adjacent cell on the left with E=V (expanding vertically), or adjacent cell on the top with E= H (expanding horizontally)

For example, in the below image, A1 is the context cell of B1 and expands vertically.

	A	B
1	{{s.team}}	{{s.name}}

And, A1 is the context cell of A2 and expands horizontally.

	A
1	{{s.team(E=H)}}
2	{{s.name}}

### Context Precedence

The priority order in which context should be applicable on a cell is determined in the following order:

**Explicit context > Default Context > Fallback context**

- **Explicit context:** The context defined by **C** property in the cell itself
- **Default context:** If no context is defined in the cell, the default context is given priority
- **Fallback context:** If there is no adjacent cell value on the left or top, the cell looks for a cell with **R** (Range) property which covers its location, and use it as its context.

The **Fallback context** can be defined in a cell using the Range property, in case no default or explicit context is defined. The cell that defines the range is followed as a context for other cells to expand.

For example, the below template layout is created to display the sales details for different camera models, which means the data needs to expand with respect to the model of the camera. Then after a break, the sales details need to be displayed for another model of the camera. Instead of adding context to every field, we can define the range R=B11:F16 for Camera model - {{ds.Name (R=B11:F16)}}, stating that the sales details need to expand and group with respect to the camera model.

E-Commerce				
<b>Sales</b>				
<b>Name:</b>	{{ds.Name(R=B11:F16)}}		<b>Revenue:</b>	{{ds.Revenue}}
<b>Area:</b>	{{ds.Area}}			
<b>City:</b>	{{ds.City}}			

The above template layout will generate the following Excel report:

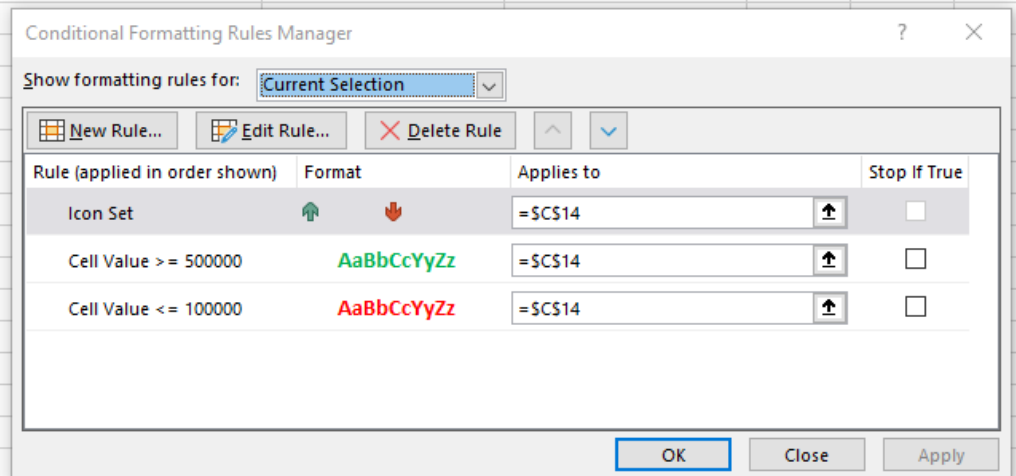
E-Commerce				
<b>Sales</b>				
<b>Name:</b>	Bose 785593-0050	<b>Revenue:</b>	\$	92,800.00
<b>Area:</b>	North America			
<b>City:</b>	Chicago			
<b>Sales</b>				
<b>Name:</b>	Canon EOS 1500D	<b>Revenue:</b>	\$	98,650.00
<b>Area:</b>	North America			
<b>City:</b>	Chicago			

## Conditional Formatting

Conditional formatting rules can be defined in Template layout which are applied to the expanded cells in Excel report.

For example, the below template layout applies a conditional formatting rule in data field: {{ds.Sales}}. The rule specifies to show the cell value in red if it is less than 200 and in green if it is equal to or greater than 200. Also, the icons are displayed alongside cell values.

<b>Business Name:</b>	E-Commerce			
<b>Sales</b>		<b>Area {{{E=H}}}</b>	<b>Category's Sales</b>	
		<b>{{ds.Area(E=H)}}</b>		
		<b>{{ds.City(E=H)}}</b>		
<b>{{ds.Category}}</b>	<b>{{ds.Name}}</b>	<b>{{ds.Revenue}}</b>	<b>{{=Sum(C14)(C=A14)}}</b>	
<b>Region's Sales</b>		<b>{{=Sum(C14)(C=C12)}}</b>	<b>{{=Sum(C15)}}</b>	



When the template is processed, the conditional formatting rule is applied to the expanded data in the final Excel report as shown below:

Business Name:		E-Commerce					
Sales		Area				Category's Sales	
		North America					
		Chicago	Minnesota	Medillin	Quito		
Consumer Electronics	Bose 785593-0050	⬇️ \$92,800.00				\$42,06,891.00	
	Canon EOS 1500D	⬇️ \$98,650.00	⬇️ \$89,110.00				
	Haier 394L 4Star	\$3,67,050.00			⬆️ \$7,29,100.00		
	IFB 6.5 Kg FullyAuto			⬇️ \$82,910.00			
	Mi LED 40inch	⬆️ \$5,50,010.00	⬆️ \$17,84,702.00				
	Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00		
Mobile	Iphone XR		⬆️ \$17,34,621.00			\$44,19,531.00	
	OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00		
	Redmi 7		⬇️ \$81,650.00		\$2,76,390.00		
	Samsung S9		⬆️ \$8,96,250.00		⬆️ \$7,16,520.00		
Region's Sales		\$63,72,043.00		\$22,54,379.00		\$86,26,422.00	

### Limitation

If the formula reference in a conditional formatting rule refers to a template cell, it is not handled correctly by GcExcel Template. The formula is not adjusted after template processing, that is, the formula will remain the same and will not get updated dynamically with the range.

For example:

If cell B5 has a formula reference in conditional formatting rule "=\$A\$5 > 100". And both A5 and B5 are template cells then after the template processing, conditional formatting rule may be applied from B5:B10 but, it "=\$A\$5 > 100" will not change dynamically with the cell range.


## Global Settings

Global settings, in GcExcel Templates, are the settings which when defined are applied throughout the template. These settings save lots of effort when same properties need to be applied on several fields. Global settings can be applied in all the template layouts and even in multiple worksheets of a workbook.

The global settings provided by GcExcel template are explained below:

Global Settings	Description	Value
TemplateOptions.KeepLineSize	It specifies whether the row height and column width should be kept the same throughout the template	<b>Type:</b> Boolean <b>Value:</b> True False (Default Value)
TemplateOptions.InsertMode	It specifies whether to insert extra cells or entire rows and columns when extra space is needed while expanding the template	<b>Type:</b> String <b>Value:</b> Cells (Default Value) EntireRowColumn

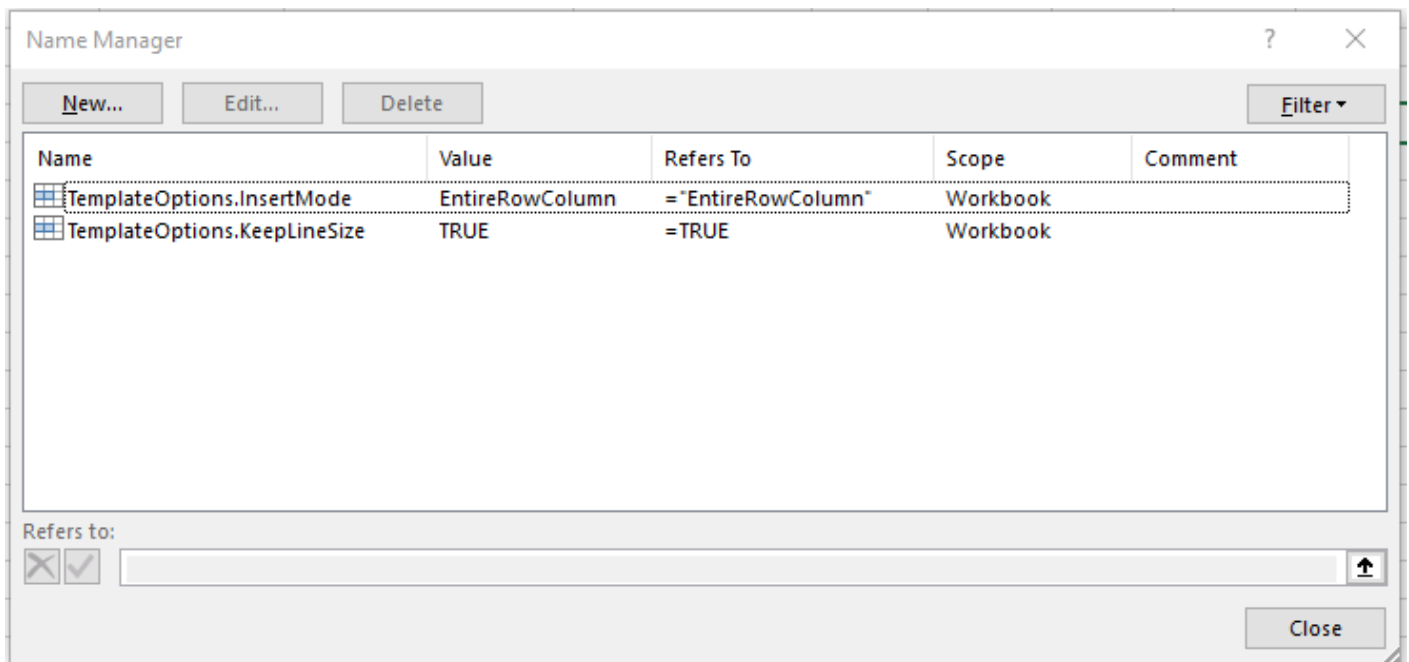
TemplateOptions.EmbedFontForFormFields	It specifies whether the fonts used by form fields should be embedded in exported PDF file.  <b>Note:</b> This setting is only applicable for <a href="#">PDF form fields</a>	<b>Type:</b> Boolean <b>Value:</b> True (Default Value) False
--	---	---

 **Note:** The scope of global settings is within a workbook only, which means, that all the worksheets in a workbook will apply the global settings.

The global settings can be applied in GcExcel template by using either of the two ways explained below:

### Define Global Settings in Template Layout

Global settings can be defined in Template layout in Excel in 'Name Manager' dialog box as shown below. The 'Name Manager' can be accessed by navigating through Formulas tab > Defined Names group, and then clicking the 'Name Manager'.



### Set Global Settings using Code

The global settings can be defined in GcExcel after loading the Excel template by using built-in workbook defined names **TemplateOptions**. The **Add** method of **INames** interface can be used to apply the global settings. The method takes **Name** and **RefersTo** properties as the parameters:

The value of **Name** property in built-in defined name is taken as the template global option's name. It is case-sensitive.

The value of **RefersTo** property in built-in defined name is taken as the template global option's value. It is case-sensitive.

Refer to the below example code to specify the global settings in template:

```
Java
```



```

Workbook workbook = new Workbook();
workbook.open("template.xlsx");

//Init template global settings
workbook.getNames().add("TemplateOptions.KeepLineSize", "true");
workbook.getNames().add("TemplateOptions.InsertMode", "EntireRowColumn");
//Global setting for PDF form fields
workbook.Names.Add("TemplateOptions.EmbedFontForFormFields", "true");

//Add data source
workbook.addDataSource("ds", ds);

//Invoke to process the template
workbook.processTemplate();


workbook.save("report.xlsx");

```

This template example records the E-commerce sales of electronic goods in different areas of a country. You can also download the **Excel template layout ('Sales[Template].xlsx' in the on-line documentation).**

### KeepLineSize

The below image shows the Excel report when 'TemplateOptions.KeepLineSize' is set to true.


	A	B	C	D	E	F	G
5							
6	Quarterly Results	Q4 Sales					
7							
8							
9	Business Name:	E-Commerce					
10							
11	Sales		Area				Category's Sales
North America							
Chicago			Minnesota	Medillin	Quito		
14	Consumer Electronics	Bose 785593-0050	\$92,800.00				\$42,06,891.00
15		Canon EOS 1500D	\$98,650.00	\$89,110.00			
16		Haier 394L 4Star	\$3,67,050.00			\$7,29,100.00	
17		IFB 6.5 Kg FullyAuto			\$82,910.00		
18		Mi LED 40inch	\$5,50,010.00	\$17,84,702.00			
19		Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00	
20	Mobile	Iphone XR		\$17,34,621.00			\$44,19,531.00
21		OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00	
22		Redmi 7		\$81,650.00		\$2,76,390.00	
23		Samsung S9		\$8,96,250.00		\$7,16,520.00	
24	Region's Sales		\$63,72,043.00		\$22,54,379.00		\$86,26,422.00
25							
26							
27							
28							
29							
30							

### InsertMode

The below image shows the Excel report when 'TemplateOptions.InsertMode' is set to EntireRowColumn. By doing this, the row height and outline groups of the rows are retained when the template expands.

1

2

	A	B	C	D	E	F	G
5							
6	Quarterly Results	Q4 Sales					
7							
8							
9	Business Name:	E-Commerce					
10							
11	Sales		Area				Category's Sales
North America							
Chicago			Minnesota	Medillin	Quito		
14	Consumer Electronics	Bose 785593-0050	\$92,800.00				\$42,06,891.00
15		Canon EOS 1500D	\$98,650.00	\$89,110.00			
16		Haier 394L 4Star	\$3,67,050.00			\$7,29,100.00	
17		IFB 6.5 Kg FullyAuto			\$82,910.00		
18		Mi LED 40inch	\$5,50,010.00	\$17,84,702.00			
19		Sennheiser HD 4.40-BT	\$1,78,100.00			\$2,34,459.00	
20	Mobile	Iphone XR		\$17,34,621.00			\$44,19,531.00
21		OnePlus 7Pro	\$4,99,100.00			\$2,15,000.00	
22		Redmi 7		\$81,650.00		\$2,76,390.00	
23		Samsung S9		\$8,96,250.00		\$7,16,520.00	
24	Region's Sales		\$63,72,043.00		\$22,54,379.00		\$86,26,422.00
25							
26							
27							
28							

### EmbedFontForFormFields

This setting allows you to embed font files used by form fields in the PDF document generated by GcExcel.

When true, any arbitrary character is displayed correctly even if your machine or browser does not have corresponding fonts installed. However, it may generate large sized PDF documents, especially when East Asian characters are used. When false, the generated PDF document is of optimal size but messy code will be displayed if your machine or browser does not have corresponding fonts installed.

The below image shows the PDF form generated by GcExcel when 'TemplateOptions.EmbedFontForFormFields' is set to True. You can also download the **Excel template layout ('EmbedFontForFormFields[Template].xlsx' in the on-line documentation)** used in below example.

(R1.5HP用)

## 新規登録申請書

日本小型船舶検査機関

この申請書フォームには入力できます。

申請者（新所有者等）

〒

住所：

（フリガナ）

氏名又は名称：

印

## Fixed Layout

When GcExcel processes a template layout, it inserts blank lines first and then sets the data and style to generate the final report. In cases where a fixed layout is defined in the template, GcExcel provides two properties you can use to properly load the data in this fixed layout area.

## 1. fillMode(FM) Property


The fillMode property can be set to 'overwrite' to set the data directly in template cells (without inserting blank rows). The style of the template layout is retained and data is filled into it.

The below template example records the E-Commerce sales of electronic goods in different areas of a country and uses 'overwrite' fillMode property. You can also download the **Excel template layout** ('SetFillModeOverwrite[Template].xlsx' in the on-line documentation) from here.

Business Name:	E-Commerce		
Area	City	Name	Sales
{{ds.Area(G=list, FM=overwrite)}}	{{ds.City}}	{{ds.Name}}	{{ds.Revenue}}
Region's Sales			\$0.00

The data in data source contains 8 rows. After GcExcel processes the template layout, the Excel report will look like below:


<b>Business Name:</b>	E-Commerce		
<b>Area</b>	<b>City</b>	<b>Name</b>	<b>Sales</b>
North America	Chicago	Bose 785593-0050	\$92,800.00
North America	Chicago	Haier 394L 4Star	\$3,67,050.00
South America	Santiago	Haier 394L 4Star	\$5,78,900.00
South America	Medillin	IFB 6.5 Kg FullyAuto	\$82,910.00
North America	Chicago	Sennheiser HD 4.40-BT	\$1,78,100.00
South America	Quito	OnePlus 7Pro	\$2,15,000.00
North America	Minnesota	Redmi 7	\$81,650.00
South America	Quito	Samsung S9	\$7,16,520.00
<b>Region's Sales</b>			

 **Note:** If fillMode property is not defined in the template layout, the default behavior is followed, which is to insert the blank lines first.

## 2. fillRange(FR) Property

The fillRange property should be used when fillMode is set to overwrite and the data in data source exceeds the area of fixed layout in template. So if the data overflows, the range defined by fillRange property is duplicated to fill additional data.

For example: If the data in data source contains 20 rows and fillRange property defines the cell range as A1:A8. The range will be duplicated to 8 more rows (total 16 rows) and then again to 8 more rows (total 24 rows), to fill the complete data of 20 rows.

 **Note:** When data in data source overflows and fillRange property is missing, range will not be duplicated to set data. Instead, data will be filled beneath the range area in existing rows.

The below template example records the E-Commerce sales of electronic goods in different areas of a country. It uses 'overwrite' fillMode property along with fillRange property to accommodate the additional data. You can also download the **Excel template layout ('SetFillModeOverflow[Template].xlsx' in the on-line documentation)** used in below example.

9	<b>Business Name:</b>	E-Commerce		
10				
11	<b>Area</b>	<b>City</b>	<b>Name</b>	<b>Sales</b>
12	{{ds.Area(G=list, FM=overwrite, FR=A12:D23)}}	{{ds.City}}	{{ds.Name}}	{{ds.Revenue}}
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24	<b>Region's Sales</b>			<b>\$0.00</b>

The data in data source contains 26 rows. The fillRange property defines cell range for 12 rows and it is duplicated after that twice to fill all the data. After GcExcel processes the template layout, the Excel report will look like below:


9	Business Name:	E-Commerce		
10				
11	Area	City	Name	Sales
12	North America	Chicago	Bose 785593-0050	\$92,800.00
13	North America	New York	Bose 785593-0050	\$92,800.00
14	South America	Santiago	Bose 785593-0050	\$19,550.00
15	North America	Chicago	Canon EOS 1500D	\$98,650.00
16	North America	Minnesota	Canon EOS 1500D	\$89,110.00
17	South America	Santiago	Canon EOS 1500D	\$4,59,000.00
18	North America	Chicago	Haier 394L 4Star	\$3,67,050.00
19	South America	Quito	Haier 394L 4Star	\$7,29,100.00
20	South America	Santiago	Haier 394L 4Star	\$5,78,900.00
21	North America	Fremont	IFB 6.5 Kg FullyAuto	\$9,04,930.00
22	South America	Buenos Aires	IFB 6.5 Kg FullyAuto	\$6,73,800.00
23	South America	Medillin	IFB 6.5 Kg FullyAuto	\$82,910.00
24	North America	Chicago	Mi LED 40inch	\$5,50,010.00
25	North America	Minnesota	Mi LED 40inch	\$17,84,702.00
26	South America	Santiago	Mi LED 40inch	\$1,02,905.00
27	North America	Chicago	Sennheiser HD 4.40-BT	\$1,78,100.00
28	South America	Quito	Sennheiser HD 4.40-BT	\$2,34,459.00
29	North America	Minnesota	Iphone XR	\$17,34,621.00
30	South America	Santiago	Iphone XR	\$1,09,300.00
31	North America	Chicago	OnePlus 7Pro	\$4,99,100.00
32	South America	Quito	OnePlus 7Pro	\$2,15,000.00
33	North America	Minnesota	Redmi 7	\$81,650.00
34	South America	Quito	Redmi 7	\$2,76,390.00
35	North America	Minnesota	Samsung S9	\$8,96,250.00
36	South America	Buenos Aires	Samsung S9	\$8,96,250.00
37	South America	Quito	Samsung S9	\$7,16,520.00
38				
39				
40				
41				
42				
43				
44				
45				
46				
47				
48	Region's Sales			

## PDF Form Builder

GcExcel Templates provide the ability to build PDF forms with various form fields using Excel as the designer. The form fields can be defined using the proper syntax while creating template layouts. After the template is processed, the result can be exported to a PDF document that includes the pre-defined form fields.


The **"form"** property can be used to define a PDF form field. The value of this property is in JSON format and a JSON string can be used to describe all settings of the form field. For example:

- `{{ds1.Name(form={"type": "textbox", "name": "username", "value": "Input your name!", "font":{"size":15, "color": "#ff0000", "bold": true}, "required": true}}}`
- `{{(form={"type": "listbox", "name": "cities", "value": ["Xi'An", "BeiJing"], "font":{"size":11, "color": "#ff00ff", "bold": true}, "required": true}}}`

 **Note:** The property name and enum values are case insensitive.

The following PDF form fields are supported:

- Check box
- Combo box
- List box
- Button
- Radio button
- Signature
- Text box

 **Note:** The form fields are visible only in PDF documents and not in Excel.

## Bound PDF Form

Consider an example for generating a bound PDF form by using GcExcel Templates. In this case, an address book is generated in PDF by defining textbox fields in template cells. The textbox fields are defined in a way that they relate to common details of an address book, like:

Name : `{{ds.Name(form={"type": "textbox", "name": "name", "font":{"color": "#000000", "bold": true}}}}}`

Email: `{{ds.Email(form={"type": "textbox", "name": "Email", "font":{"color": "#EC881D"}}}}}`

These textbox fields are bound fields, whose data is populated from the data source and is displayed in the PDF form after template processing. You can also download the **Excel template layout ('TextFields[Template].xlsx' in the on-line documentation)** from here.

## The Address Book

NAME	WORK	CELL	HOME	EMAIL	BIRTHDAY	ADDRESS
<code>{{ds.Name(form={"type": "textbox", "name": "name", "font":{"color": "#000000", "bold": true}}}}}</code>	<code>{{ds.Work(form={"type": "textbox", "name": "work", "font":{"color": "#000000", "bold": true}}}}}</code>	<code>{{ds.Cell(form={"type": "textbox", "name": "cell", "font":{"color": "#000000", "bold": true}}}}}</code>	<code>{{ds.Home(form={"type": "textbox", "name": "home", "font":{"color": "#000000", "bold": true}}}}}</code>	<code>{{ds.Email(form={"type": "textbox", "name": "email", "font":{"color": "#EC881D", "bold": true}}}}}</code>	<code>{{ds.Birthday(form={"type": "textbox", "name": "birthday", "font":{"color": "#000000", "bold": true}}}}}</code>	<code>{{ds.Address(form={"type": "textbox", "name": "address", "font":{"color": "#000000", "bold": true}}}}}</code>

After GcExcel processes the template and exports it to a PDF document, the PDF form will look like below:

## The Address Book

NAME	WORK	CELL	HOME	EMAIL	BIRTHDAY	ADDRESS	CITY	STATE	ZIP
Andrew Lepp	6235320178	6235320178	6235320178	Andrew@example.com	10/9/1996	123 N. Maple	Augusta	ME	04330
James Williams	5235550879	5235550879	5235550879	James@example.com	4/5/1995	123 N. Maple	Denver	CO	80214
John Smith	3215230123	3215230123	3215230123	John@example.com	5/20/1990	4456 E. Aspen	Montgomery	AL	36134
Kim Abercrombie	1235550123	1235550123	1235550123	Kim@example.com	4/13/1991	123 N. Maple	Cherryville	WA	98037
Mark Jordan	1238640185	1238640185	1238640185	Mark@example.com	12/13/1988	123 N. Maple	Boise	ID	83704

## Unbound PDF Form

Consider an example for generating an unbound PDF form by using GcExcel Templates. In this case, a wage and tax statement is generated in PDF by defining textbox and checkbox fields in template cells, like:

**Textbox field:**

```
{{(form={"type": "textbox", "name": "tips", "backgroundcolor": "#ffabab"})}}
```

**Checkbox fields:**

```
{{(form={"type": "checkbox", "name": "Retirement", "border": {"color": "#ff0000"})}}}
```

```
{{(form={"type": "checkbox", "name": "Statutory", "border": {"color": "#ff0000"})}}}
```

These fields are unbound fields, and their data should be filled directly in the PDF form after template processing. You can also download the **Excel template layout ('WageAndTaxStatement[Template].xlsx' in the on-line documentation)** from here.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U									
1																														
2	22222	OID	{{(form=	Employee's social security num					For Official Use Only <input type="checkbox"/> OMB No. 1545-0008																					
4	b Employer identification number (EIN)										1 Wages, tips, other compensation					2 Federal income tax withheld														
6	c Employer's name, address, and ZIP code										3 Social security wages					4 Social security tax withheld														
8											5 Medicare wages and tips					6 Medicare tax withheld														
10											7 Social security tips					8 Allocated tips														
12	d Control number										9 {{(form="type": "textbox", "name": "tips", "backg					10 Dependent care benefits														
13											11 Nonqualified plans					12a See instructions for box 12														
14	e Employee's first name and initial					Last name					13 Statutory employee Retirement plan Third-party sick pay					12b														
16											14 Other					12c														
18																12d														
20																														
22																														
23	f Employee's address and ZIP code																													
24	15 State Employer's state ID number					16 State wages, tips, etc.					17 State income tax					18 Local wages, tips, etc.					19 Local income tax					20 Locality name				
25																														
26																														
27																														
29	Form <b>W-2</b> Wage and Tax Statement										2020										Department of the Treasury—Internal Revenue Service For Privacy Act and Paperwork Reduction Act Notice, see the separate instructions.									
32	Copy A—For Social Security Administration. Send this entire page with																				Cat. No. 10134D									
33	Form W-3 to the Social Security Administration; photocopies are not																													
34																														
35																														

**Do Not Cut, Fold, or Staple Forms on This Page**

After GcExcel processes the template and exports it to a PDF document, the PDF form will look like below:

22222		VOID <input type="checkbox"/>		a Employee's social security number		For Official Use Only ► OMB No. 1545-0008	
b Employer identification number (EIN)				1 Wages, tips, other compensation		2 Federal income tax withheld	
c Employer's name, address, and ZIP code				3 Social security wages		4 Social security tax withheld	
				5 Medicare wages and tips		6 Medicare tax withheld	
				7 Social security tips		8 Allocated tips	
d Control number				9		10 Dependent care benefits	
e Employee's first name and initial		Last name		Suff		11 Nonqualified plans	
						12a See instructions for box 12	
						12b	
						12c	
f Employee's address and ZIP code		13 Statutory employee <input type="checkbox"/>		Retirement plan <input type="checkbox"/>		Third-party sick pay <input type="checkbox"/>	
15 State Employer's state ID number		16 State wages, tips, etc.		17 State income tax		18 Local wages, tips, etc.	
						19 Local income tax	
						20 Locality name	

Form **W-2** Wage and Tax Statement

2020

Department of the Treasury—Internal Revenue Service  
For Privacy Act and Paperwork Reduction Act Notice, see the separate instructions.

Copy A—For Social Security Administration. Send this entire page with Form W-3 to the Social Security Administration; photocopies are not acceptable.

Cat. No. 10134D

**Do Not Cut, Fold, or Staple Forms on This Page**

Various settings can be applied on form fields to enhance and customize their appearance. These are explained as below:

- The below table describes common settings which can be applied to any PDF form field.

Name	Value Type	Example	Description
<b>type</b>	Enum string: <ul style="list-style-type: none"><li>checkbox</li><li>textbox</li><li>listbox</li><li>combobox</li><li>radiobutton</li><li>pushbutton</li><li>signature&gt;</li></ul>	{"type": "listbox"}	Indicates the type of form field. (Mandatory field)
<b>alternateName</b>	String	{"alternateName": "The alt name"}	Displays text which is

			helpful for a user while filling in the form field. Tooltips appear when the pointer hovers briefly over the form field.	
backgroundcolor		String	<pre>{"backgroundcolor": "#ffff00"} {"backgroundcolor": "rgb(255, 178, 0)"} {"backgroundcolor": "rgba(188, 100, 0, 255)"}</pre> Indicates background color of the form field.	
border	width	Yes	<pre>{"border":{"width": 120}}</pre>	Indicates width, color and style settings of the border for the form field.
	color	String	<pre>{"border":{"color": "#ffff00"} {"border": {"color": "rgb(255, 178, 0)"}} {"border": {"color": "rgba(188, 100, 0, 255)"}}</pre>	
	style	Enum string: <ul style="list-style-type: none"><li>none</li><li>solid</li><li>dashed</li><li>beveled</li><li>inset</li><li>underline</li><li>unknown</li></ul>	<pre>{"border": {"style": "dashed"}}</pre>	
font	size	Yes	<pre>{"font": {"size": 18}}</pre>	Indicates various font settings which can be used in the form field.
	color	String	<pre>{"font": {"color": "#ffff00"}} {"font": {"color": "rgb(255, 178, 0)"}} {"font": {"color": "rgba(188, 100, 0, 255)"}}</pre>	
	name	String	<pre>{"font": {"name": "sans-serif"}}</pre>	
	bold	Boolean	<pre>{"font": {"bold&gt;": true}}</pre>	
	italic	Boolean	<pre>{"font": {"italic": true}}</pre>	
locked		Boolean	<pre>{"locked": true}</pre> Indicates whether the user can change the properties of field or not.	
name		String	<pre>{"name": "The field name"}</pre> Indicates the unique name of field.	
readOnly		Boolean	<pre>{"readOnly": true}</pre> Indicates whether the user can change the value of field or not	
required		Boolean	<pre>{"required": true}</pre> Indicates whether the field must have a value.	
printed		Boolean	<pre>{"printed":false}</pre> Indicates whether to print the field when page is printed.	


<b>hidden</b>	Boolean	{"hidden":true}	Indicates whether to display the field or not.
<b>mouseUp</b>	JsonObject	{ "mouseUp":{"script":"fBox1 = this.getField(\"checkbox\") ;\r\nfBox1.display = display.hidden", "submit":"http://localhost:80//myscript#FDF", "reset":{"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the mouse button is released in the active area of the field.
<b>mouseDown</b>	JsonObject	{ "mouseDown":{"script":"fBox1 = this.getField(\"checkbox\") ;\r\nfBox1.display = display.hidden", "submit":"http://localhost:80//myscript#FDF", "reset":{"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the mouse button is pressed in the active area of the field.
<b>mouseEnter</b>	JsonObject	{ "mouseEnter":{"script":"fBox1 = this.getField(\"checkbox\") ;\r\nfBox1.display = display.hidden", "submit":"http://localhost:80//myscript#FDF", "reset":{"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the mouse button enters the field's active area.
<b>mouseExit</b>	JsonObject	{ "mouseExit":{"script":"fBox1 = this.getField(\"checkbox\") ;\r\nfBox1.display = display.hidden", "submit":"http://localhost:80//myscript#FDF", "reset":{"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the mouse button exits the field's active area.
<b>onFocus</b>	JsonObject	{ "onFocus":{"script":"fBox1 = this.getField(\"checkbox\") ;\r\nfBox1.display = display.hidden", "submit":"http://localhost:80//myscript#FDF", "reset":{"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the annotation receives the input focus.
<b>onBlur</b>	JsonObject	{ "onBlur":{"script":"fBox1 = this.getField(\"checkbox\") ;\r\nfBox1.display = display.hidden", "submit":"http://localhost:80//myscript#FDF", "reset":{"fieldNames": ["checkbox", "textbox"]}}}	Indicates the actions to be performed in sequence when the annotation loses the input focus.

<b>format</b>	String	<code>{"format": "event.value = (event.value * 100) + \" % \";"}</code>	Indicates a JavaScript action to be performed before the field is formatted to display its current value. This action can modify the field's value before formatting.
<b>validate</b>	String	<code>{"validate": "if (event.value &lt; 0    event.value &gt; 100) {\r\n" + "app.beep(0);\r\n" + "app.alert(\"Invalid value for field \" + event.target.name);\r\n" + "event.rc = false;\r\n" + "}"}</code>	Indicates a JavaScript action to be performed when the field's value is changed. This action can check the new value for validity.
<b>calculate</b>	String	<code>{"calculate": "var oil = this.getField(\"Oil\");\r\n" + "var filter = this.getField(\"Filter\");\r\n" + "event.value (oil.value + filter.value) * 1.0825;"}</code>	Indicates a JavaScript action to be performed to recalculate the value of this field when that of another field changes.
<b>keystroke</b>	String	<code>{"keystroke": "if (!event.willCommit) {\r\n" + "var f = this.getField(\"myPictures\");\r\n" + "var i = this.getIcon(event.change);\r\n" + "f.buttonSetIcon(i);\r\n" + "}"}</code>	Indicates a JavaScript action to be performed when the user types a keystroke into a text field or combo box or modifies the selection in a scrollable list box. This action can check the keystroke for validity and reject or modify it.

2. The below table describes the settings which can be applied to JsonObject to indicate an action:

Name		Value Type	Example	Description
<b>script</b>		String	<code>{"script": "fBox1 = this.getField(\"checkbox\");\r\nfBox1.display = display.hidden"}</code>	Indicates an action which causes a script to be compiled and executed by the JavaScript interpreter.
<b>submit</b>		String	<code>{"submit": "http://localhost:80//myscript#FDF"}</code>	Indicates an action to transmit the names and values of selected interactive form fields to a specified uniform resource locator (URL), presumably the address of a Web server that will process them and send back a response.
<b>reset</b>	<b>fieldNames</b>	Yes	<code>{"fieldNames": ["checkbox", "textbox"]}</code>	Indicates the list of names of fields that should be processed (or

			excluded from processing)by this action. If empty then all fields will be processed.
	<b>exclude</b>	Boolean	<code>{"exclude":true}</code> Indicates whether to exclude the fields specified in fieldNames from processing (by default, this property is false and the specified fields are included).

 **Note:** Snippets of JavaScript code needs to be escaped.

3. The below table describes the settings which can be applied to Checkbox form field:

Name	Value Type	Example	Description
<b>checkStyle</b>	Enum string: <ul style="list-style-type: none"> <li>• check</li> <li>• circle</li> <li>• cross</li> <li>• diamond</li> <li>• square</li> <li>• Star</li> </ul>	<code>{"checkStyle": "circle"}</code>	Indicates the style of check mark.
<b>value</b>	Boolean	<code>{"value": true}</code>	Indicates the value of Checkbox. (If the value is missing, GcExcel automatically tries to convert the cell's value to Boolean, and then, set it to the property after processing the template.)
<b>defaultValue</b>	Boolean	<code>{"defaultValue": false}</code>	Indicates the default value of Checkbox.

4. The below table describes the settings which can be applied to Textbox form field:

Name	Value Type	Example	Description
<b>value</b>	String	<code>{"value": "Hunter"}</code>	Indicates the value of Textbox.
<b>defaultValue</b>	String	<code>{"defaultValue": "Input your name!"}</code>	Indicates the default value of Textbox.
<b>combo</b>	Boolean	<code>{"combo":true}</code>	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
<b>password</b>	Boolean	<code>{"password":true}</code>	Indicates whether the field is intended for entering a secure password that should not be visible on the screen.
<b>spellcheck</b>	Boolean	<code>{"spellcheck":false}</code>	Indicates whether the text entered in the field is spell-checked.

<b>scrollable</b>	Boolean	{"scrollable":false}	Indicates whether the field is scrollable to accommodate more text than it fits within its annotation rectangle.
<b>maxLen</b>	Integer	{"maxLen":10}	Indicates the maximum length of the field's text, in characters.
<b>multiline</b>	Boolean	{"multiline":true}	Indicates whether the field can contain multiple lines of text.
<b>justification</b>	Enum string: <ul style="list-style-type: none"> <li>• left</li> <li>• center</li> <li>• right</li> </ul>	{"justification": "center"}	Indicates the justification to be used while displaying the field's text.

5. The below table describes the settings which can be applied to Listbox form field:

Name	Value Type	Example	Description
<b>value</b>	String Array	{"value": ["US", "UK"]}	Indicates the value of Listbox.
<b>defaultValue</b>	String Array	{"defaultValue": ["US", "UK"]}	Indicates the default value of Listbox.
<b>commitOnSelChange</b>	Boolean	{"commitOnSelChange": true}	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
<b>selectedIndex</b>	Integer	{"selectedIndex": 0}	Indicates the indexes of selected item.
<b>sort</b>	Boolean	{"sort": true}	Indicates whether the field's option items should be sorted alphabetically.
<b>spellCheck</b>	Boolean	{"spellCheck": true}	Indicates whether the text entered in the field is spell-checked.
<b>selectedIndexes</b>	Integer Array	{"selectedIndexes": [0, 2, 5]}	Indicates the indexes of selected items.
<b>multiSelect</b>	Boolean	{"multiSelect": true}	Indicates whether more than one of the field's option items may be selected simultaneously.
<b>exportValue</b>	String	{"exportValue": "TheResult"}	Indicates the export value of Listbox field.

6. The below table describes the settings which can be applied to Combobox form field:

Name	Value Type	Example	Description
<b>value</b>	String Array	{"value": ["US", "UK"]}	Indicates the value of Combobox.
<b>defaultValue</b>	String Array	{"defaultValue": ["US", "UK"]}	Indicates the default value of Combobox.

<b>commitOnSelChange</b>	Boolean	{"commitOnSelChange": true}	Indicates whether the new value is committed as soon as a selection is made with the pointing device.
<b>selectedIndex</b>	Integer	{"selectedIndex": 0}	Indicates the indexes of selected item.
<b>sort</b>	Boolean	{"sort": true}	Indicates whether the field's option items should be sorted alphabetically.
<b>spellCheck</b>	Boolean	{"spellCheck": true}	Indicates whether the text entered in the field is spell-checked.
<b>editable</b>	Boolean	{"editable": true}	Indicates whether the Combobox includes an editable text box as well as a drop-down list.

7. The below table describes the settings which can be applied to Radiobutton form field:

Name	Value Type	Example	Description
<b>checkStyle</b>	Enum string: <ul style="list-style-type: none"> <li>• check</li> <li>• circle</li> <li>• cross</li> <li>• diamond</li> <li>• square</li> <li>• Star</li> </ul>	{"checkStyle": "circle"}	Indicates the style of check mark.
<b>groupName</b>	String	{"groupName": "Teams"}	Indicates the name of radio button group.  Radio buttons with the same group name are added in the same group.  (If the value is missing, GcExcel automatically adds radio buttons expanded from the same template cell to the same group after processing the template)
<b>radiosInUnison</b>	Boolean	{"radiosInUnison": true}	Indicates whether a group of radio buttons within a radio button field that use the same value for the on state will turn on and off in unison. If one is checked, they are all checked. If clear, the buttons are mutually exclusive (the same behavior as HTML radio buttons).
<b>checkedChoice</b>	String	{"checkedChoice": "Team5"}	Indicates the value of checked option.
<b>defaultCheckedChoice</b>	String	{"defaultCheckedChoice": "Team1"}	Indicates the value of checked option when the user first opens the



			form.
--	--	--	-------


8. The below table describes the settings which can be applied to Pushbutton form field:

Name		Value Type	Example	Description
<b>highlighting</b>		<b>Enum string:</b> <ul style="list-style-type: none"> <li>• none</li> <li>• invert</li> <li>• outline</li> <li>• push</li> </ul>	{"highlighting": "outline"}	Indicates the annotation's highlighting mode.
<b>caption</b>		String	{"caption": "Push"}	Indicates the button's caption.
<b>image</b>		Base64 String	{"image": "The base64 image data."}	Indicates the button's image.
<b>captionImageRelation</b>		<b>Enum string:</b> <ul style="list-style-type: none"> <li>• captionOnly</li> <li>• imageOnly</li> <li>• captionBelowIcon</li> <li>• captionAboveIcon</li> <li>• captionAtRight</li> <li>• captionAtLeft</li> <li>• captionOverlaid</li> </ul>	{"captionImageRelation": "captionBelowIcon"}	Indicates the positioning of button's caption relative to image.
<b>downCaption</b>		String	{"downCaption": "Push Down"}	Indicates the button's caption when user presses the button.
<b>downImage</b>		Base64 String	{"downImage": "The base64 image data."}	Indicates the button's image when user presses the button.
<b>rolloverCaption</b>		String	{"rolloverCaption": "Rollover"}	Indicates the button's caption when the user rolls the cursor into its active area without pressing the mouse button.
<b>rolloverImage</b>		Base64 String	{"rolloverImage": "The base64 image data."}	Indicates the button's image when the user rolls the cursor into its active area without pressing the mouse button.
<b>imageScale</b>	mode	Yes	{"imageScale": {"mode": "bigger"}}	Indicates the scaling mode.
	proportional	Boolean	{"imageScale": {"proportional": true}}	Indicates whether an image should be scaled proportionally.
	x	Float	{"imageScale": {"proportional": true, "x":	Indicates the position of an image.

			0.6}}	The two numbers between 0.0 and 1.0 indicates the fraction of leftover space to allocate at the left and bottom of an image. A value of (0.0, 0.0) positions the image at the bottom-left corner of the button rectangle. A value of (0.5, 0.5) centers it within the rectangle.  This value is used only if the image is scaled proportionally.
	y	Float	{"imageScale": {"proportional": true, "y": 0.8}}	
	ignoreBorder	Boolean	{"imageScale": {"ignoreBorder": true}}	Indicates whether a button's appearance should be scaled to fit fully within the bounds of the annotation without taking into consideration the line width of the border.

9. The below table describes the settings which can be applied to Signature form field:

Name	Value Type	Example	Description
<b>lockType</b>	<b>Enum string:</b> <ul style="list-style-type: none"> <li>all</li> <li>specifiedOnly</li> <li>allButSpecified</li> </ul>	{"lockType": "specifiedOnly"}	Indicates the type of locked fields.
<b>fieldNames</b>	String Array	{"fieldNames": ["signerName", "time"]}	Indicates the list of field names which should be included or excluded from processing depending on lockType property.
<b>LockedFields</b>	Boolean	{"LockedFields": true}	Indicates whether to lock the fields when SignatureFormField is signed or not.

 **Note:** GcExcel Template generates only digital signature fields in PDF documents. If you want to add signatures on signature fields, you need to use GcPDF or PDFBox to process.

## Charts

Excel charts can be added in Template layout which are visible in the Excel report. It is very useful as charts are often used in Excel reports to display graphical data. Along with that, it provides an advantage that the final chart will always be updated with the latest data, when the template is processed.

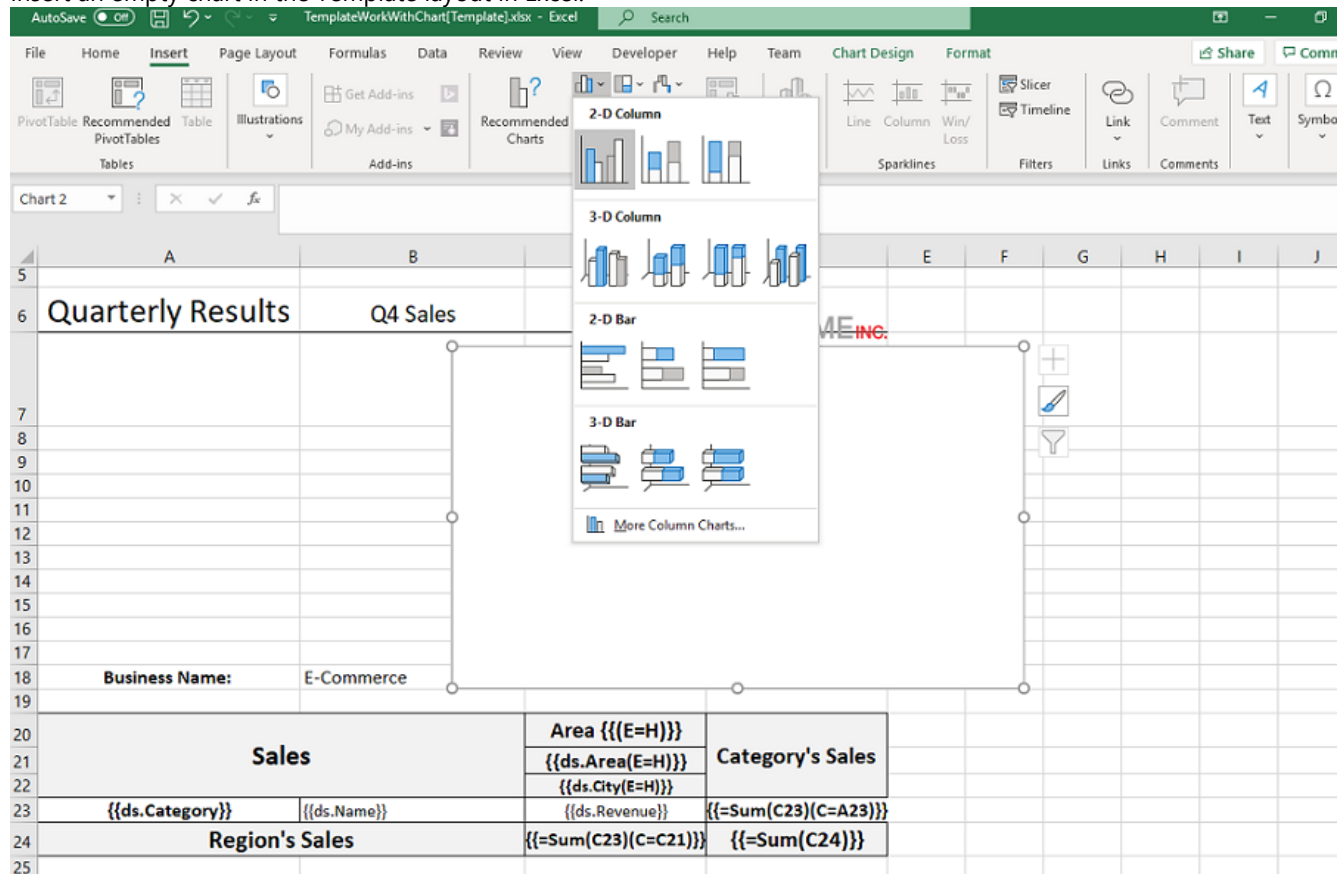
The Excel charts are bound with template cells by specifying the series name, series value and axis labels in the template layout. While processing the template layout, the chart is bound to the data, and the Excel report is generated with the chart displaying

final data. A chart can be placed in a worksheet with its data or in another worksheet too.

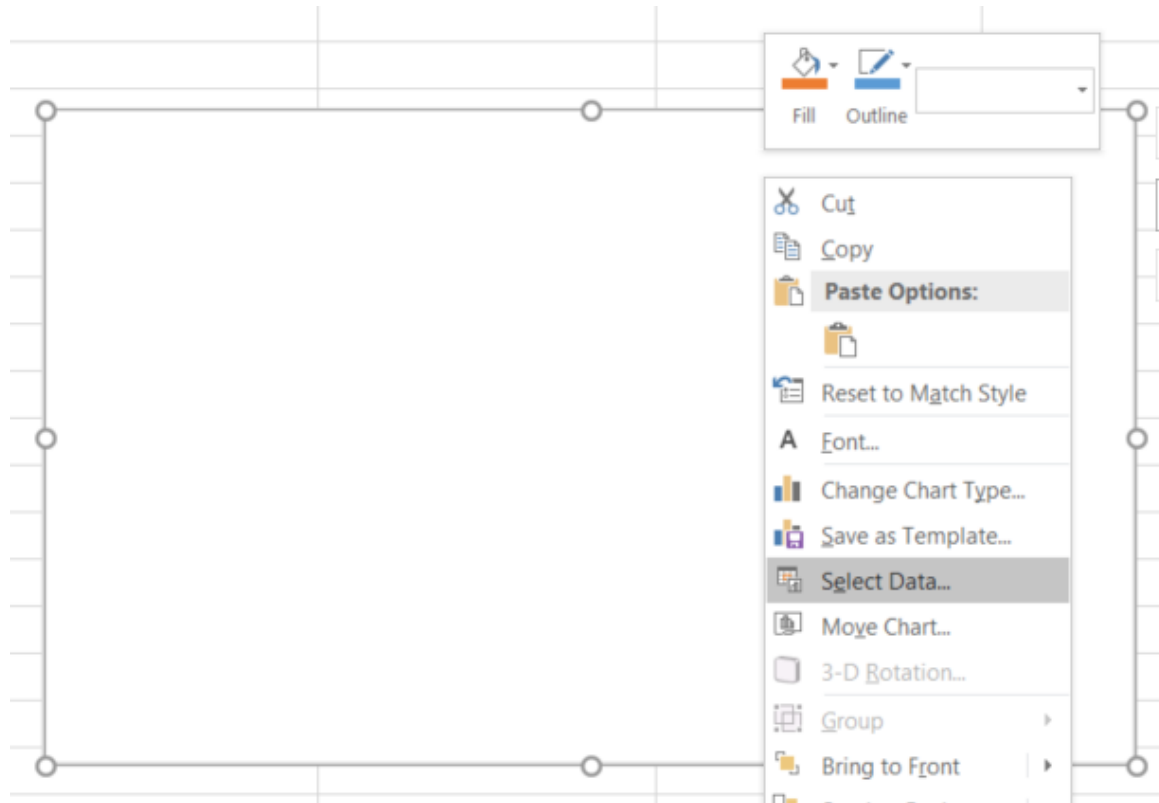
Follow the steps mentioned below to add chart in a template layout and configure its data to template cells:

Here we are adding a chart to a 'Sales Data' report which displays the sales of fruits and vegetables in different areas of North and South China by various salespersons. The chart in the template configures 'Salespersons' as series name, 'Sales' as series value and 'Products' as axis labels to display the sales of products in a graphical manner.

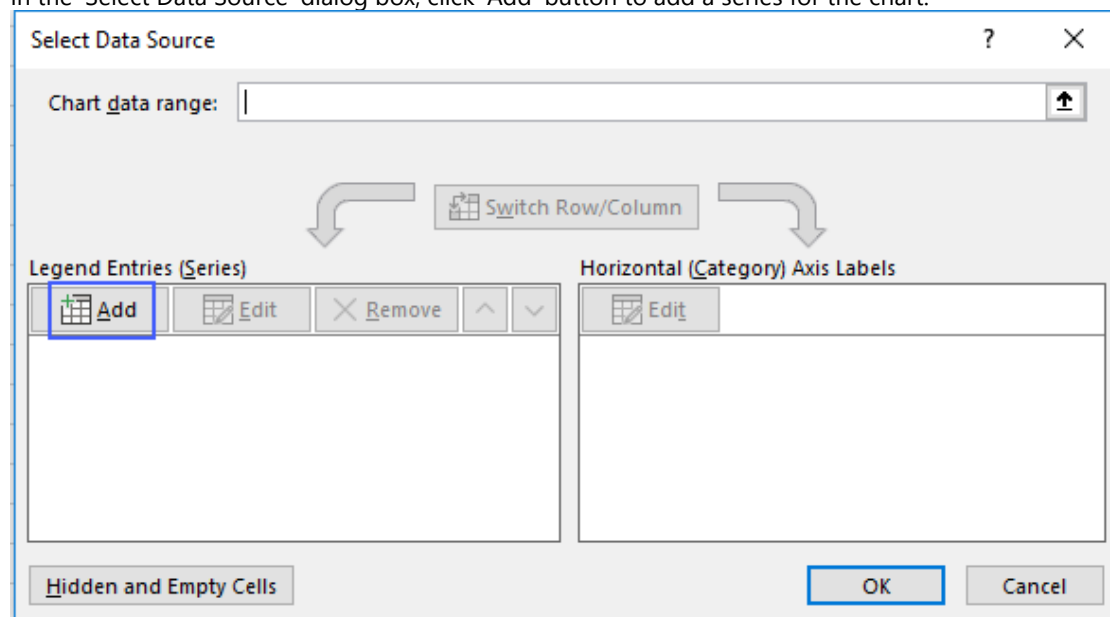
1. Insert an empty chart in the Template layout in Excel.



2. Right click on the chart and choose 'Select Data' from context menu



3. In the 'Select Data Source' dialog box, click 'Add' button to add a series for the chart.



4. In the 'Edit Series' dialog box, click in 'Series Name' and then select 'ds.Salesman' field of the template layout as salesman field is being used as series for the chart.

Sales		Area {{{E=H}}}	Category's Sales
		{{ds.Area(E=H)}}	
		{{ds.City(E=H)}}	
{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C23)(C=A23)}}
Region's Sales		{{=Sum(C23)(C=C21)}}	{{=Sum(C24)}}

Edit Series

?

×

Series name:

=Sales!\$B\$23

↑

Select Range

Series values:

=1

↑

= 1

OK

Cancel

5. Next, click in 'Series Values' and then select 'ds.Sales' field of the template layout as sales field is being used as the value for the series of the chart.

Sales		Area {{{E=H}}}	Category's Sales
		{{ds.Area(E=H)}}	
		{{ds.City(E=H)}}	
{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	{{=Sum(C23)(C=A23)}}
Region's Sales		{{=Sum(C23)(C=C21)}}	{{=Sum(C24)}}

Edit Series

?

×

Series name:

=Sales!\$B\$23

↑

= {{ds.Name}}

Series values:

=Sales!\$C\$23

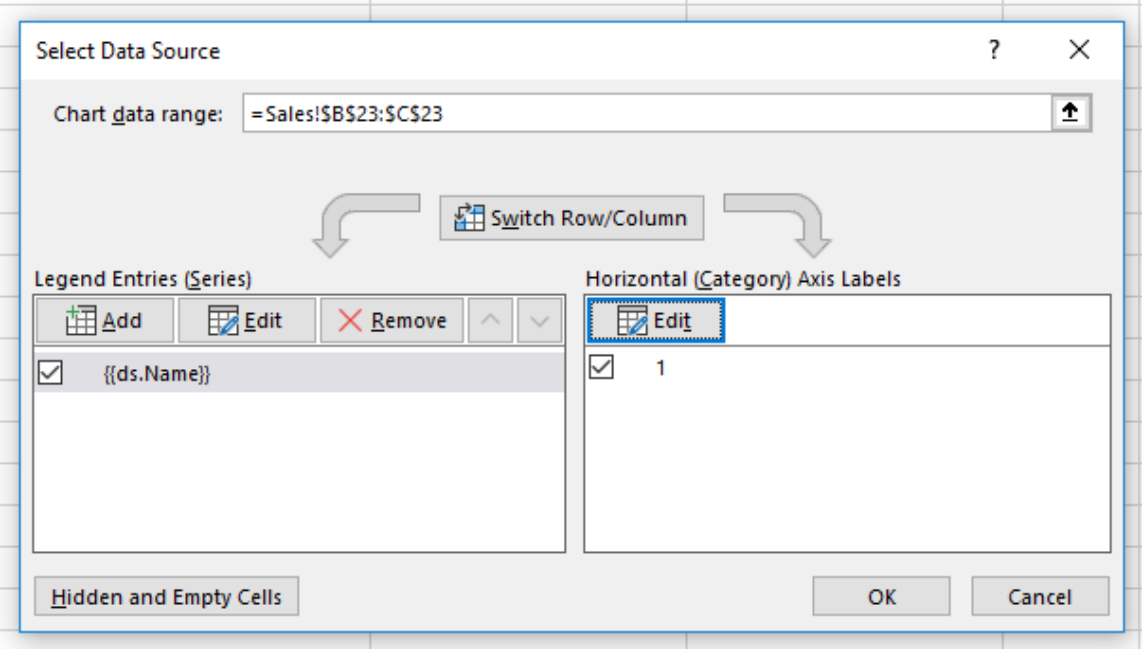
↑

= 1

OK

Cancel

6. Click on the 'Edit' button highlighted in the below screenshot.



7. In the 'Axis Labels' dialog box, click in 'Axis Label Range' and then select 'ds.Product' field of the template layout as products field is being used as axis label of the chart.

Sales		Area {{{E=H}}}	Category's Sales
		{{ds.Area(E=H)}}	
		{{ds.City(E=H)}}	
{{ds.Category}}	{{ds.Name}}	{{ds.Revenue}}	
Region's Sales		{{=Sum(C23)(C=C21)}}	{{=Sum(C24)}}

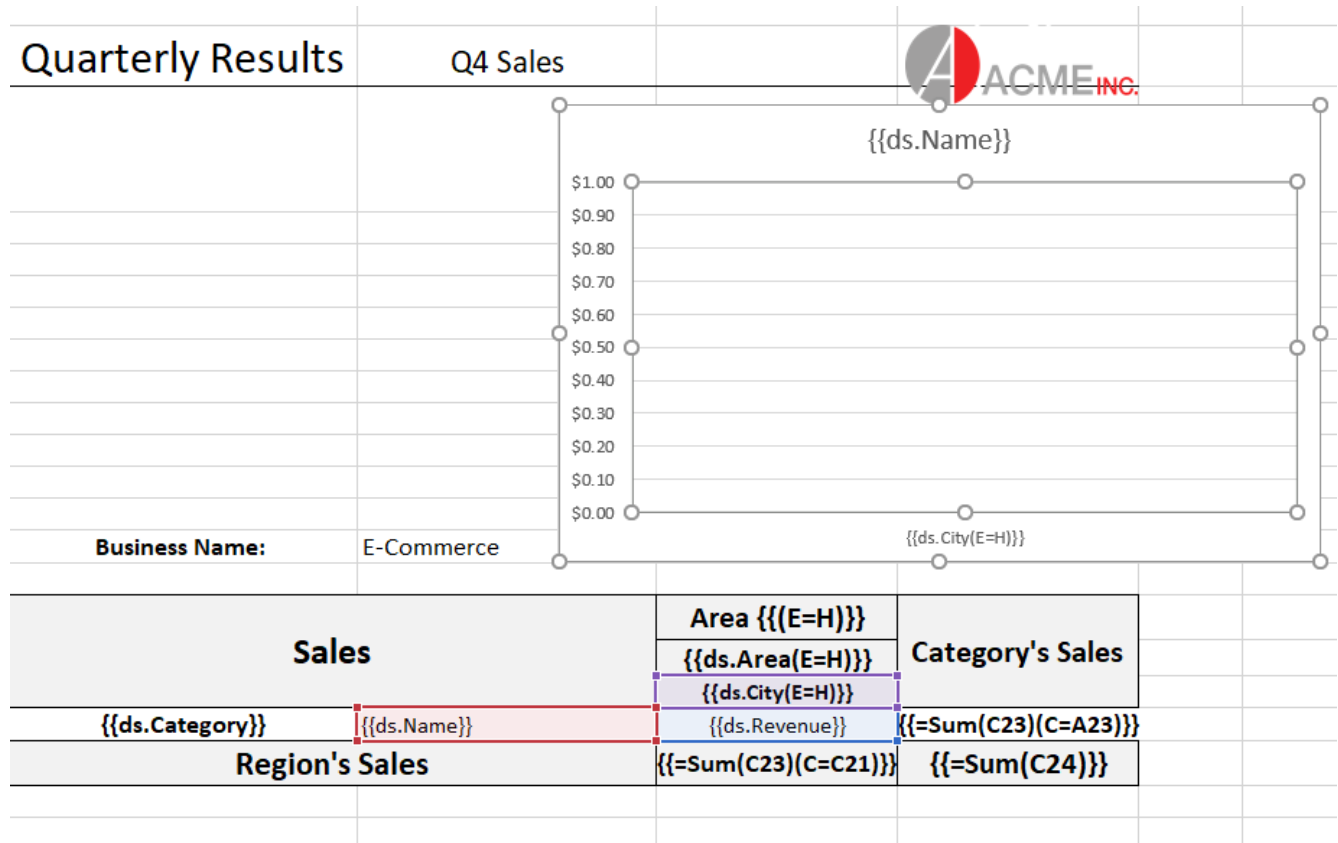
Axis Labels

Axis label range:

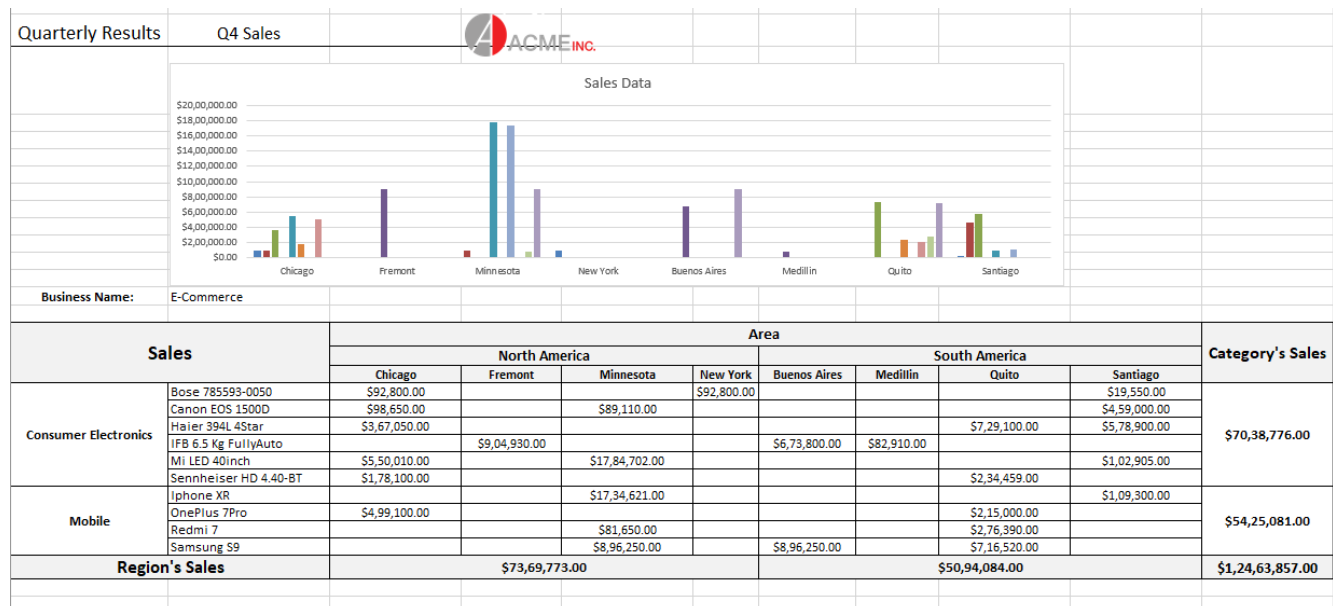
=Sales!\$C\$22

OK Cancel

8. Click Ok to submit the data configuration.



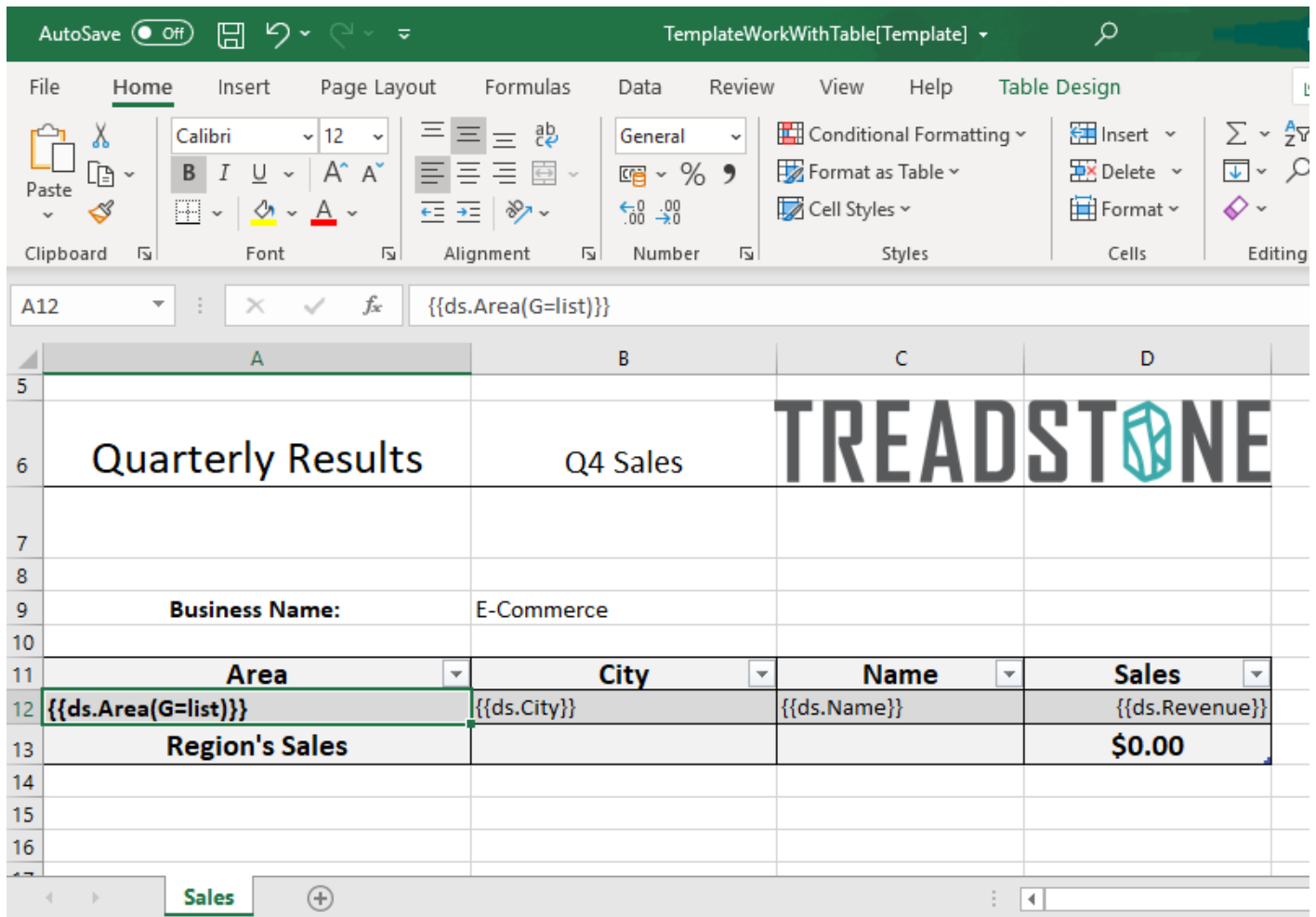
After processing the template layout, the Excel report will look like below:



## Tables

Tables are essential to depict large amounts of data in an organized way. GcExcel supports using Excel tables in template layouts where various operations can also be performed on it like filtering, sorting etc.

This template example lists E-Commerce sales for different areas grouped as a list. The template cells are defined within the table layout. You can also download the **Excel template layout ('TemplateWorkWithTable[Template].xlsx' in the on-line documentation)** from here.




After GcExcel processes the template layout, the Excel report will look like below:



A	B	C	D
Quarterly Results	Q4 Sales	TREADSTONE	
Business Name:	E-Commerce		
Area	City	Name	Sales
North America	Chicago	Bose 785593-0050	\$92,800.00
North America	New York	Bose 785593-0050	\$92,800.00
South America	Santiago	Bose 785593-0050	\$19,550.00
North America	Chicago	Canon EOS 1500D	\$98,650.00
North America	Minnesota	Canon EOS 1500D	\$89,110.00
South America	Santiago	Canon EOS 1500D	\$4,59,000.00
North America	Chicago	Haier 394L 4Star	\$3,67,050.00
South America	Quito	Haier 394L 4Star	\$7,29,100.00
South America	Santiago	Haier 394L 4Star	\$5,78,900.00
North America	Fremont	IFB 6.5 Kg FullyAuto	\$9,04,930.00
South America	Buenos Aires	IFB 6.5 Kg FullyAuto	\$6,73,800.00
Region's Sales			\$41,05,690.00

An Excel table can be incorporated in a template layout in two ways:

1. **Template cells inside an Excel table:** You can insert a table in Excel's template layout and define template cells inside it, as shown in above screenshot. The table is resized according to the expanded data after processing the template in GcExcel.
2. **Excel table inside a template cell's range:** You can define a template cell with [Range property](#) and insert a table anywhere within that range. The table is copied according to the expansion data after processing the template in GcExcel.

 **Note:** Table formulas are also supported in template cells.

## Limitations

- In GcExcel Templates, the default group type is "Merge", which is not supported in case of tables. Hence, you should explicitly set the group type to any other value except "Merge".
- **Excel table inside a template cell's range:** The complete range of table should be included in the template cell's Range property. For example, if a table occurs in the range C5:D8, the template cell should have the "Range(R)" property, for example: `{{ds.Area(R=C5:D8)}}`, to include table inside cell range C5:D8. However, for [sheet name template](#), any table in the current sheet is included by default. So, it doesn't need to set "Range" property.
- **Template cells inside an Excel table:** If [sheet name template](#) is also used along with table, there might be layout issues while expanding the template and the table might be moved to an incorrect location. Hence, you should convert table to cell range before processing.

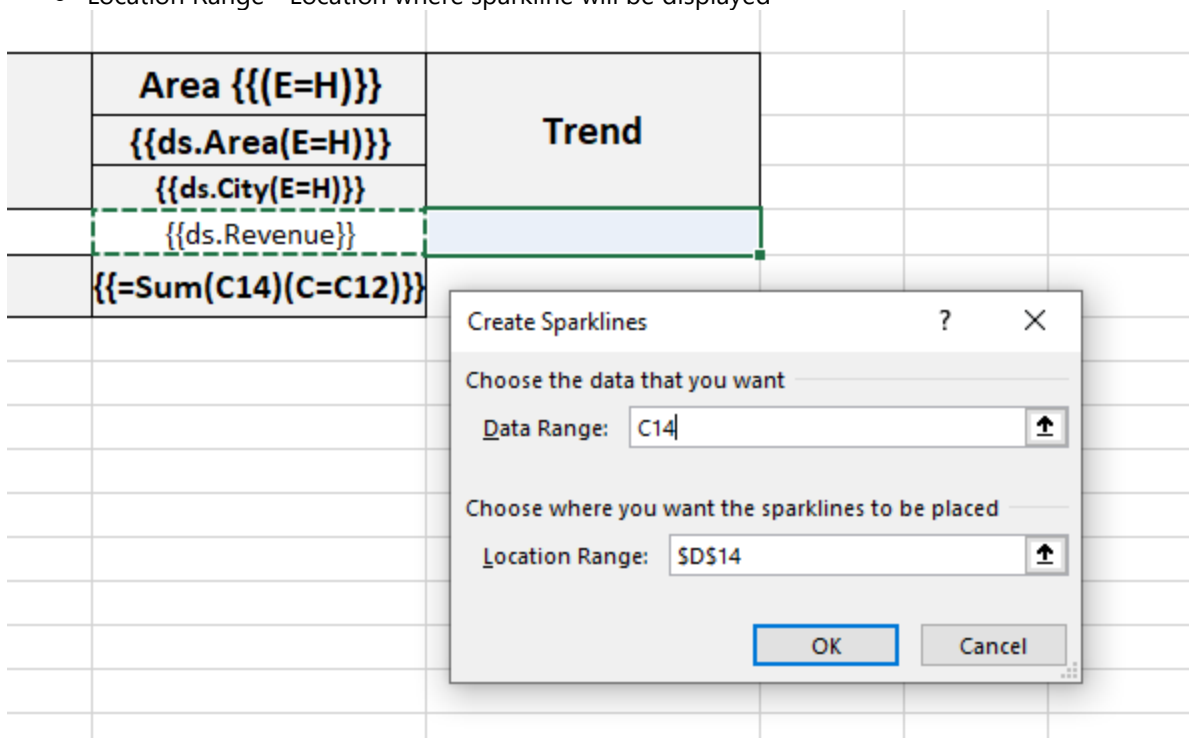
## Sparklines

GcExcel supports adding sparklines in template layout, which are visible in the Excel report generated after processing the template.

Follow the steps mentioned below to add a sparkline in template layout and configure its data to template cells:

You can also download the **Excel template layout ('TemplateWorkWithSparkline[Template].xlsx' in the on-line documentation)** used in below example.

1. Insert a sparkline in Excel's template layout by choosing **Menu | Insert | Sparklines**.
2. In the "Create Sparklines" dialog box, choose a template cell as:
  - Data Range - Data to be displayed by sparkline
  - Location Range - Location where sparkline will be displayed



After GcExcel processes the template layout, the Excel report will look like below:

Business Name:		E-Commerce									
Sales		Area								Trend	
		North America				South America					
		Chicago	Fremont	Minnesota	New York	Buenos Aires	Medillin	Quito	Santiago		
Consumer Electronics	Bose 785593-0050	\$92,800.00			\$92,800.00				\$19,550.00	<div><div></div><div></div><div></div></div>	
	Canon EOS 1500D	\$98,650.00		\$89,110.00					\$4,59,000.00	<div><div></div><div></div><div></div></div>	
	Haier 394L 4Star	\$3,67,050.00						\$7,29,100.00	\$5,78,900.00	<div><div></div><div></div><div></div></div>	
	IFB 6.5 Kg FullyAuto		\$9,04,930.00			\$6,73,800.00	\$82,910.00			<div><div></div><div></div><div></div></div>	
	Mi LED 40inch	\$5,50,010.00		\$17,84,702.00					\$1,02,905.00	<div><div></div><div></div><div></div></div>	
	Sennheiser HD 4.40	\$1,78,100.00						\$2,34,459.00		<div><div></div><div></div><div></div></div>	
Mobile	Iphone XR			\$17,34,621.00					\$1,09,300.00	<div><div></div><div></div><div></div></div>	
	OnePlus 7Pro	\$4,99,100.00						\$2,15,000.00		<div><div></div><div></div><div></div></div>	
	Redmi 7			\$81,650.00				\$2,76,390.00		<div><div></div><div></div><div></div></div>	
	Samsung S9			\$8,96,250.00		\$8,96,250.00		\$7,16,520.00		<div><div></div><div></div><div></div></div>	
Region's Sales		\$73,69,773.00				\$50,94,084.00					

**Note:** In Excel report, the sparkline whose data range and location range are in the same column is displayed as

a 'vertical' sparkline, otherwise, as 'horizontal' sparkline.

## Data Source Binding

Once the template layout is prepared in Excel including bound fields, expressions, formula and sheet name fields, these fields need to be bound to a data source. You can add a data source using the **AddDataSource** ('**addDataSource Method**' in the **on-line documentation**) method and bind the data with template using the **ProcessTemplate** ('**processTemplate Method**' in the **on-line documentation**) method. This will populate the data from datasource in the template fields to generate the Excel report.

Also, you can use multiple data sources or multiple data tables within a data source and populate data through them. The syntax requires you to define the object of the data source followed by the data field. For example, the below template layout merges data from two data sources, the employee information from one data table and Department information from another table.

Employee Name	Employee ID	Department	Department HOD Name
{{emp.name(G = list, S = None)}}	{{emp.id}}	{{emp.department.department}}	{{emp.department.hodname}}

GcExcel supports the below data sources while using templates:

DataSource	Description	Bind DataSource	Template Syntax
ResultSet	A single table which has collection of rows and columns from any type of database	<pre> Java  //Here in the demo, we use a mock class to generate instance of java.sql.ResultSet. //User who use template in product, must get instance of java.sql.ResultSet from the //related database connection. java.sql.ResultSet datasource = new GcMockResultSet(this.getResourceStream("score.csv"));  //Add data source workbook.addDataSource("ds", datasource); </pre>	<p><b>[Alias of data source].[Table name].[Column name]</b></p> <p>For example:</p> <p>{{ds.Table1.ID}}</p> <p>{{ds.Table2.Team}}</p>
Custom Object	A user-defined object from user code or serialized object of JSON String/File/XML, etc. GcExcel Template supports any data source that can be serialized as a custom object.	<pre> Java  NestedDataTemplate_Student student2 = new NestedDataTemplate_Student(); student2.name = "Mark"; student2.address = "101, Halford Avenue, Fremont, CA"; Family family3 = new Family();  family3.father = new Guardian(); family3.father.name = "Jonathan Williams"; family3.father.setOccupation("Product Engineer"); family3.mother = new Guardian(); family3.mother.name = "Joanna Williams"; family3.mother.setOccupation("Surgeon"); </pre>	<p><b>[Alias of data source].[Field name]</b></p> <p>or</p> <p><b>[Alias of data source].[Property name]</b></p> <p>For example:</p> <p>{{ds.Records.Area}}</p> <p>{{ds.Records.Product}}</p>

		<pre> student2.family = new ArrayList&lt;Family&gt;(); student2.family.add(family3);  //Add data source workbook.addDataSource("ds", student2); </pre>	
Variable	A user-defined variable in code.	<div>Java</div> <pre> String className = "Class 3"; int count = 500;  //Add data source workbook.addDataSource("cName", datasource); workbook.addDataSource("count", count); workbook.addDataSource("owner", "Hunter Liu"); </pre>	<b>[Alias of data source]</b> For example: {{cName}} {{count}} {{owner}}
Array or List	A user-defined array or list in code.	<div>Java</div> <pre> int[] numbers = new int[] { 10, 12, 8, 15}; List&lt;String&gt; countries = new List&lt;String&gt;() { "USA", "Japan", "UK", "China" };  List&lt;Person&gt; peoples = new List&lt;Person&gt;();  Person p1 = new Person(); p1.Name = "Helen"; p1.Age = 12; peoples.Add(p1);  Person p2 = new Person(); p2.Name = "Jack"; p2.Age = 23; peoples.Add(p2);  Person p3 = new Person(); p3.Name = "Fancy"; p3.Age = 25; peoples.Add(p3);  workbook.addDataSource("p", peoples); workbook.addDataSource("countries", countries); workbook.addDataSource("numbers", numbers); </pre>	1. Array or List of base type variable(string, int , double, etc.): <b>[Alias of data source]</b>  2. Array or List of custom object: <b>[Alias of data source].[Field name]</b> or <b>[Alias of data source].[Property name]</b> For example: {{p.Name}} {{p.Age}} {{countries}} {{numbers}}

## Create Excel Report using Template

This walkthrough considers the use case to create a Marketing Report of a company which is launching a new series of smartphones. Hence, an Excel report for the planned marketing activities needs to be created. The report details out the planned events for the launch, its budget and expenses. The datasource used for binding the data, in this case, is Custom Object. The template layout is created in different Excel tabs to generate multiple reports.

The below steps describe how to create an Excel report using template:

1. Create template layouts in different Excel worksheets of a workbook. Define the template layout of Marketing report using different types of fields:
  - **Static Fields:** Define the static fields in template layout, that is, the fields whose values will remain constant in the final report. For example, the header fields or template header like Marketing Report, SmartPhone, Event etc.
  - **Bound Fields:** Specify the datasource bound fields in mustache braces {{ }}. For custom object datasource, define the bound fields as {{ds.Records.FieldName}} where ds is the alias of the datasource, specified in code using addDataSource method.
  - **Expression Fields:** Specify the functions in fields whose value will be calculated using formulas.
  - **Sheet Name:** Create multiple template layouts in different Excel tabs, namely, Marketing Report, Smartphone expenses and Launch events. Specify a data bound FieldName for last sheet, {{ds.Records.Country}}, which will generate multiple reports based on the values of 'Country' field in the data source.

### Template Layout: Marketing Report

The below layout uses the Group property (G=Merge), which will group the smartphones against the corresponding records by displaying it once per group. The merge value merges the cells of each group.

	A	B	C	D	E	F	G
1	Marketing Report						
2							
3	SmartPhone	Event	Budget	Expense	City	Country	Saving
4	{{ds.Records.SmartPhone(G=Merge)}}	{{ds.Records.Event}}	{{ds.Records.Budget}}	{{ds.Records.Expense}}	{{ds.Records.City}}	{{ds.Records.Country}}	{{ds.Records.Budget-ds.Records.Expense}}
5							
6			Total Expenses:	{{=SUM(ds.Records.Expense)}}			
7							
8							
9							
10							
11							

Marketing Report | SmartPhone Expenses | Launch Events | {{ds.Records.Country}}

### Template Layout: SmartPhone Expenses

The below layout uses two template properties, Cell expansion (E=H) and Cell context (C=A3)

- The cell expansion property will expand the smartphone field horizontally.
- The cell context property will make sure that the expense field expands horizontally depending upon the smartphone field.

	A	B	C	D	E	F	G	H
1	SmartPhone Expenses							
2								
3	{{ds.Records.SmartPhone (E=H)}}							
4								
5	Expense							
6	\$ {{ds.Records.Expense(C=A3)}}							
7								
8								
9								
10								
11								

Marketing Report | SmartPhone Expenses | Launch Events | {{ds.Records.Country}}

### Template Layout: Launch Events

The below layout uses four template properties, Range (R=A3:B5), Sort (S=None), Cell expansion (E=H) and Page break (PageBreak=True)

- The Range property which acts as the fallback context for the fields in specified range, which means, that the fields which have no default or explicit context will use this current field as their context.
- The Sort property will not sort the events based on its 'none' value
- The event field will expand horizontally based on the cell expansion property
- The Page Break property will add a vertical and a horizontal page break

	A	B	C	D	E	F	G	H	I
1	Launch Events								
2									
3	SmartPhone	{{ds.Records.SmartPhone (R=A3:B5)}}							
4	Event	{{ds.Records.Event (S=None,E=H,PageBreak=True)}}							
5									
6									
7									
8									
9									
10									
11									

Marketing Report | SmartPhone Expenses | **Launch Events** | {{ds.Records.Country}} | +

#### Template Layout: {{ds.Records.Country}}

	A	B	C	D	E	F	G
1	Marketing Report						
2							
3	SmartPhone	Event	Budget	Expense	City		
4							
5	{{ds.Records.SmartPhone}}	{{ds.Records.Event}}	{{ds.Records.Budget}}	{{ds.Records.Expense}}	{{ds.Records.City}}		
6							
7			Total Expenses:	{{=SUM(ds.Records.Expense)}}			
8							
9							
10							
11							

Marketing Report | SmartPhone Expenses | Launch Events | **{{ds.Records.Country}}** | +

#### 2. Load the template in GcExcel.

Java

```
System.out.println("Generating Marketing Report using GcExcel Templates");
// Initialize workbook
Workbook workbook = new Workbook();
// Load BudgetPlan_CustomObject.xlsx Template in workbook
String templateFile = "BudgetPlan_CustomObject.xlsx";
workbook.open(templateFile);
```

#### 3. Configure DataSource and add Records.

Java

```
// We can have mutiple types of DataSource like Custom Object/ DataSet/
// DataTable/ Json/ Variable.
// Here dataSource is a Custom Object
BudgetVals dataSource = new BudgetVals();
{
```

```
        dataSource.Records = new ArrayList<BudgetRecord>();
    }

    BudgetRecord record1 = new BudgetRecord();
    record1.SmartPhone = "Apple iPhone 11";
    record1.Event = "Phone Launch";
    record1.Budget = 1000;
    record1.Expense = 950;
    record1.City = "Seattle";
    record1.Country = "USA";
    dataSource.Records.add(record1);

    BudgetRecord record2 = new BudgetRecord();
    record2.SmartPhone = "Apple iPhone 11";
    record2.Event = "CEO Meet";
    record2.Budget = 2000;
    record2.Expense = 1850;
    record2.City = "New York";
    record2.Country = "USA";
    dataSource.Records.add(record2);

    BudgetRecord record3 = new BudgetRecord();
    record3.SmartPhone = "Samsung Galaxy S10";
    record3.Event = "CEO Meet";
    record3.Budget = 1600;
    record3.Expense = 1550;
    record3.City = "Paris";
    record3.Country = "France";
    dataSource.Records.add(record3);

    BudgetRecord record4 = new BudgetRecord();
    record4.SmartPhone = "Apple iPhone XR";
    record4.Event = "Phone Launch";
    record4.Budget = 1800;
    record4.Expense = 1650;
    record4.City = "Cape Town";
    record4.Country = "South Africa";
    dataSource.Records.add(record4);

    BudgetRecord record5 = new BudgetRecord();
    record5.SmartPhone = "Samsung Galaxy S9";
    record5.Event = "Phone Launch";
    record5.Budget = 1500;
    record5.Expense = 1350;
    record5.City = "Paris";
    record5.Country = "France";
    dataSource.Records.add(record5);

    BudgetRecord record6 = new BudgetRecord();
    record6.SmartPhone = "Apple iPhone XR";
```

```
record6.Event = "CEO Meet";
record6.Budget = 1600;
record6.Expense = 1550;
record6.City = "New Jersey";
record6.Country = "USA";
dataSource.Records.add(record6);

BudgetRecord record7 = new BudgetRecord();
record7.SmartPhone = "Samsung Galaxy S9";
record7.Event = "CEO Meet";
record7.Budget = 1200;
record7.Expense = 1150;
record7.City = "Seattle";
record7.Country = "USA";
dataSource.Records.add(record7);

BudgetRecord record8 = new BudgetRecord();
record8.SmartPhone = "Samsung Galaxy S10";
record8.Event = "Phone Launch";
record8.Budget = 1100;
record8.Expense = 1070;
record8.City = "Durban";
record8.Country = "South Africa";
dataSource.Records.add(record8);
```

4. Add DataSource in GcExcel, using the addDataSource method.

Java

```
// Add DataSource
// Here "ds" is the alias name of dataSource which is used in templates to
// define fields like {{ds.Records.SmartPhone}}
workbook.addDataSource("ds", dataSource);
```

5. Execute the template using ProcessTemplate method.

Java

```
// Invoke to process the template
workbook.processTemplate();
```

6. Save the final report.

Java

```
// Save to an excel file
System.out.println(
    "BudgetPlan_DataTable.xlsx Template is now bound to Custom Object and generated
    MarketingReport_CustomObject.xlsx file");
workbook.save("MarketingReport_CustomObject.xlsx");
```

The output of the Marketing Report is shown as below:

**Excel Report: Marketing Report**



	A	B	C	D	E	F	G	H
1	<b>Marketing Report</b>							
2								
3	SmartPhone	Event	Budget	Expense	City	Country	Saving	
4								
5	Apple iPhone 11	CEO Meet	2000	1850	New York	USA	150	
6		Phone Launch	1000	950	Seattle	USA	50	
7	Apple iPhone XR	CEO Meet	1600	1550	New Jersey	USA	50	
8		Phone Launch	1800	1650	Cape Town	South Africa	150	
9	Samsung Galaxy S10	CEO Meet	1600	1550	Paris	France	50	
10		Phone Launch	1100	1070	Durban	South Africa	30	
11	Samsung Galaxy S9	CEO Meet	1200	1150	Seattle	USA	50	
12		Phone Launch	1500	1350	Paris	France	150	
13								
14			Total Expenses:	11120				
15								
16								
17								

Excel Report: Smartphone Expenses

	A	B	C	D	E	F	G
2							
3	Apple iPhone 11	Apple iPhone XR	Samsung Galaxy S10	Samsung Galaxy S9			
4							
5	Expense						
6	\$ 950	\$ 1500	\$ 1070	\$ 1150			
7	\$ 1800	\$ 1650	\$ 1550	\$ 1300			
8							
9							

Excel Report: Launch Events

	A	B	C	D	E	F	G	H
1	<b>Launch Events</b>							
2								
3	SmartPhone	Apple iPhone 11						
4	Event	Phone Launch	CEO Meet					
5								
6	SmartPhone	Apple iPhone XR						
7	Event	Phone Launch	CEO Meet					
8								
9	SmartPhone	Samsung Galaxy S10						
10	Event	CEO Meet	Phone Launch					
11								
12	SmartPhone	Samsung Galaxy S9						
13	Event	Phone Launch	CEO Meet					
14								
15								

**Excel Report: Countries** (Multiple reports are created)

	A	B	C	D	E	F	G
1	Marketing Report						
2							
3	SmartPhone	Event	Budget	Expense	City		
4							
5		CEO Meet	2000	1800	New York		
6	Apple iPhone 11	Phone Launch	1000	950	Seattle		
7	Apple iPhone XR	CEO Meet	1600	1500	New Jersey		
8	Samsung Galaxy S9	CEO Meet	1200	1150	Seattle		
9							
10			Total Expenses:	5400			
11							

	A	B	C	D	E	F	G	H
1	Marketing Report							
2								
3	SmartPhone							
4								
5	Apple iPhone XR							
6	Samsung Galaxy S10	Phone Launch	1100	1070	Durban			
7								
8			Total Expenses:	2720				
9								
10								

	A	B	C	D	E	F	G	H
1	Marketing Report							
2								
3	SmartPhone							
4								
5	Samsung Galaxy S1							
6	Samsung Galaxy S9	Phone Launch	1500	1300	Paris			
7								
8			Total Expenses:	2850				
9								
10								

## File Operations

GcExcel Java enables you to manage all the file operations without any hassle.

Users can import (open) files with different formats including .xlsx, .csv and .json files. Also, you can export (save) data from a workbook into several formats including .xlsx, .csv, .pdf and .json files into GcExcel Java. Further, you can choose to save the whole component, a specific sheet or data chunks from a particular cell range to different file types or output streams based on your requirements.

Shared below are some common ways to execute file management operations for a range of file types in GcExcel Java:

- [Import and Export .xlsx Document](#)
- [Export to a PDF File](#)
- [Working with PageSetup](#)
- [Import and Export CSV File](#)
- [Import and Export CSV Files with Delimiters](#)
- [Import and Export JSON Stream](#)
- [Import and Export Macros](#)
- [Import and Export OLE Objects](#)
- [Convert to Image](#)

## Import and Export .xlsx Document

This section summarizes how GcExcel Java handles the spreadsheet documents(.xlsx files).

When you create a workbook using GcExcel Java and save it, you automatically export it to an external location or folder. When bringing an Excel file into GcExcel Java (importing a file or opening a file) and when saving GcExcel Java 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.

GcExcel Java also provides support for preserving the Japanese Ruby characters while executing the import and export operations on an Excel file. Also, users can adjust cells containing Japanese Ruby characters with utmost accuracy after performing other spreadsheet tasks like Insert, Delete, Copy, Cut, Merge, Clear, Sort operations etc.

### Working With Import Flags

While opening a workbook, GcExcel Java also provides you with several open options that can be used during the import operation.

The **ImportFlags ('ImportFlags Enumeration' in the on-line documentation)** enumeration allows users to import the workbook with the specified open options (a total of ten options are available: NoFlag, Data and Formulas, Table, mergeArea, Style, ConditionalFormatting, DataValidation, PivotTable and Shapes) as described in the table shared below.

Import Flag Option	Description
NoFlag	Refers to "No option". This option is used when you don't want to put any import flag while opening the Excel file. This means that all the data in the worksheet will be imported as it is.
Data	Refers to "Read the Data". This option is used when you want to import only the data in the worksheet while opening the Excel file.
Formulas	Refers to "Read the Data and Formulas". This option is used when you want to

	import both the data and the formulas in the worksheet while opening the Excel file.
Table	Refers to "Read the Tables". This option is used when you want to import only the tables in the worksheet while opening the Excel file.
MergeArea	Refers to "Read the Merge Cells". This option is used when you want to import only the merged cells or spanned cells in the worksheet while opening the Excel file.
Style	Refers to "Read the Styles". This option is used when you want to import only the styles applied to the cells in the worksheet while opening the Excel file.
ConditionalFormatting	Refers to "Read the Conditional Formatting". This option is used when you want to import only the conditional formatting rule applied to the worksheet while opening the Excel file.
DataValidation	Refers to "Read the Data Validation". This option is used when you want to import only the data validation rule applied to the worksheet while opening the Excel file.
PivotTable	Refers to "Read the Pivot Tables". This option is used when you want to import only the pivot tables in the worksheet while opening the Excel file.
Shapes	Refers to "Read all the Shapes". This option is used when you want to import only the shapes embedded in the worksheet while opening the Excel file.

The **getDoNotRecalculateAfterOpened** ('**getDoNotRecalculateAfterOpened Method**' in the on-line documentation) and the **setDoNotRecalculateAfterOpened** ('**setDoNotRecalculateAfterOpened Method**' in the on-line documentation) methods of the **XlsxOpenOptions** ('**XlsxOpenOptions Class**' in the on-line documentation) class allows users to get or set a boolean value (True or False) which specifies whether or not the formulas will be recalculated when the file is being imported.

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

#### Java

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

// Setting ImportFlags
EnumSet<ImportFlags> impFlags;
impFlags =
EnumSet.of(ImportFlags.Data, ImportFlags.Formulas,
ImportFlags.Table, ImportFlags.Style);

// Setting XlsxOpenOptions
XlsxOpenOptions options = new XlsxOpenOptions();
options.setImportFlags(impFlags);

// Opening excel file with XlsxOpenOptions
workbook.open("test.xlsx", options);
```

```
// Saving the workbook to xlsx
workbook.save("ImportFlagsForOpenOptions.xlsx");
```

## Export to a PDF File

GcExcel Java provides you with the facility to export workbook to a PDF file. You can set pagination for each worksheet in the workbook and export it to required pages in a PDF file. You can also apply styles, customize fonts, add security options, configure document properties and adjust row height or column width while performing the export operation.

Morover, you can export Excel sheets with charts, slicers and sheet background image to PDF document.

GcExcel Java allows you to save all visible spreadsheets in a workbook to a Portable Document File (PDF) using the **save()** ('**save Method**' in the on-line documentation) method of the **IWorkbook** ('**IWorkbook Interface**' in the on-line documentation) interface. Each worksheet in a workbook is saved to a new page in the PDF file. However, if you want to export only the current sheet (active sheet) to PDF format, you can use the **save()** ('**save Method**' in the on-line documentation) method of the **IWorksheet** ('**IWorksheet Interface**' in the on-line documentation) interface.

The handling of images in the case of PDF export is also very efficient. If a picture is used multiple times in a spreadsheet, GcExcel maintains a single copy of the picture which reduces the size of exported PDF file.

In order to export a spreadsheet to a PDF file, refer to the following example code.

### Java

```
// Create a new workbook and add worksheets
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
IWorksheet worksheet1 = workbook.getWorksheets().add();

// Set value and apply styles to the worksheet
worksheet1.getRange("A1").setValue("Sheet1");
worksheet1.getRange("A1").getFont().setName("Wide Latin");
worksheet1.getRange("A1").getFont().setColor(Color.GetRed());
worksheet1.getRange("A1").getInterior().setColor(Color.GetGreen());


// Export Workbook to pdf file, the exported file has two pages.
workbook.save("ConvertWorkbookToPDF.pdf", SaveFileFormat.Pdf);

// Just export a particular worksheet to pdf file
worksheet1.save("ConvertWorksheetToPDF.pdf", SaveFileFormat.Pdf)
```

While executing the export operation, you can configure fonts, set style and specify the page setup options in order to customize the PDF as per your preferences. Refer to the following topics for more details:

- [Configure Fonts and Set Style](#)
- [Export Pivot Table Styles And Format](#)
- [Export Shapes](#)
- [Export Borders](#)
- [Export Conditional Formatting](#)

- [Export Fills](#)
- [Export Picture](#)
- [Export Sparkline](#)
- [Export Table](#)
- [Export Text](#)
- [Export Vertical Text](#)
- [Shrink To Fit With Text Wrap](#)
- [Export Slicers](#)
- [Support Security Options](#)
- [Support Document Properties](#)
- [Adjust Column Width and Row Height](#)
- [Support Sheet Background Image](#)
- [Control Pagination](#)

 **Note:** The Export to PDF feature in GcExcel Java doesn't support charts and slicers while saving a spreadsheet into PDF format. Besides this, exporting picture settings (such as LineFormat, FillFormat, Brightness, Contrast, Watermark Color Type and black and white pictures in emf format) to PDF are also not supported.

## Configure Fonts and Set Style

GcExcel Java enables users to configure custom fonts and set styles while saving worksheets in PDF format.

Before executing the export operation, users need to make sure they specify the font path that should be used while saving the PDF. If the folder path to the font is not specified and the user is working on Windows OS, the path "C:\Windows\Fonts" will be used by default. However, if the folder path to the font is not specified and the user is working on any other operating system, it is necessary that the user sets the font folder path and copies the used font files to the folder "C:\Windows\Fonts".

The **getUsedFonts()** ('**getUsedFonts Method**' in the on-line documentation) method of the **IWorkbook** ('**IWorkbook Interface**' in the on-line documentation) interface can be used to get the collection of all the fonts used in the workbook.

While exporting to a PDF file, GcExcel Java uses the fonts specified in the **Workbook.FontsFolderPath** ('**FontsFolderPath Field**' in the on-line documentation) in order to render the PDF. However, if the used font doesn't exist, it will make use of some fallback fonts. In case, fallback fonts don't exist in the file, GcExcel Java will throw the exception : "There are no available fonts. Please set a valid path to the FontsFolderPath method of the Workbook!"

In order to configure fonts and set style while saving to a PDF, refer to the following example code.

Java

```
// Create a new workbook and add worksheets
Workbook workbook = new Workbook();
IWorksheet sheet1 = workbook.getWorksheets().get(0);
IWorksheet sheet2 = workbook.getWorksheets().add();

// Set style.
sheet1.getRange("A1").setValue("Sheet1");
sheet1.getRange("A1").getFont().setName("Wide Latin");
sheet1.getRange("A1").getFont().setColor(Color.GetRed());
```

```

sheet1.getRange("A1").getInterior().setColor(Color.GetGreen());

// Add Table
ITable table = sheet1.getTables().add(sheet1.getRange("C1:E5"), true);
sheet2.getRange("A1").setValue("Sheet2");


// Specify font path
Workbook.FontsFolderPath = "C:\\Users\\GPCTAdmin\\Documents\\Fonts";

// Get the used fonts list in workbook, the list are:"Wide Latin", "Calibri"
List<FontInfo> fonts = workbook.getUsedFonts();

// Save to a pdf file
workbook.save("configureFontsAndSetStyle.pdf", SaveFileFormat.Pdf);

// Just export sheet1 to pdf file.
sheet1.save("configureFontsAndSetStyle_sheet.pdf", SaveFileFormat.Pdf);

```

 **Note:** The Export to PDF feature in GcExcel Java doesn't support saving the following worksheet styles to PDF format:

- a) Usage of double underline, single accounting underline, double accounting underline, superscript font effect, subscript font effect.
- b) Alignment Preferences like center across selection, fill alignment, justify alignment, distributed alignment, orientation and text reading order etc.

## Export Pivot Table Styles And Format

GcExcel Java allows users to save Excel files containing distinct pivot table styles and formats into a PDF file.

With extensive support for exporting pivot table styles and format, users can customize how the pivot table is displayed in the PDF format. This includes saving Excel files with custom pivot table layout, pivot table fields, orientation, page size etc. into PDF files as per your specific preferences.

The **getStyle()** ('getStyle Method' in the on-line documentation) and the **setStyle()** ('setStyle Method' in the on-line documentation) methods of the **IPivotTable** ('IPivotTable Interface' in the on-line documentation) interface can be used to get or set the pivot table style. While exporting PDFs with pivot table styles in GcExcel Java, refer to the complete listing of methods with their descriptions shared in the table below:

Method	Description
<b>getShowTableStyleColumnHeaders</b> ('getShowTableStyleColumnHeaders Method' in the on-line documentation)  <b>setShowTableStyleColumnHeaders</b> ('setShowTableStyleColumnHeaders Method' in the on-line documentation)	These methods can be used to get or set whether the column headers should be displayed in the Pivot table.
<b>getShowTableStyleRowHeaders</b> ('getShowTableStyleRowHeaders Method' in the on-line documentation)  <b>setShowTableStyleRowHeaders</b>	These methods can be used to get or set whether the row headers should be displayed in the Pivot table.

<b>('setShowTableStyleRowHeaders Method' in the on-line documentation)</b>	
<b>getShowTableStyleColumnStripes</b> <b>('getShowTableStyleColumnStripes Method' in the on-line documentation)</b> <b>setShowTableStyleColumnStripes</b> <b>('setShowTableStyleColumnStripes Method' in the on-line documentation)</b>	These methods can be used to get or set whether the banded columns in which even columns are formatted differently from odd columns.
<b>getShowTableStyleRowStripes</b> <b>('getShowTableStyleRowStripes Method' in the on-line documentation)</b> <b>setShowTableStyleRowStripes</b> <b>('setShowTableStyleRowStripes Method' in the on-line documentation)</b>	These methods can be used to get or set whether the banded rows in which even row are formatted differently from odd rows.
<b>getShowTableStyleLastColumn</b> <b>('getShowTableStyleLastColumn Method' in the on-line documentation)</b> <b>setShowTableStyleLastColumn</b> <b>('setShowTableStyleLastColumn Method' in the on-line documentation)</b>	These methods can be used to get or set whether to display the grand total columns style.
<b>getShowAsAvailablePivotStyle</b> <b>('getShowAsAvailablePivotStyle Method' in the on-line documentation)</b> <b>setShowAsAvailablePivotStyle</b> <b>('setShowAsAvailablePivotStyle Method' in the on-line documentation)</b>	These methods can be used to get or set whether the specified style is shown as available in the pivot styles gallery.
<b>getNumberFormat</b> <b>('getNumberFormat Method' in the on-line documentation)</b> <b>setNumberFormat</b> <b>('setNumberFormat Method' in the on-line documentation)</b>	These methods can be used to get or set the current field's number format string.

### Using Code

Refer to the following example code in order to export Excel files with pivot table styles and format.

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Create PivotTable
Object sourceData = new Object[][] {
{ "Order ID", "Product", "Category", "Amount", "Date", "Country" },
{ 1, "Carrots", "Vegetables", 4270, new GregorianCalendar(2012, 1, 6), "United States" },
```



```
{ 2, "Broccoli", "Vegetables", 8239, new GregorianCalendar(2012, 1, 7), "United Kingdom" },
{ 3, "Banana", "Fruit", 617, new GregorianCalendar(2012, 1, 8), "United States" },
{ 4, "Banana", "Fruit", 8384, new GregorianCalendar(2012, 1, 10), "Canada" },
{ 5, "Beans", "Vegetables", 2626, new GregorianCalendar(2012, 1, 10), "Germany" },
{ 6, "Orange", "Fruit", 3610, new GregorianCalendar(2012, 1, 11), "United States" },
{ 7, "Broccoli", "Vegetables", 9062, new GregorianCalendar(2012, 1, 11), "Australia" },
{ 8, "Banana", "Fruit", 6906, new GregorianCalendar(2012, 1, 16), "New Zealand" },
{ 9, "Apple", "Fruit", 2417, new GregorianCalendar(2012, 1, 16), "France" },
{ 10, "Apple", "Fruit", 7431, new GregorianCalendar(2012, 1, 16), "Canada" },
{ 11, "Banana", "Fruit", 8250, new GregorianCalendar(2012, 1, 16), "Germany" },
{ 12, "Broccoli", "Vegetables", 7012, new GregorianCalendar(2012, 1, 18), "United States" },
{ 13, "Carrots", "Vegetables", 1903, new GregorianCalendar(2012, 1, 20), "Germany" },
{ 14, "Broccoli", "Vegetables", 2824, new GregorianCalendar(2012, 1, 22), "Canada" },
{ 15, "Apple", "Fruit", 6946, new GregorianCalendar(2012, 1, 24), "France" }, };

worksheet.getRange("A1:F16").setValue(sourceData);
IPivotCache pivotcache = workbook.getPivotCaches().create(worksheet.getRange("A1:F16"));
IPivotTable pivottable = worksheet.getPivotTables().add(pivotcache,
worksheet.getRange("H5"), "pivottable1");

// Create PivotTable style
ITableStyle style = workbook.getTableStyles().add("pivotStyle");

// Set the table style as a pivot table style
style.setShowAsAvailablePivotStyle(true);
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getBorders()
.setLineStyle(BorderLineStyle.DashDotDot);
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getBorders()
.setColor(com.grapecity.documents.excel.Color.FromArgb(204, 153, 255));
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getInterior()
.setColor(com.grapecity.documents.excel.Color.FromArgb(169, 208, 142));
style.getTableStyleElements().get(TableStyleElementType.WholeTable).getFont().setItalic(true);
style.getTableStyleElements().get(TableStyleElementType.WholeTable)
.getFont().setThemeColor(ThemeColor.Accent2);

// Apply the style to current pivot table
pivottable.setStyle(style);

pivottable.setShowTableStyleColumnHeaders(true);
pivottable.setShowTableStyleRowHeaders(true);
pivottable.setShowTableStyleColumnStripes(true);
pivottable.setShowTableStyleRowStripes(true);
pivottable.setShowTableStyleLastColumn(true);

// Add pivot filed and set number format code

// Add two fields
IPivotField field_product = pivottable.getPivotFields().get(1);
field_product.setOrientation(PivotFieldOrientation.RowField);
IPivotField field_Amount = pivottable.getPivotFields().get(3);
field_Amount.setOrientation(PivotFieldOrientation.DataField);
```

```
// Set number format code
field_Amount.setNumberFormat("#,##0");

// Saving workbook to xlsx
workbook.save("PivotTableStyleAndNumberFormat.pdf", SaveFileFormat.Pdf);
```

## Export Shapes

GcExcel Java provides extensive support for loading, saving, printing and exporting Excel files comprising shapes and other drawing objects embedded in the worksheets.

The **getIsPrintable** ('getIsPrintable Method' in the on-line documentation) and the **setIsPrintable** ('setIsPrintable Method' in the on-line documentation) methods of the **IShape** ('IShape Interface' in the on-line documentation) interface can be used to get or set whether the object will be printed in the PDF document. By default, this value is TRUE and hence the shapes embedded in the Excel files are printed. In case you do not want to export shapes to the PDF files, this value must be set to FALSE.

The Export Shapes to PDF feature allows users to print and export different types of shapes such as callouts, lines, rectangles, basic shapes, block arrows, flowcharts, equation shapes, stars and banners etc. This feature is useful especially when the following scenarios are encountered while working with spreadsheets:

- When users have Excel files with graphs, reports and dashboards containing various shapes that they want to export to a PDF file.
- With the help of this feature, users can export spreadsheets that contain preset shapes, basic shapes, custom shapes and grouped shapes with different operations like rotation, flipping, connector arrows and text etc. into a PDF file.
- When users need to export Excel template files and spreadsheets containing shapes with different types of fills (like Solid fill, Gradient fill etc.) while saving to a PDF file.

### Using Code

Refer to the following example code in order to export shapes to PDF.

```
Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Adding Shapes
IShape ShapeBegin =
worksheet.getShapes().addShape(AutoShapeType.CloudCallout, 1, 1, 100, 100);
IShape EndBegin =
worksheet.getShapes().addShape(AutoShapeType.Wave, 200, 200, 100, 100);

// Adding Connector Shape
```

```
IShape ConnectorShape = worksheet.getShapes()  
.addConnector(ConnectorType.Straight, 1, 1, 101, 101);  
  
// Connect shapes by connector shape  
ConnectorShape.getConnectorFormat().beginConnect(ShapeBegin, 3);  
ConnectorShape.getConnectorFormat().endConnect(EndBegin, 0);  
  
/* Get second shape in current worksheet( here it's a connector shape)  
   and do not print it(default value is true) */  
worksheet.getShapes().get(2).setIsPrintable(false);  
  
// Saving workbook to PDF  
workbook.save("ExportShapesToPDF.pdf", SaveFileFormat.Pdf);
```



**Note:** While exporting Excel files containing shapes into the PDF format, some of the exported shapes including the shapes with gradient fill and pattern fill; shapes with multiple lines and gradient lines; shape effects, text settings like text justify, text distribution and vertical text may not work exactly the same as in Excel.

## Export Borders

GcExcel Java allows users to save worksheets with borders while exporting to a PDF file.

In order to export an excel file with borders to PDF format, refer to the following example code.

### Java

```
// Create a new workbook and access the default worksheet  
Workbook workbook = new Workbook();  
IWorksheet sheet = workbook.getWorksheets().get(0);  
  
// Single cell border  
sheet.getRange("B2").getBorders().setThemeColor(ThemeColor.Accent1);  
sheet.getRange("B2").getBorders().setLineStyle(BorderLineStyle.SlantDashDot);  
sheet.getRange("B2").getBorders().get(BordersIndex.DiagonalUp)  
.setThemeColor(ThemeColor.Accent1);  
sheet.getRange("B2").getBorders().get(BordersIndex.DiagonalUp)  
.setLineStyle(BorderLineStyle.SlantDashDot);  
sheet.getRange("B2").getBorders().get(BordersIndex.DiagonalDown)  
.setThemeColor(ThemeColor.Accent1);  
sheet.getRange("B2").getBorders().get(BordersIndex.DiagonalDown)  
.setLineStyle(BorderLineStyle.SlantDashDot);  
  
// Range border  
sheet.getRange("D2:E3").getBorders().setThemeColor(ThemeColor.Accent1);  
sheet.getRange("D2:E3").getBorders().setLineStyle(BorderLineStyle.DashDot);  
sheet.getRange("D2:E3").getBorders().get(BordersIndex.DiagonalDown)  
.setThemeColor(ThemeColor.Accent1);
```

```
sheet.getRange("D2:E3").getBorders().get(BordersIndex.DiagonalDown)
.setLineStyle(BorderLineStyle.DashDot);

// Merge cell border
sheet.getRange("B6:C7").merge();
sheet.getRange("B6:C7").getBorders().setThemeColor(ThemeColor.Accent1);
sheet.getRange("B6:C7").getBorders().setLineStyle(BorderLineStyle.Double);
sheet.getRange("B6:C7").getBorders().get(BordersIndex.DiagonalUp)
.setThemeColor(ThemeColor.Accent1);
sheet.getRange("B6:C7").getBorders().get(BordersIndex.DiagonalUp)
.setLineStyle(BorderLineStyle.Double);

// Apply border style on table
ITable table = sheet.getTables().add(sheet.getRange("B12:G22"), true);

// Create custom table style
ITableStyle customTableStyle = workbook.getTableStyles()
.get("TableStyleMedium10").duplicate();

// Set outline border for "whole table" style
ITableStyleElement wholeTableStyle = customTableStyle.getTableStyleElements()
.get(TableStyleElementType.WholeTable);
wholeTableStyle.getBorders().get(BordersIndex.EdgeTop)
.setThemeColor(ThemeColor.Accent1);
wholeTableStyle.getBorders().get(BordersIndex.EdgeTop)
.setLineStyle(BorderLineStyle.Thick);
wholeTableStyle.getBorders().get(BordersIndex.EdgeRight)
.setThemeColor(ThemeColor.Accent1);
wholeTableStyle.getBorders().get(BordersIndex.EdgeRight)
.setLineStyle(BorderLineStyle.Thick);
wholeTableStyle.getBorders().get(BordersIndex.EdgeBottom)
.setThemeColor(ThemeColor.Accent1);
wholeTableStyle.getBorders().get(BordersIndex.EdgeBottom)
.setLineStyle(BorderLineStyle.Thick);
wholeTableStyle.getBorders().get(BordersIndex.EdgeLeft)
.setThemeColor(ThemeColor.Accent1);
wholeTableStyle.getBorders().get(BordersIndex.EdgeLeft)
.setLineStyle(BorderLineStyle.Thick);

// Set vertical border for "first row strip" style
ITableStyleElement firstRowStripStyle = customTableStyle.getTableStyleElements()
.get(TableStyleElementType.FirstRowStripe);
firstRowStripStyle.getBorders().get(BordersIndex.InsideVertical)
.setThemeColor(ThemeColor.Accent6);
firstRowStripStyle.getBorders().get(BordersIndex.InsideVertical)
.setLineStyle(BorderLineStyle.Dashed);

// Apply custom style to table
```

```
table.setTableStyle(customTableStyle);

// Save to a pdf file
workbook.save("ExportBorders.pdf", SaveFileFormat.Pdf);
```

## Export Conditional Formatting

GcExcel Java allows users to save worksheets with conditional formatting while exporting to a PDF file.

In order to export an excel file with conditional formatting to PDF format, refer to the following example code.

### Java

```
// Create a new workbook and access the default worksheet
Workbook workbook = new Workbook();
IWorksheet sheet = workbook.getWorksheets().get(0);

// Conditional formatting on merge cell
sheet.getRange("B2:C4").merge();
sheet.getRange("B2:C4").setValue(123);
IFormatCondition cf = (IFormatCondition) sheet.getRange("B2:C4").getFormatConditions()
    .add(FormatConditionType.CellValue, FormatConditionOperator.Greater, 0, 0);
cf.getBorders().setThemeColor(ThemeColor.Accent1);
cf.getBorders().setLineStyle(BorderLineStyle.Thin);

// Set cell values
int[] data = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
sheet.getRange("B10:B19").setValue(data);
sheet.getRange("C10:C19").setValue(data);
sheet.getRange("D10:D19").setValue(data);

// Apply conditional formatting
// Color scale
IColorScale cfl = sheet.getRange("B10:B19").getFormatConditions()
    .addColorScale(ColorScaleType.ThreeColorScale);
cfl.getColorScaleCriteria().get(0).setType(ConditionValueTypes.LowestValue);
cfl.getColorScaleCriteria().get(0).getFormatColor()
    .setColor(Color.FromArgb(248, 105, 107));
cfl.getColorScaleCriteria().get(1).setType(ConditionValueTypes.Percentile);
cfl.getColorScaleCriteria().get(1).setValue(50);
cfl.getColorScaleCriteria().get(1).getFormatColor()
    .setColor(Color.FromArgb(255, 235, 132));
cfl.getColorScaleCriteria().get(2).setType(ConditionValueTypes.HighestValue);
cfl.getColorScaleCriteria().get(2).getFormatColor()
    .setColor(Color.FromArgb(99, 190, 123));
```

```
// Data bar
sheet.getRange("C14").setValue(-5);
sheet.getRange("C17").setValue(-8);
IDataBar cf2 = sheet.getRange("C10:C19").getFormatConditions().addDatabar();
cf2.getMinPoint().setType(ConditionValueTypesAutomaticMin);
cf2.getMaxPoint().setType(ConditionValueTypesAutomaticMax);
cf2.setBarFillType(DataBarFillType.Gradient);
cf2.getBarColor().setColor(Color.FromArgb(0, 138, 239));
cf2.getBarBorder().getColor().setColor(Color.FromArgb(0, 138, 239));
cf2.getNegativeBarFormat().getColor().setColor(Color.FromArgb(255, 0, 0));
cf2.getNegativeBarFormat().setBorderColorType(DataBarNegativeColorType.Color);
cf2.getNegativeBarFormat().getBorderColor().setColor(Color.FromArgb(255, 0, 0));
cf2.getAxisColor().setColor(Color.GetBlack());
cf2.setAxisPosition(DataBarAxisPositionAutomatic);

// Icon set
IIconSetCondition cf3 = sheet.getRange("D10:D19")
    .getFormatConditions().addIconSetCondition();
cf3.setIconSet(workbook.getIconSets().get(IconSetType.Icon3Symbols));

// Save to a pdf file
workbook.save("ExportConditionalFormatting.pdf", SaveFileFormat.Pdf);
```

## Control Pagination

GcExcel Java enables users to manage pagination while exporting to a PDF file.

The pagination settings are used to control how the data lying in the worksheets breaks across the pages in the PDF file. The **PrintManager** ('**PrintManager Class**' in the on-line documentation) class contains the methods that help users in handling custom pagination requirements and preferences.

The custom pagination process works in four basic steps as described below.

- Step 1 - Create an instance of the **PrintManager** ('**PrintManager Class**' in the on-line documentation) class
- Step 2 - Get the default pagination information of the workbook using the **paginate()** ('**paginate Method**' in the on-line documentation) method.
- Step 3 - Adjust pagination using different methods of the PrintManager class.
- Step 4 - Save the PDF file by using either the **savePageInfosToPDF()** ('**savePageInfosToPDF Method**' in the on-line documentation) method or the **saveWorkbooksToPDF()** ('**saveWorkbooksToPDF Method**' in the on-line documentation) method.

While configuring the pagination options for a workbook containing multiple worksheets, users have complete control over the flow of content (both textual and graphic) across the pages in the PDF file. Further, users can customize the PDF file by adjusting the automatic page breaks while adding or deleting the pages, and modifying the print information on each page of the PDF file. Also, users can keep some rows together in a page; save the content from more than one worksheet to a single PDF; display custom cell ranges inside the exported PDF file; configure custom page settings (like page number, page count, page content, row headers, column headers, title columns, tail columns, page margins, page header, page footer, paper width, paper height etc.); save different header information on different pages; save the last

page of the PDF file without any headers, export custom page information and export only some specific pages (or worksheets) in the PDF file.

For more information on handling pagination while working with spreadsheets, refer to the following topics:

- [Render Excel Range Inside PDF](#)
- [Export Multiple Sheets To One Page](#)
- [Keep Rows Together Over Page Breaks](#)
- [Delete Blank Pages From Middle](#)
- [Export Different Headers On Different Pages](#)
- [Export Last Page Without Headers](#)
- [Export Custom Page Information](#)
- [Export Specific Pages to PDF](#)
- [Save Multiple Workbooks to Single PDF](#)

## Render Excel Range Inside PDF

GcExcel Java enables users to render Excel cell ranges inside PDF.

This feature is useful especially when you're dealing with bulk data in the spreadsheets and you want to render only specific Excel range inside an existing PDF file. For instance - let's say you have a worksheet containing large amounts of sales data with fields such as "Number of Products Sold", "Area Sales Manager", "Region" etc. but you want to export only a chunk of useful data (like only "Number of Products Sold" and "Region") at some location in a PDF file and not all the data (you don't want to include the "Area Sales Manager" information). In this scenario, the "Render Excel Range Inside PDF" feature can be used to select some specific ranges in the worksheet and render them to specific location in a PDF file to generate full PDF reports.

In order to render Excel range inside the PDF file, you need to first create an instance of the **PrintManager** ('**PrintManager Class**' in the **on-line documentation**) class and then use the **draw()** ('**draw Method**' in the **on-line documentation**) method to render the Excel range on a PDF page at a location. In case, you want to add some extra information in your PDF file (data which is not present in your Excel file), you can use the **appendPage()** ('**appendPage Method**' in the **on-line documentation**) method of the PrintManager class after configuring all the pagination settings. Finally, call the **updatePageNumberAndPageSettings()** ('**updatePageNumberAndPageSettings Method**' in the **on-line documentation**) method in order to update the indexes of the page number and the page settings for each page. When everything is done, simply save your PDF file using the **savePageInfosToPDF()** ('**savePageInfosToPDF Method**' in the **on-line documentation**) method.

Refer to the following example code to allow users to render Excel ranges inside the PDF file .

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Set value
worksheet.getRange("A4:C4").setValue(new Object[]
{ "Device", "Quantity", "Unit Price" });
```

```
worksheet.getRange("A5:C8").setValue(new Object[][]  
{  
  { "T540p", 12, 9850 }, { "T570", 5, 7460 },  
  { "Y460", 6, 5400 }, { "Y460F", 8, 6240 }  
});  
  
// Set style  
worksheet.getRange("A4:C4").getFont().setBold(true);  
worksheet.getRange("A4:C4").getFont()  
  .setColor(Color.GetWhite());  
worksheet.getRange("A4:C4").getInterior()  
  .setColor(Color.GetLightBlue());  
worksheet.getRange("A5:C8")  
  .getBorders().get(BordersIndex.InsideHorizontal)  
  .setColor(Color.GetOrange());  
worksheet.getRange("A5:C8").getBorders()  
  .get(BordersIndex.InsideHorizontal)  
  .setLineStyle(BorderLineStyle.DashDot);  
  
// Configure Page size  
float width = 600f;  
float height = 500f;  
PDRectangle pageSize = new PDRectangle(width, height);  
  
// Create a PDF document  
PDDocument doc = new PDDocument();  
PDPage page = new PDPage(pageSize);  
doc.addPage(page);  
  
// Create an instance of the PrintManager class  
PrintManager printManager = new PrintManager();  
  
// Draw the Range"A4:C8" to the specified location on the page  
printManager.draw(doc, page, new Point(30, 100),  
worksheet.getRange("A4:C8"));  
  
// Save the modified pages into PDF file  
try  
{  
  doc.save("RenderExcelRangesInsidePDFBasic.pdf");  
}  
catch (IOException e)  
{  
  // TODO Auto-generated catch block  
  e.printStackTrace();  
}
```

Refer to the following example code to allow users to render Excel ranges inside the PDF file along with some custom



textual information at runtime to the specified location on the page.

#### Java

```
private static void RenderExcelRangesInPDF() throws Exception
{
    // Create to a pdf file stream
    FileOutputStream outputStream = null;

    try
    {
        outputStream =
            new FileOutputStream("RenderExcelRangesInsideAPDF.pdf");
    }
    catch (FileNotFoundException e)
    {
        e.printStackTrace();
    }

    // Create a new workbook
    Workbook workbook = new Workbook();
    workbook.open(getResourceStream("xlsx/FinancialReport.xlsx"));

    // Create a PDF document.
    PDDocument doc = null;
    try
    {
        doc = PDDocument.load(getResourceStream
            ("xlsx/Acme-Financial Report 2018.pdf"));
    }
    catch (IOException e1)
    {
        e1.printStackTrace();
    }

    // Create an instance of the PrintManager class.
    PrintManager printManager = new PrintManager();

    // Draw the contents of the sheet3 to the fourth page.
    IRange printArea1 =
        workbook.getWorksheets().get(2).getRange("A3:C24");
    Size size1 =
        printManager.getSize(printArea1);
    printManager.draw(doc, doc.getPage(3), new Rectangle(306, 215,
        size1.getWidth(), size1.getHeight()),
        printArea1);

    // Draw the contents of the sheet1 to the fifth page.
```

```
IRange printArea2 =
workbook.getWorksheets().get(0).getRange("A3:F29");
Size size2 = printManager.getSize(printArea2);
printManager.draw(doc, doc.getPage(4),
new Rectangle(71, 250, size2.getWidth(), size2.getHeight()),
printArea2);

// Draw the contents of the sheet2 to the sixth page.
IRange printArea3 =
workbook.getWorksheets().get(1).getRange("A3:G27");
Size size3 = printManager.getSize(printArea3);
printManager.draw(doc, doc.getPage(5),
new Rectangle(71, 230, 783, size3.getHeight()), printArea3);

// Save the modified pages into pdf file.
try
{
    doc.save(outputStream);
    doc.close();
}
catch (IOException e)
{
    e.printStackTrace();
}

// Close the file stream
try
{
    outputStream.close();
}
catch (IOException e)
{
    e.printStackTrace();
}
}

private static InputStream getResourceStream(String resource) throws Exception
{
    return UpcomingFeatures.class.getClassLoader()
        .getResourceAsStream(resource);
}
```

## Export Multiple Sheets To One Page

GcExcel Java enables users to export multiple worksheets to a single page in the PDF file .

This feature is useful especially when you want to analyse all the crucial data at one place in order to facilitate the sharing, manipulation and printing of data in an efficient way. For instance - let's say you have a workbook with multiple worksheets wherein you want to export the content of some worksheets (containing similar type of data) so that all the related data appears on the same page and is saved into a specific page in the PDF file. In this scenario, you can use this feature to export and print data from more than one worksheet to a single page in the PDF file as per your custom requirements and preferences.

In order to export multiple worksheets into a single page of the PDF file, users need to create an instance of the **PrintManager** ('**PrintManager Class**' in the **on-line documentation**) class, get the default pagination settings of the workbook using the **paginate()** ('**paginate Method**' in the **on-line documentation**) method, use the **draw()** ('**draw Method**' in the **on-line documentation**) method of the PrintManager class and finally save the PDF file.

## Using Code

Refer to the following example code to allow users to export multiple worksheets to a single page in the PDF file.

### Java

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

// Open Excel file
workbook.open("MultipleSheetsOnePage.xlsx");

// Create a PDF document
PDDocument doc = new PDDocument();

// This page will save data for multiple pages
PDPage page = new PDPage();
doc.addPage(page);

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Get the pagination information of the workbook
List<PageInfo> pages = printManager.paginate(workbook);

/* Divide the multiple pages into 1 rows and 2 columns
   and printed them on one page */
printManager.draw(doc, page, pages, 1, 2);

// Save the document to pdf file
try
{
    doc.save(outputStream); doc.close();
}
catch (IOException e)
{
    e.printStackTrace();
}
```

```
// Close the file stream
try
{
    outputStream.close();
}
catch (IOException e)
{
    e.printStackTrace();
}
```

## Keep Rows Together Over Page Breaks

GcExcel Java enables users to keep some rows together over page breaks while exporting to a PDF file.

This feature is useful especially when you have data lying in large number of rows in the worksheet that you want to export to a PDF file. For instance - let's say you have a spreadsheet having multiple groups of rows that are often hidden, but ultimately modify the number of pages and page breaks while printing. Now, you want to export your Excel file to PDF in such a way that it keeps some groups of rows together so that they don't split across page breaks or pages when the print operation is executed. In such a scenario, it is extremely helpful to utilize this feature to achieve flawless printing experience and accurate content publishing while exporting to the PDF file.

In order to keep some groups of rows together over page breaks, you need to first create a cell range including the rows that you want to show together in the PDF file. Next, create an instance of the **PrintManager** ('**PrintManager Class** in the on-line documentation) class and use the **paginate()** ('**paginate Method** in the on-line documentation) method to ensure the desired rows are displayed together. When you are done, simply save your PDF file using the **savePageInfosToPDF()** ('**savePageInfosToPDF Method** in the on-line documentation) method.

### Using Code

Refer to the following example code to allow users to keep some rows together over page breaks while exporting to a PDF file.

#### Java

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

// Open Excel file
workbook.open("KeepTogether.xlsx");

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

/* The first page of the natural pagination is from
   row 1st to 36th, the second page is from row 37th to 73rd */
List<IRange> keepTogetherRanges = new ArrayList<IRange>();

/* The row 37th and 38th need to keep together.
```

```
    So the pagination results are: the first page is from row
    1st to 35th, the second page is from row 36th to 73rd*/
keepTogetherRanges.add(worksheet.getRange("36:37"));

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Get the pagination information of the worksheet
List<PageInfo> pages =
printManager.paginate(worksheet, keepTogetherRanges, null);

// Save the modified pages into pdf file
printManager.savePageInfosToPDF("KeepTogether.pdf", pages);
```

## Delete Blank Pages From Middle

While exporting a workbook to a PDF file, sometimes you may encounter a couple of extra pages that are completely blank. In a workbook with large number of worksheets, it is extremely difficult to find out which pages are empty and even more time-consuming to delete them from the middle without impacting the pagination.

In order to avoid printing and publishing of blank pages, GcExcel Java enables users to scan through the pages of the PDF, find out which pages are blank and then exclude the blank pages from the middle while also updating the pagination information accurately.

For removing blank pages from your PDF file, you need to first create an instance of the **PrintManager** ('**PrintManager Class** in the on-line documentation) class and use the **paginate()** ('**paginate Method** in the on-line documentation) method to get the default pagination of the workbook. Now, you can use the **hasPrintContent()** ('**hasPrintContent Method** in the on-line documentation) method to check whether the pages have content or not. Finally, call the **updatePageNumberAndPageSettings()** ('**updatePageNumberAndPageSettings Method** in the on-line documentation) method in order to update the indexes of the page number and the page settings for each page. When you are done, simply save your PDF file using the **savePageInfosToPDF()** ('**savePageInfosToPDF Method** in the on-line documentation) method.

### Using Code

Refer to the following example code to delete blank pages from the middle while exporting to PDF.

```
Java

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

// Open Excel file
workbook.open("DeletingBlankPages.xlsx");

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Get the default pagination information of the workbook
```

```
List<PageInfo> pages = printManager.paginate(workbook);

// Remove empty pages
List<PageInfo> newPages =new ArrayList<PageInfo>();
for(PageInfo page : pages)
{
    // True if there is content in the range to print
    if(printManager.hasPrintContent(page.getPageContent()
        .getRange()))
    {
        newPages.add(page);
    }
}

// Update the page number and the page settings of each page
printManager.updatePageNumberAndPageSettings(newPages);

// Save to PDF file
printManager.savePageInfosToPDF("DeleteBlankPagesFromMiddle.pdf", newPages);
```

## Export Different Headers On Different Pages

GcExcel Java enables users to export different headers on different pages of the PDF file. This feature is useful especially when you have different information on each page of the PDF file and you want to provide different headers to each page of the PDF.

In order to configure different headers for different pages in the PDF file, you can use the **setTitleRowStart()** ('setTitleRowStart Method' in the on-line documentation) method, the **setTitleRowEnd()** ('setTitleRowEnd Method' in the on-line documentation) method, and other methods of the **RepeatSetting** ('RepeatSetting Class' in the on-line documentation) class. When you are done, simply create an instance of the **PrintManager** ('PrintManager Class' in the on-line documentation) class, get the default pagination information using the **paginate()** ('paginate Method' in the on-line documentation) method and finally save your PDF file using the **savePageInfosToPDF()** ('savePageInfosToPDF Method' in the on-line documentation) method.

### Using Code

Refer to the following example code in order to export different headers on different pages while exporting to a PDF file.

```
Java

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

// Open Excel file
workbook.open("MultipleHeaders.xlsx");

// Fetch default worksheet
```

```
IWorksheet worksheet = workbook.getWorksheets().get(0);
List<RepeatSetting> repeatSettings = new ArrayList<RepeatSetting>();

// The title rows of the "B2:F87" is "$2:$2"
RepeatSetting repeatSetting = new RepeatSetting();
repeatSetting.setTitleRowStart(1);
repeatSetting.setTitleRowEnd(1);
repeatSetting.setRange(worksheet.getRange("B2:F87"));
repeatSettings.add(repeatSetting);

// The title rows of the "B89:F146" is "$89:$89"
RepeatSetting repeatSetting2 = new RepeatSetting();
repeatSetting2.setTitleRowStart(88);
repeatSetting2.setTitleRowEnd(88);
repeatSetting2.setRange(worksheet.getRange("B89:F146"));
repeatSettings.add(repeatSetting2);

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();
worksheet.getPageSetup().setRightMargin(10);

// Get the pagination information of the worksheet
List<PageInfo> pages =
printManager.paginate(worksheet, null, repeatSettings);

// Save the modified pages into pdf file
printManager.savePageInfosToPDF("ManageHeadersOnDifferentPages.pdf", pages);
```

## Export Last Page Without Headers

GcExcel Java enables users to export the last page of a PDF file without headers while keeping the headers intact in rest of the pages across the PDF file. For instance - While saving a workbook to a PDF file, you may sometimes have data in the last page of the PDF that doesn't need any headers. In such a scenario, this feature can be helpful in order to save the last page of the PDF without displaying any header information.

In order to export the last page without headers while saving to a PDF file, you need to first get the default pagination by using the **paginate()** (**'paginate Method' in the on-line documentation**) method of the **PrintManager** (**'PrintManager Class' in the on-line documentation**) class. Then, you can use the **setPageContent()** (**'setPageContent Method' in the on-line documentation**) method of the **PageInfo** (**'PageInfo Class' in the on-line documentation**) class and the **setTitleRowStart()** (**'setTitleRowStart Method' in the on-line documentation**) method of the **PageContentInfo** (**'PageContentInfo Class' in the on-line documentation**) class in order to modify the header index of the last page. When you are done, simply save your file using the **savePageInfosToPDF()** (**'savePageInfosToPDF Method' in the on-line documentation**) method.

### Using Code

Refer to the following example code to allow users to save the last page of the PDF without any headers while exporting

to a PDF file.

#### Java

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

// Open Excel file
workbook.open("ExcelData.xlsx");

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Rows to be repeated on the top of each page, while saving pdf
worksheet.getPageSetup().setPrintTitleRows("$1:$2");

// Get the default pagination information of the workbook
List<PageInfo> pages = printManager.paginate(workbook);

// Modify the print header of the last page
pages.get(pages.size() - 1).getPageContent().setTitleRowStart(-1);

// Save the modified pages into pdf file
printManager.savePageInfosToPDF("ExportLastPageWithoutHeaders.pdf", pages);
```

## Export Custom Page Information

GcExcel Java enables users to save and print custom page information while exporting to a PDF file.

For instance - Sometimes, users may want to apply different page settings and display custom page number, page count, title rows, tail rows, column headers, row headers, title columns, tail columns, range, paper width, paper height, page margins, page headers, page footers etc. as per their own preferences while exporting to a PDF file or while printing a PDF file. In this scenario, they can use this feature to showcase the desired page information instead of the default page information in the PDF file.

Depending upon the specific requirements of the users, the custom page information can be exported using the following APIs:

- Creating and using an instance of the **PageInfo ('PageInfo Class' in the on-line documentation)** class - The PageInfo object represents a page containing all the information needed for printing. This includes page number, page count, page content and page settings, etc.
- Creating and using an instance of the **PageContentInfo ('PageContentInfo Class' in the on-line documentation)** class- The PageContentInfo object represents the data area of a page which includes row header, title rows, tail rows, column header, title columns, tail columns, range, etc.



- Creating and using an instance of the **PageSettings ('PageSettings Class' in the on-line documentation)** class- The PageSettings object contains all the methods effecting the page settings including the paper width, paper height, page margins, page header, page footer, etc.

## Using Code

Refer to the following example code to allow users to export custom page information while saving the workbook to a PDF file.

### Java

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

// Open Excel file
workbook.open("KeepTogether.xlsx");

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

/* Get the default pagination information of the worksheet.
   The first page of the natural pagination is "A1:F37",
   the second page is from row "A38:F73" */
List<PageInfo> pages = printManager.paginate(worksheet);

// Get Custom Page Information and Export it to PDF

// The first page is "A1:F36"
pages.get(0).getPageContent().setRange(worksheet.getRange("A1:F36"));

// The center header of the first page will show the text "Budget summary report"
pages.get(0).getPageSettings().setCenterHeader("&KFF0000&18 Budget summary report");

// The center footer of the first page will show the page number "1"
pages.get(0).getPageSettings().setCenterFooter("&KFF0000&16 Page &P");

// The second page is "A37:F73"
pages.get(1).getPageContent().setRange(worksheet.getRange("A37:F73"));

// Save the modified pages into pdf file
printManager.savePageInfosToPDF("CustomPageInfos.pdf", pages);
```

## Export Specific Pages to PDF

GcExcel Java enables users to export only some specific worksheets in the workbook (and not the entire workbook) into the pages of the PDF file.

This feature is useful especially when you have a workbook containing large number of worksheets. For instance - While saving to a PDF file, you may not want to export the entire workbook containing multiple worksheets and want only some important worksheets to be saved to the PDF file. In this scenario, you can use this feature to generate a custom PDF file as per your requirements.

In order to export specific pages to the PDF file, create an instance of the **PrintManager** ('**PrintManager Class**' in the **on-line documentation**) class and get the default pagination using the **paginate()** ('**paginate Method**' in the **on-line documentation**) method. Next, you need to specify the pages that you want to export or print. Finally, call the **updatePageNumberAndPageSettings()** ('**updatePageNumberAndPageSettings Method**' in the **on-line documentation**) method in order to update the indexes of the page number and the page settings for each page. When you are done, simply save your PDF file using the **savePageInfosToPDF()** ('**savePageInfosToPDF Method**' in the **on-line documentation**) method.

## Using Code

Refer to the following example code to export some specific pages to the PDF file.

Java

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

// Open Excel file
workbook.open("PrintSpecificPDFPages.xlsx");

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

// Get the natural pagination information of the workbook
List<PageInfo> pages = printManager.paginate(workbook);

// Pick some pages to print
List<PageInfo> newPages = new ArrayList<PageInfo>();
newPages.add(pages.get(0));
newPages.add(pages.get(2));

/* Update the page number and the page settings of each page.
   The page number is continuous */
printManager.updatePageNumberAndPageSettings(newPages);

// Save the pages into pdf file
printManager.savePageInfosToPDF("PrintSpecificPages.pdf", newPages);
```

## Save Multiple Workbooks to Single PDF

GcExcel Java allows users to save multiple workbooks into a single Portable Document File (PDF) by either using

the **saveWorkbooksToPDF()** ('**saveWorkbooksToPDF Method**' in the **on-line documentation**) method or using the **savePageInfosToPDF()** ('**savePageInfosToPDF Method**' in the **on-line documentation**) method of the **PrintManager** ('**PrintManager Class**' in the **on-line documentation**) class. Each workbook is saved to a new page in the PDF file. The information in the PDF such as the page number, number of pages, odd and even pages, first page etc. is saved on the basis of the final pagination results.

## Advantage of Saving Multiple Workbooks to Single PDF File

This feature is useful especially when you need consolidated information at one place for enhanced analysis and visualization. For instance - let's say you have sales information about different versions of a product in different workbooks. Instead of sharing multiple spreadsheets or PDF files; you can share a combined PDF (by saving all the workbooks to a single PDF file) showcasing the annual sales figures of the product. This will not only help users to analyse all the crucial information at one place but it will also facilitate them in sharing, manipulating and printing all the sales data in an efficient way.

Refer to the following example code in order to export multiple workbooks to a single PDF file using the **saveWorkbooksToPDF()** ('**saveWorkbooksToPDF Method**' in the **on-line documentation**) method.

Java

```
// Initialize first workbook
Workbook workbook1 = new Workbook();

// Opening Excel file
workbook1.open("Book1.xlsx");

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

// Opening Excel file
workbook2.open("Book2.xlsx");

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();
List<IWorkbook> workbooks = new ArrayList<IWorkbook>();
workbooks.add(workbook1);
workbooks.add(workbook2);

printManager.saveWorkbooksToPDF("SaveToOnePDF.pdf", workbooks);
```

Refer to the following example code in order to export multiple workbooks to a single PDF file using the **savePageInfosToPDF()** ('**savePageInfosToPDF Method**' in the **on-line documentation**) method.

Java

```
// Initialize first workbook
Workbook workbook1 = new Workbook();

// Opening Excel file
workbook1.open("Book1.xlsx");
```

```
// Initialize second workbook
Workbook workbook2 = new Workbook();

// Opening Excel file
workbook2.open("Book2.xlsx");

// Create an instance of the PrintManager class
PrintManager printManager = new PrintManager();

workbook1.getWorksheets().get(0).getPageSetup().setCenterFooter("&P of &N");
workbook1.getWorksheets().get(0).getPageSetup().setCenterHeader("&G");
try {
    workbook1.getWorksheets().get(0).getPageSetup().getCenterHeaderPicture()
        .setFilename("C:\\Documents\\GcExcelJAVA-May'19Samples\\logo.png");
} catch (FileNotFoundException e) {

    e.printStackTrace();
}
workbook1.getWorksheets().get(0).getPageSetup().getCenterHeaderPicture().setWidth(150);
workbook1.getWorksheets().get(0).getPageSetup().getCenterHeaderPicture().setHeight(50);
workbook1.getWorksheets().get(0).getPageSetup().setTopMargin(100);

workbook1.getWorksheets().get(1).getPageSetup().setCenterFooter("&P of &N");
workbook1.getWorksheets().get(1).getPageSetup().setCenterHeader("&G");
try {
    workbook1.getWorksheets().get(1).getPageSetup().getCenterHeaderPicture()
        .setFilename("C:\\Documents\\GcExcelJAVA-May'19Samples\\logo.png");
} catch (FileNotFoundException e) {

    e.printStackTrace();
}
workbook1.getWorksheets().get(1).getPageSetup().getCenterHeaderPicture().setWidth(150);
workbook1.getWorksheets().get(1).getPageSetup().getCenterHeaderPicture().setHeight(50);
workbook1.getWorksheets().get(1).getPageSetup().setTopMargin(100);

workbook2.getWorksheets().get(0).getPageSetup().setCenterFooter("&P of &N");
workbook2.getWorksheets().get(0).getPageSetup().setCenterHeader("GrapeCity");
workbook2.getWorksheets().get(0).getPageSetup().setTopMargin(100);

List<PageInfo> pages1 = printManager.paginate(workbook1);
List<PageInfo> pages2 = printManager.paginate(workbook2);

ArrayList<PageInfo> pages = new ArrayList<PageInfo>();
pages.addAll(pages1);
pages.addAll(pages2);

printManager.updatePageNumberAndPageSettings(pages);
```

```
// Save the workbook1 and workbook2 into the PDF file
printManager.savePageInfosToPDF("SaveToOnePDF.pdf", pages);
```

## Export Worksheet to PDF

GcExcel Java provides the option to paginate a worksheet automatically, according to page boundaries, while exporting to PDF file.

The **GetPaginationInfo** method of **PrintManager** class gets an array of the page boundaries for horizontal and vertical paging. The method needs to be called separately for horizontal and vertical pagination. The retrieved pagination information is based on the page setup settings. The print area of the worksheet can also be defined. In case it is not defined, the default area is considered from cell A1 till the last cell where any cell data is present.

In addition to the page setup settings, you can also define ranges which need to be kept together and repeat settings of a range by using the overload of **GetPaginationInfo** method.

### Using Code

Refer to the following example code to paginate a worksheet while exporting to PDF file based on the page setup settings.

#### Java

```
IWorkbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// The row and column headings are printed
worksheet.getPageSetup().setPrintHeadings(true);

// The range "B6:N80" will be printed
worksheet.getPageSetup().setPrintArea("B6:N80");

// Set data
worksheet.getRange("B6:S8").setValue(10);
// Add a table
worksheet.getTables().add(worksheet.getRange("B6:N20"), true);

PrintManager printManager = new PrintManager();

// The columnIdxs is [9, 13], this means that the horizontal direction is split after
the column 10th and 14th
List<Integer> columnIdxs = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Horizontal);

// The rowIdxs is [50, 79], this means that the vertical direction is split after the
row 51th and 80th
```

```
List<Integer> rowIndexs = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Vertical);

worksheet.save("Export.pdf", SaveFileFormat.Pdf);
```

Refer to the following example code to paginate a worksheet while exporting to PDF file based on the page setup settings, range to be kept together and repeat settings.

#### Java

```
// create a new workbook
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// The row and column headings are printed
worksheet.getPageSetup().setPrintHeadings(true);
// The range "B6:N80" will be printed
worksheet.getPageSetup().setPrintArea("B6:N80");
// Set data
worksheet.getRange("B60:N80").setValue(1);
// Add a table
worksheet.getTables().add(worksheet.getRange("B6:N20"), true);
// The row 6th will be printed at the top of each page
ArrayList<RepeatSetting> repeatSettings = new ArrayList<RepeatSetting>();

RepeatSetting repeatSetting = new RepeatSetting();
repeatSetting.setTitleRowStart(5);
repeatSetting.setTitleRowEnd(5);
repeatSetting.setRange(worksheet.getRange("B6:N80"));
repeatSettings.add(repeatSetting);
// The rows from 25th to 60th should be paged to one page
ArrayList<IRange> keepTogetherRanges = new ArrayList<IRange>();

keepTogetherRanges.add(worksheet.getRange("$25:$60"));
PrintManager printManager = new PrintManager();
// The columnIndex is [9, 13], this means that the horizontal direction is
// split after the column 10th and 14th.
List<Integer> columnIndexs = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Horizontal,
keepTogetherRanges, repeatSettings);
// The rowIndexs is [23, 66, 79], this means that the vertical direction is
// split after the row 24th, 67th and 80th.
List<Integer> rowIndexs = printManager.GetPaginationInfo(worksheet,
PaginationOrientation.Vertical,
keepTogetherRanges, repeatSettings);

List<PageInfo> pages = printManager.paginate(worksheet, keepTogetherRanges,
repeatSettings);
```

```
printManager.savePageInfosToPDF("GetPaginationRangesRepeatSettings.pdf", pages);
```

## Export Fills

GcExcel Java allows users to save worksheets possessing cells with different fill styles.

In order to export an excel file with fills to PDF format, refer to the following example code.

Java

```
// Create a new workbook and access the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Merge cells and apply fill style
worksheet.getRange("A1:C2").merge();
worksheet.getRange("A1:C2").getInterior().setColor(Color.GetGreen());

// Save to a pdf file
workbook.save("ExportFills.pdf", SaveFileFormat.Pdf);
```

## Export Picture

GcExcel Java allows users to save worksheets with pictures while exporting to a PDF file.

In order to export an excel file with pictures to PDF format, refer to the following example code.

Java

```
// Create a new workbook and access the default worksheet
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getPageSetup().setOrientation(PageOrientation.Landscape);

FileInputStream stream = null;
try
{
    stream = new FileInputStream("Pictures/logo.png");
}

catch (FileNotFoundException e)
{
    e.printStackTrace();
}
```

```
System.out.println(stream.toString());
IShape picture = null;
try
{
    picture = worksheet.getShapes().addPicture(stream, ImageType.PNG, 20, 20, 690, 100);
}
catch (IOException ioe)
{
}

// Save to a pdf file
workbook.save("ExportPicture.pdf", SaveFileFormat.Pdf);
```

## Export Sparkline

GcExcel Java allows users to save worksheets with sparklines while exporting to a PDF file.

In order to export an excel file with sparklines to PDF format, refer to the following example code.

### Java

```
// Create a new workbook and access the default worksheet
Workbook workbook = new Workbook();
Object[][] data = new Object[][]
{
    { "Customer", "0-30 Days", "30-60 Days", "60-90 Days", ">90 Days" },
    { "Customer A", 1200.15, 1916.18, 1105.23, 1806.53 },
    { "Customer B", 896.23, 1005.53, 1800.56, 1150.49 },
    { "Customer C", 827.63, 1009.23, 1869.23, 1002.56 }
};

IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("B2:E5").setValue(data);
worksheet.getRange("B:F").setColumnWidth(15);
worksheet.getRange("B:E").setHorizontalAlignment(HorizontalAlignment.Center);
ITable table = worksheet.getTables().add(worksheet.getRange("B2:F5"), true);
table.setTableStyle(workbook.getTableStyles().get("TableStyleMedium3"));
table.getColumns().get(4).setName("Sparklines");

// Create a new group of sparklines.
worksheet.getRange("F3").getSparklineGroups().add(SparkType.Line, "C3:E3");
worksheet.getRange("F4").getSparklineGroups().add(SparkType.Column, "C4:E4");
worksheet.getRange("F5").getSparklineGroups()
    .add(SparkType.ColumnStacked100, "C5:E5");

// Save to a pdf file
```



```
workbook.save("ExportSparklines.pdf", SaveFileFormat.Pdf);
```

## Export Table

GcExcel Java allows users to save worksheets with tables while exporting to a PDF file.

In order to export an excel file with tables to PDF format, refer to the following example code.

### Java

```
// Create a new workbook and access the default worksheet
Workbook workbook = new Workbook();
IWorksheet sheet = workbook.getWorksheets().get(0);

// Add Table
ITable table = sheet.getTables().add(sheet.getRange("B5:G16"), true);
table.setShowTotals(true);

// Set values to the range
int[] data = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
sheet.getRange("C6:C16").setValue(data);
sheet.getRange("D6:D16").setValue(data);

// Set total functions
table.getColumns().get(1).setTotalsCalculation(TotalsCalculation.Average);
table.getColumns().get(2).setTotalsCalculation(TotalsCalculation.Sum);

// Create custom table style
ITableStyle customTableStyle = workbook.getTableStyles()
    .get("TableStyleMedium10").duplicate();

ITableStyleElement wholeTableStyle = customTableStyle.getTableStyleElements()
    .get(TableStyleElementType.WholeTable);
wholeTableStyle.getFont().setItalic(true);
wholeTableStyle.getBorders().get(BordersIndex.EdgeTop)
    .setThemeColor(ThemeColor.Accent1);
wholeTableStyle.getBorders().get(BordersIndex.EdgeTop)
    .setLineStyle(BorderLineStyle.Thick);
wholeTableStyle.getBorders().get(BordersIndex.EdgeRight)
    .setThemeColor(ThemeColor.Accent1);
wholeTableStyle.getBorders().get(BordersIndex.EdgeRight)
    .setLineStyle(BorderLineStyle.Thick);
wholeTableStyle.getBorders().get(BordersIndex.EdgeBottom)
    .setThemeColor(ThemeColor.Accent1);
wholeTableStyle.getBorders().get(BordersIndex.EdgeBottom)
    .setLineStyle(BorderLineStyle.Thick);
```

```
wholeTableStyle.getBorders().get(BordersIndex.EdgeLeft)
    .setThemeColor(ThemeColor.Accent1);
wholeTableStyle.getBorders().get(BordersIndex.EdgeLeft)
    .setLineStyle(BorderLineStyle.Thick);

ITableStyleElement firstRowStripStyle = customTableStyle.getTableStyleElements()
    .get(TableStyleElementType.FirstRowStripe);
firstRowStripStyle.getFont().setBold(true);

// Apply custom style to table
table.setTableStyle(customTableStyle);

// Save to a pdf file
workbook.save("ExportTable.pdf", SaveFileFormat.Pdf);
```

## Export Text

GcExcel Java allows users to save worksheets with different text formats and font effects while exporting to a PDF file.

You can save an excel file with different text formats and font effects in the following ways:

1. **Export Number Formats**
2. **Export Overflow Text**
3. **Export Font Effects**

### Export Number Formats

In order to export an excel file to PDF having text with excel number formatter, refer to the following example code.

#### Java

```
// Create a new workbook and access the default worksheet
Workbook workbook = new Workbook();
IWorksheet sheet = workbook.getWorksheets().get(0);

// Assign values to the range
sheet.getRange("B3:B7").setValue(123456.789);
sheet.getRange("B9:B13").setValue(-123456.789);

// Set number format
sheet.getRange("B4, B10").setNumberFormat("0.00;[Red]0.00");
sheet.getRange("B5, B11").setNumberFormat("$#,##0.00;[Red]$#,##0.00");
sheet.getRange("B6, B12").setNumberFormat("0.00E+00");
sheet.getRange("B7, B13")
    .setNumberFormat("_($* #,##0.00_);_($* (#,##0.00);_($* \" - \"??_);_(@_)");

// Customize other settings
```

```
sheet.getColumns().get(1).setColumnWidthInPixel(100);

// Save to a pdf file
workbook.save("ExportNumberFormats.pdf", SaveFileFormat.Pdf);
```

## Export Overflow Text

In order to export an excel file to PDF having overflow text, refer to the following example code.

### Java

```
// Settings for overflow text
sheet.getRange("F2, F4").setValue("This is a test string of overflow");
sheet.getRange("F6, F8").setValue("This is a test string of overflow with right alignment");
sheet.getRange("F6, F8").setHorizontalAlignment(HorizontalAlignment.Right);
sheet.getRange("D8, H4").setValue(123);

// Apply style
sheet.getRange("A1:J10").getBorders().setLineStyle(BorderLineStyle.Thin);

// Save to a pdf file
workbook.save("ExportOverflowText.pdf", SaveFileFormat.Pdf);
```

## Export Font Effects

In order to export an excel file to PDF having font effects such as text alignment, wordwrap, text indent, shrink to fit, underline and strikethrough, refer to the following example code.

### Java

```
// Set Alignment
sheet.getRange("A1").setValue("Alignment");
sheet.getRange("B2").setValue("Left Alignment");
sheet.getRange("B2").setHorizontalAlignment(HorizontalAlignment.Left);
sheet.getRange("C2").setValue("Center Alignment");
sheet.getRange("C2").setHorizontalAlignment(HorizontalAlignment.Center);
sheet.getRange("D2").setValue("Right Alignment");
sheet.getRange("D2").setHorizontalAlignment(HorizontalAlignment.Right);
sheet.getRange("B3").setValue("Top Alignment");
sheet.getRange("B3").setVerticalAlignment(VERTICAL_ALIGNMENT.Top);
sheet.getRange("C3").setValue("Middle Alignment");
sheet.getRange("C3").setVerticalAlignment(VERTICAL_ALIGNMENT.Center);
sheet.getRange("D3").setValue("Bottom Alignment");
sheet.getRange("D3").setVerticalAlignment(VERTICAL_ALIGNMENT.Bottom);
sheet.getRange("B4").setValue(
    "Test String. \nThis is a test string for Justify Alignment. ");
sheet.getRange("B4").setHorizontalAlignment(HorizontalAlignment.Justify);
sheet.getRange("B4").setVerticalAlignment(VERTICAL_ALIGNMENT.Justify);
```

```
sheet.getRange("C4").setValue(
    "Test String. \nThis is a test string for Distributed Alignment. ");
sheet.getRange("C4").setHorizontalAlignment(HorizontalAlignment.Distributed);
sheet.getRange("C4").setVerticalAlignment(VERTICAL_ALIGNMENT_DISTRIBUTED);

// Set wordwrap
sheet.getRange("A6").setValue("Wordwrap");
sheet.getRange("B7").setValue("This is a test string for Wordwrap");
sheet.getRange("C7").setValue("This is a test string \n for Wordwrap");
sheet.getRange("B7:C7").setWrapText(true);

// Set text indent
sheet.getRange("A9").setValue("Indent");
sheet.getRange("B10").setValue("Left Indent");
sheet.getRange("B10").setIndentLevel(3);
sheet.getRange("C10").setValue("Right Indent");
sheet.getRange("C10").setIndentLevel(3);
sheet.getRange("C10").setHorizontalAlignment(HorizontalAlignment.Right);

// Apply Shrink to fit
sheet.getRange("A12").setValue("Shrink to fit");
sheet.getRange("B13").setValue("This is a test string for \"Shrink to fit\"");
sheet.getRange("B13").setShrinkToFit(true);

// Set Underline
sheet.getRange("A15").setValue("Underline");
sheet.getRange("B16").setValue("Single Underline");
sheet.getRange("B16").getFont().setUnderline(UnderlineType.Single);

// Use Strikethrough
sheet.getRange("A18").setValue("Strikethrough");
sheet.getRange("B19").setValue("Strikethrough");
sheet.getRange("B19").getFont().setStrikethrough(true);

// Customize other settings
sheet.getColumns().get(0).getFont().setBold(true);
sheet.getColumns().get(0).setColumnWidthInPixel(100);
sheet.getColumns().get(1).setColumnWidthInPixel(200);
sheet.getColumns().get(2).setColumnWidthInPixel(245);
sheet.getColumns().get(3).setColumnWidthInPixel(234);
sheet.getRows().get(2).setRowHeightInPixel(72);
sheet.getRows().get(3).setRowHeightInPixel(123);
sheet.getRows().get(6).setRowHeightInPixel(48);
sheet.getRange("A1:D19").getBorders().setLineStyle(BorderLineStyle.Thin);
sheet.getPageSetup().setPaperSize(PaperSize.A3);

// Save to a pdf file
workbook.save("ExportFontEffects.pdf", SaveFileFormat.Pdf);
```

## Export Vertical Text

GcExcel Java allows users to export Excel files with vertical text to PDF without any issues.

While saving an Excel file with vertical text correctly to a PDF file, the following properties can be used -

- IRange.Orientation - The **setOrientation()** ('**setOrientation Method**' in the on-line documentation) method of the **IRange** ('**IRange Interface**' in the on-line documentation) interface sets the orientation of the text.
- IRange.Font.Name - Sets the specific font name using the **getFont** ('**getFont Method**' in the on-line documentation) method of the **IRange** interface. If the font name starts with "@", each double-byte character in the text is rotated to 90 degrees.

Refer to the following example code in order to export vertical text to PDF.

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Fetch the cell range A1
IRange a1 = worksheet.getRange("A1");

// Setting Cell A1 Text
a1.setValue("This is a vertical text");

// Formatting A1 cell
a1.getFont().setName("Verdana");
a1.setHorizontalAlignment(HorizontalAlignment.Right);
a1.setVerticalAlignment(VerticalAlignment.Top);
a1.setOrientation(90);
a1.setWrapText(true);
a1.setColumnWidth(27);
a1.setRowHeight(190);

// Saving workbook to PDF
workbook.save("6- ExportVerticalTextToPDF.pdf", SaveFileFormat.Pdf);
```



**Note:** The following limitations must be kept in mind while exporting Excel files with vertical text to PDF -

- The orientation can only be set to 0, 90, -90 and 255. Other values will be treated as 0 while rendering the PDF file.

- If the font name starts with "@" and the orientation is 255, GcExcel will ignore the "@".

## Shrink To Fit With Text Wrap

GcExcel Java allows users to apply the shrink to fit feature in a cell along with the wrapped text. This feature reduces the font size of the text automatically so that it fits inside the cells of the spreadsheet without wrapping.

### Advantage of Using Shrink To Fit Feature

The Shrink to Fit feature implemented with wrapped text is useful especially when you need to deal with spreadsheets possessing tightly constrained layouts with vertical spaces and wrapped text. Also, this feature can be used when users don't want to opt for Auto fit row height and column width option to adjust the column width and row height as per their preferred worksheet layout.

The following points should be kept in mind while working with the shrink to fit feature :

- If you're exporting your Excel files to a pdf file or stream, the **PdfSaveOptions** ('PdfSaveOptions Class' in the on-line documentation) class can be used to configure the save settings.
- In order to get or set the settings about enabling the shrink to fit feature on the wrapped text, you can use the **getShrinkToFitSettings()** ('getShrinkToFitSettings Method' in the on-line documentation) method of the **PdfSaveOptions** class.
- The **getCanShrinkToFitWrappedText()** ('getCanShrinkToFitWrappedText Method' in the on-line documentation) and **setCanShrinkToFitWrappedText()** ('setCanShrinkToFitWrappedText Method' in the on-line documentation) methods of the **IShrinkToFitSettings** ('IShrinkToFitSettings Interface' in the on-line documentation) interface can be used to get or set whether to apply the shrink to fit feature on the wrapped text. If the value is true, the font size of the wrapped text may be reduced so that the wrapped text can be fully displayed.
- The **getMinimumFont()** ('getMinimumFont Method' in the on-line documentation) and the **setMinimumFont()** ('setMinimumFont Method' in the on-line documentation) methods of the **IShrinkToFitSettings** interface can be used to get or set the minimum font size while enabling the shrink to fit feature.
- The **getEllipsis()** ('getEllipsis Method' in the on-line documentation) and the **setEllipsis()** ('setEllipsis Method' in the on-line documentation) methods of the **IShrinkToFitSettings** interface can be used to get or set the omitted string if the wrapped text is not fully displayed. This can be used with the **getMinimumFont** and **setMinimumFont** methods.

### Using Code

Refer to the following example code in order to allow users to use the shrink to fit feature with text wrap.

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getPageSetup().setPrintGridlines(true);
worksheet.getRange("A1").setRowHeightInPixel(20);
```

```
worksheet.getRange("A1").setColumnWidthInPixel(90);
worksheet.getRange("A1").setWrapText(true);
worksheet.getRange("A1").setShrinkToFit(true);
worksheet.getRange("A1").setValue("GrapeCity Documents For Excel");

// Setting PdfSaveOptions
PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
pdfSaveOptions.getShrinkToFitSettings().setCanShrinkToFitWrappedText(true);
pdfSaveOptions.getShrinkToFitSettings().setMinimumFont(10);
pdfSaveOptions.getShrinkToFitSettings().setEllipsis("~");

// Saving the workbook to pdf
workbook.save("ShrinkToFitWrappedText.pdf", pdfSaveOptions);
```

## Export Slicers

Slicers are visual filters that are used to filter data in Excel spreadsheets. You can filter the data by clicking on desired type of data in slicer.

GcExcel supports the export of Excel spreadsheet containing a slicer to PDF document. So, if an Excel spreadsheet containing a slicer is exported to PDF, the resulting PDF will contain the applied slicer.

### Using Code

Refer to the following example code to export slicers to PDF document.

#### Java

```
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Set Data
worksheet.getRange("A1:F16").setValue(sourceData);
worksheet.getRange("A:F").setColumnWidth(15);

ITable table = worksheet.getTables().add(worksheet.getRange("A1:F16"), true);
table.getColumns().get(3).getDataBodyRange().setNumberFormat("$#,##0.00");
// Create slicer cache for table
ISlicerCache cache = workbook.getSlicerCaches().add(table, "Category", "categoryCache");

// Add two slicers for Category column
@SuppressWarnings("unused")
ISlicer slicer1 = cache.getSlicers().add(workbook.getWorksheets().get("Sheet1"),
    "catel", "Category", 300, 50,
    100, 200);

// Or we can just open the Excel having slicers and then export it
// workbook.Open("ExcelContainingSlicers.xlsx");
```

```
// Saving workbook to pdf
workbook.save("13-ConvertExcelSlicersToPDFExport.pdf");
```

### Limitations

The following is not supported while exporting slicers to PDF documents:

- Pivot table slicers or report connections
- Custom height of slicer items
- Slicer settings
- Slicer styles (except the color property)
- Slicer header styles
- Scroll viewer which surrounds the items panel
- Slicer item styles for the "No data" visual state group

## Export Signature Lines

GcExcel supports exporting signature lines to PDF documents. The signature lines are exported as images and the exported signature line is different depending upon the validity of certificate. This validity or invalidity is decided by **setSkipCertificateValidationOnExporting** method of **ISignatureSet** interface. Its default value is true, meaning that the certificate will be treated as valid. However, you can set it to false to validate the certificate which requires an internet connection.

### Using Code

Refer to the following example code to export signature lines to a PDF document.

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

workbook.open("Signature.xlsx");
workbook.getSignatures().setSkipCertificateValidationOnExporting(false);

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

## Support Security Options

Sometimes, users need to secure their digital documents with user/owner passwords, print permission, content permission, annotation permission etc. PDF documents have always been the preferred format for sharing digital files among professionals. The GcExcel library supports Security Options while saving Excel spreadsheets to PDF files. It helps in securing a PDF Document by restricting the PDF's access to unauthorized users as per the options specified.

With GcExcel's **PdfSecurityOptions** ('**PdfSecurityOptions Class**' in the on-line documentation) class, you can restrict access to your PDF document, while converting Excel spreadsheet to PDF document. You can choose through the following security properties in the **PdfSecurityOptions** class:



Properties	Description
<b>getUserPassword</b> ('getUserPassword Method' in the on-line documentation) / <b>setUserPassword</b> ('setUserPassword Method' in the on-line documentation)	Gets or sets the user password of the PDF document.
<b>getOwnerPassword</b> ('getOwnerPassword Method' in the on-line documentation) / <b>setOwnerPassword</b> ('setOwnerPassword Method' in the on-line documentation)	Gets or sets the owner password of the PDF document. This password is required to change the permissions for the PDF document.
<b>getPrintPermission</b> ('getPrintPermission Method' in the on-line documentation) / <b>setPrintPermission</b> ('setPrintPermission Method' in the on-line documentation)	Gets or sets the permission to print the PDF document. The default value is true for this property.
<b>getFullQualityPrintPermission</b> ('getFullQualityPrintPermission Method' in the on-line documentation) / <b>setFullQualityPrintPermission</b> ('setFullQualityPrintPermission Method' in the on-line documentation)	Gets or sets the permission to print in high quality. The default value is true for this property, and it only works when <b>PrintPermission</b> property is set to true.
<b>getExtractContentPermission</b> ('getExtractContentPermission Method' in the on-line documentation) / <b>setExtractContentPermission</b> ('setExtractContentPermission Method' in the on-line documentation)	Gets or sets the permission to copy or extract content. The default value is true for this property.
<b>getModifyDocumentPermission</b> ('getModifyDocumentPermission Method' in the on-line documentation) / <b>setModifyDocumentPermission</b> ('setModifyDocumentPermission Method' in the on-line documentation)	Gets or sets the permission to modify the PDF document. The default value is true for this property.
<b>getAssembleDocumentPermission</b> ('getAssembleDocumentPermission Method' in the on-line documentation) / <b>setAssembleDocumentPermission</b> ('setAssembleDocumentPermission Method' in the on-line documentation)	Gets or sets the permission to insert, rotate or delete pages, and create bookmarks/thumbnail images. The default value is true for this property. If you want to prevent a user from inserting, rotating or deleting pages, you need to set <b>ModifyDocumentPermission</b> property to false as well.
<b>getModifyAnnotationsPermission</b> ('getModifyAnnotationsPermission Method' in the on-line documentation) / <b>setModifyAnnotationsPermission</b> ('setModifyAnnotationsPermission Method' in the on-line documentation)	Gets or sets the permission to modify text annotations and fill the form fields. The default value is true for this property.
<b>getFillFormsPermission</b> ('getFillFormsPermission Method' in the on-line documentation) / <b>setFillFormsPermission</b> ('setFillFormsPermission Method' in the on-line documentation)	Gets or sets the permission to fill the form fields even if the <b>ModifyAnnotationsPermission</b> property returns false. The default value for this property is true. Note that if you want to

prevent a user from filling interactive form fields, you need to set the **ModifyAnnotationsPermission** property to false.

## Using Code

Refer to the following example to add security options while exporting Excel spreadsheets to PDF documents.

### Java


```
// Initialize workbook
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
Object data = new Object[][] { { "Name", "City", "Sex", "Weight", "Height", "Age" },
    { "Bob", "NewYork", "male", 80, 180, 56 }, { "Betty", "NewYork", "female", 72,
168, 45 },
    { "Gary", "NewYork", "male", 71, 179, 50 }, { "Hunk", "Washington", "male", 80,
171, 59 },
    { "Cherry", "Washington", "female", 58, 161, 34 }, { "Coco", "Virginia",
"female", 58, 181, 45 },
    { "Lance", "Chicago", "female", 49, 160, 57 }, { "Eva", "Washington", "female",
71, 180, 81 } };

worksheet.setName("Listing");
// Set data
worksheet.getRange("A1:G9").setValue(data);

// The security settings of pdf when converting excel to pdf
PdfSecurityOptions securityOptions = new PdfSecurityOptions();
// Sets the user password
securityOptions.setUserPassword("user");
// Sets the owner password
securityOptions.setOwnerPassword("owner");
// Printing the pdf document is not allowed
securityOptions.setPrintPermission(false);
// Filling the form fields of the pdf document is not allowed
securityOptions.setFillFormsPermission(false);

PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
// Sets the security settings of the pdf
pdfSaveOptions.setSecurityOptions(securityOptions);

// Saving workbook to PDF
workbook.save("4-SavePDFPdfSecurityOptions.pdf", pdfSaveOptions);
```

 **Note:** GcExcel uses RC4 encryption with key from 40 to 128 bit length and allows to define additional permission flags.

## Support Document Properties

GcExcel provides support for document properties while saving Excel spreadsheets to PDF documents. The document properties contain the basic information about a document, such as title, author, creation date, subject, creator, version etc. You can store such useful information in the exported PDF document.

The **DocumentProperties** ('DocumentProperties Class' in the on-line documentation) class contains the methods such as **setPdfVersion** ('setPdfVersion Method' in the on-line documentation), **setTitle** ('setTitle Method' in the on-line documentation), **setAuthor** ('setAuthor Method' in the on-line documentation), **setSubject** ('setSubject Method' in the on-line documentation), **setKeywords** ('setKeywords Method' in the on-line documentation), **setCreator** ('setCreator Method' in the on-line documentation), **setProducer** ('setProducer Method' in the on-line documentation), **setCreationDate** ('setCreationDate Method' in the on-line documentation) and **setModifyDate** ('setModifyDate Method' in the on-line documentation).

### Using Code

Refer to the following example code to add document properties in the exported PDF document.

Java

```
DocumentProperties documentProperties = new DocumentProperties();
// Sets the name of the person that created the document
documentProperties.setAuthor("Will Smith");
// Sets the title of the document
documentProperties.setTitle("Document Properties Info Sample");
// Sets the subject of the document
documentProperties.setSubject("PDF created from GcExcel");

PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
// Sets the DocumentProperties of the pdf
pdfSaveOptions.setDocumentProperties(documentProperties);
```

## Adjust Column Width and Row Height

GcExcel provides **BestFitColumns** and **BestFitRows** properties in the **IPageSetup** interface to properly display long or large-size font texts in cells while printing PDF documents. These properties support SSJSON I/O and are consistent with SpreadJS. The **BestFitColumns** property when set to True, resizes the column width to fit the text with the longest width for printing. Similarly, the **BestFitRows** property when set to True, resizes the row height to fit the text with the tallest height for printing.



**Note:** GcExcel preserves useMax property during JSON I/O. In case a file contains large amount of data, these properties may not work as expected.

### Using Code

Refer to the following example code to adjust column width or row height.

Java

```
// Set bestFitColumns/bestFitRows as true
worksheet.getPageSetup().setBestFitColumns(true);
worksheet.getPageSetup().setBestFitRows(true);
```

## Support Sheet Background Image

GcExcel supports sheet background image which can be included while exporting the worksheet to a PDF file. This is very useful for displaying company logos and watermarks in PDF documents.

### Render Background Image

In a worksheet, you can set a background image using the **setBackgroundPicture** ('setBackgroundPicture Method' in the on-line documentation) method of the **IWorksheet** ('IWorksheet Interface' in the on-line documentation) interface.

GcExcel provides the **setPrintBackgroundPicture** ('setPrintBackgroundPicture Method' in the on-line documentation) method in **PdfSaveOptions** ('PdfSaveOptions Class' in the on-line documentation) class to render the background image in the center of the page while exporting worksheet to PDF file.

Refer to the following example code to include sheet background image while exporting to PDF document.

Java

```
Workbook workbook = new Workbook();
// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
worksheet.getRange("A1").setValue("GrapeCity Documents for Excel");
worksheet.getRange("A1").getFont().setSize(25);

// Load an image from a specific file in input stream
InputStream inputStream = ClassLoader.getResourceAsStream("grapecity.png");
try {
    byte[] bytes = new byte[inputStream.available()];
    // Read an image from input stream
    inputStream.read(bytes, 0, bytes.length);

    // Add background image of the worksheet
    worksheet.setBackgroundPicture(bytes);
} catch (IOException ioe) {
    ioe.printStackTrace();
}

PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
// Print the background picture in the centre of exported pdf file
pdfSaveOptions.setPrintBackgroundPicture(true);

// Saving workbook to pdf
workbook.save("12-PrintBackgroundPicture.pdf", pdfSaveOptions);
```

### Render Multiple Background Images

Multiple background images can be rendered in GcExcel using the **getBackgroundPictures** method of the **IWorksheet** interface. These images can be included while exporting the worksheet to PDF documents. The background images in PDF are drawn based on the gridlines and can be positioned anywhere in the document by specifying the coordinates of the destination rectangle.

Further, the image transparency, border, corner radius and other formatting options can also be applied. For setting the corner radius, the minimum value is 0 and the maximum value is the height or width (whichever is smaller) of the destination rectangle divided by two. The **ImageLayout** enum can be used to specify the way the image should be placed to fill the destination rectangle in PDF.

However, the image is discarded while exporting to Excel.

Refer to the following example code to include multiple background images while exporting to PDF document.

#### Java

```
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

//Add a background picture in the worksheet
IBackgroundPicture picture1 =
worksheet.getBackgroundPictures().addPictureInPixel("image.png", 10, 10, 250, 150);
IBackgroundPicture picture2 =
worksheet.getBackgroundPictures().addPictureInPixel("ConvertShapeToImage.png", 180, 10,
150, 100);

//Set the border style of the destination rectangle.
picture1.getLine().getColor().setRGB(Color.GetRed());
picture1.getLine().setWeight(1);

//The background picture will be resized to fill the destination dimensions.The aspect
ratio is not preserved.
picture1.setBackgroundImageLayout(ImageLayout.Tile);

//Sets the rounded corner of the destination rectangle.
picture1.setCornerRadius(50);
//Sets the transparency of the background picture.
picture1.setTransparency(0.5);
picture2.setTransparency(0.5);

//Save to PDF file
workbook.save("ExportBackgroundImageToPDF.pdf");
```

For more information about adding a background image to a worksheet, refer the [Customize Worksheets](#) topic.

## Support Background Color Transparency

When backcolor is applied on a cell or range, any background image or data gets hidden behind it while exporting to PDF.

GcExcel allows you to make the cell's backcolor transparent when exported to PDF by using the **setPrintTransparentCell** method of the **PdfSaveOptions ('PdfSaveOptions Class' in the on-line documentation)** class. The default value of this property is false. When set to true, it prints the transparency of the cell's background color which makes any background image or data visible.

Refer to the following example code to make cell's backcolor transparent to view the background image in PDF document.

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

worksheet.getRange("A1:K20").getInterior().setColor(Color.FromArgb(50, 0, 255, 255));

// Add a background picture
IBackgroundPicture picture =
worksheet.getBackgroundPictures().addPictureInPixel("image.png", 0, 0, 300, 200);

// Set the transparency of cell's background color, so the background picture will come
out to the front
PdfSaveOptions pdfSaveOptions = new PdfSaveOptions();
pdfSaveOptions.setPrintTransparentCell(true);

// Save to pdf file
workbook.save("PrintTransparentCell.pdf", pdfSaveOptions);
```


## Export to an HTML File

Many organizations maintain their product inventories, hiring positions, price lists etc. in Excel files. However, it is very convenient to publish such data on websites to share it with relevant customers. Hence, exporting to HTML files becomes an important feature in such cases.

GcExcel allows users to export a workbook, worksheet or any specific range to an HTML file. By default, it exports an HTML file and a folder containing additional files. These additional files can be images in a worksheet, htm files of other worksheets in a workbook or css file used for styling the html files. However, a single HTML file can also be exported while exporting a worksheet or any range of a worksheet.

With various methods in **HtmlSaveOptions** class, the exported content can be controlled in various ways like exporting headings, gridlines, document properties or apply other settings like scalable width, page title, displaying tooltip text etc.

The **setExportCssSeparately** method in **HtmlSaveOptions** class exports the css file separately (in the additional folder), as its default value is true. However, it can be set to false so that the css style data is exported directly to each worksheet and separate css file is not created.

 **Note:** If unlicensed version of GcExcel is used:

- While exporting a workbook to HTML: An "Evaluation warning" sheet is appended to the workbook along with an "Evaluation warning" message at the head of each worksheet.
- While exporting a worksheet or range to HTML: An "Evaluation warning" message is added at the head of worksheet or range file.

### Export workbook to HTML

The **save** method of **IWorkbook** interface can be used to export a workbook to HTML file.

Refer to the following example code to export workbook to a zip folder containing workbook's HTML file and a folder carrying additional files.

C#

```
// Create a zip file stream
FileOutputStream outputStream = new FileOutputStream("SaveWorkbookToHTML.zip");
// Create a new workbook
Workbook workbook = new Workbook();
workbook.open("NetProfit.xlsx");
// Save workbook to html format
workbook.save(outputStream, SaveFileFormat.Html);
```

### Export worksheet to HTML

The **save** method of **IWorkbook** interface can be used to export a worksheet to HTML file. The headings and gridlines of the worksheet can also be exported by using **setExportHeadings** and **setExportGridlines** methods of **HtmlSaveOptions** class. The **setExportSheetName** method can be used to define which worksheet needs to be exported.

Refer to the following example code to export worksheet to a zip folder containing worksheet's HTML file and a folder carrying additional files.

Java

```
// Create a zip file stream
FileOutputStream outputStream = null;
outputStream = new FileOutputStream("SaveWorksheetToHTML.zip");

// Create a new workbook
Workbook workbook = new Workbook();
workbook.open("ProjectTracker.xlsx");
HtmlSaveOptions options = new HtmlSaveOptions();

// Set exporting row/column headings
options.setExportHeadings(true);

// Set exporting gridlines
options.setExportGridlines(true);

// Export first sheet
options.setExportSheetName(workbook.getWorksheets().get(0).getName());
```

```
// Set exported html file name
options.setExportFileName("HiringDetails");
workbook.save(outputStream, options);
```

A worksheet can also be exported to a single HTML file when the specific methods of **HtmlSaveOptions** class are set, as in the code below.

#### Java

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

// Open an xlsx file
workbook.open("ProjectTracker.xlsx");

// Create HtmlSaveOptions
HtmlSaveOptions options = new HtmlSaveOptions();

// Export first sheet
options.setExportSheetName(workbook.getWorksheets().get(0).getName());

// Set exported image as base64
options.setExportImageAsBase64(true);

// Set exported css style in html file
options.setExportCssSeparately(false);

// Set not to export single tab in html
options.setExportSingleTab(false);

// Save first worksheet to html
workbook.save("SaveWorksheetToSingleHTML.html", options);
```

### Export worksheet range to HTML

The **save** method of **IWorkbook** interface can be used to export any range of a worksheet to HTML file. The **setExportArea** method of **HtmlSaveOptions** class can be used to define the range which needs to be exported.

Refer to the following example code to export range in a worksheet to a zip folder containing range's HTML file and a folder carrying additional files.

#### Java

```
// Get detail range and set style.
for (IPivotLine item : pivottable.getPivotRowAxis().getPivotLines()) {
    if (item.getLineType() == PivotLineType.Subtotal) {

item.getPivotLineCells().get(0).getRange().getInterior().setColor(Color.GetGreenYellow());
```



```
}  
}
```

A range in a worksheet can also be exported to a single HTML file when the specific methods of **HtmlSaveOptions** class are set, as in the code below.

#### Java

```
// Create a new workbook  
Workbook workbook = new Workbook();  
workbook.open("ProjectTracker.xlsx");  
  
// Create HtmlSaveOptions  
HtmlSaveOptions options = new HtmlSaveOptions();  
  
// Specify exported sheet name  
options.setExportSheetName(workbook.getWorksheets().get(0).getName());  
  
// Set export area  
options.setExportArea("D2:G23");  
  
// Set exported image as base64  
options.setExportImageAsBase64(true);  
  
// Set exported css style in html file  
options.setExportCssSeparately(false);  
  
// Set not to export single tab in html  
options.setExportSingleTab(false);  
  
// Save the specified range of first worksheet to html  
workbook.save("WorksheetRangeToHTML.html", options);
```

#### Limitations

The following features are not supported while exporting to HTML file:

- Charts
- Gradient Fill
- Slicers
  - Pivot table slicers or report connections
  - Custom height of slicer items
  - Slicer settings
  - Slicer styles (except the color property)
  - Slicer header styles
  - Scroll viewer which surrounds the items panel
  - Slicer item styles for the "No data" visual state groups

## Working With Page Setup

GcExcel Java allows users to paginate each worksheet using the methods of the **IPageSetup** ('IPageSetup Interface' in the on-line documentation) interface.

While exporting a spreadsheet to a PDF file, you can customize the page size, print area, print title rows, print title columns; set horizontal page breaks, vertical page breaks, the maximum number of pages for horizontal and vertical pagination etc. along with zoom and scale factors as per your preferences.

In order to set pagination in a worksheet, users can explore the following methods of the **IPageSetup** ('IPageSetup Interface' in the on-line documentation) interface and the **IWorksheet** ('IWorksheet Interface' in the on-line documentation) interface:

Methods	Descriptions
<b>IPageSetup.getPaperSize</b> ('getPaperSize Method' in the on-line documentation) <b>IPageSetup.setPaperSize</b> ('setPaperSize Method' in the on-line documentation)	Used to get or set the size of each page. For more information on implementation, refer to <a href="#">Configure Paper Size</a> .
<b>IPageSetup.getOrientation</b> ('getOrientation Method' in the on-line documentation) <b>IPageSetup.setOrientation</b> ('setOrientation Method' in the on-line documentation)	Used to get or set whether the worksheet should be printed in landscape orientation or portrait orientation. For more information on implementation, refer to <a href="#">Configure Page Orientation</a> .
<b>IPageSetup.getPrintTitleRows</b> ('getPrintTitleRows Method' in the on-line documentation) <b>IPageSetup.setPrintTitleRows</b> ('setPrintTitleRows Method' in the on-line documentation)	Used to get or set the rows that you want to print at the top of each page. For more information on implementation, refer to <a href="#">Configure Rows to Repeat at Top</a> .
<b>IPageSetup.getPrintTitleColumns</b> ('getPrintTitleColumns Method' in the on-line documentation) <b>IPageSetup.setPrintTitleColumns</b> ('setPrintTitleColumns Method' in the on-line documentation)	Used to get or set the columns that you want to print at the left of each page. For more information on implementation, refer to <a href="#">Configure Columns to Repeat at Left</a> .
<b>IPageSetup.getPrintArea</b> ('getPrintArea Method' in the on-line documentation) <b>IPageSetup.setPrintArea</b> ('setPrintArea Method' in the on-line documentation)	Used to get or set the print area in a worksheet. If the print area is not specified by the user, the used range of the sheet is printed by default. For more information on implementation, refer to <a href="#">Configure Print Area</a> .
<b>IPageSetup.getZoom</b> ('getZoom Method' in the on-line documentation) <b>IPageSetup.setZoom</b> ('setZoom Method' in the on-line documentation)	Used to get or set the result of zoom while paginating a worksheet. For more information on implementation, refer to <a href="#">Configure Paper Scaling</a> .
<b>IPageSetup.getFitToPagesWide</b>	Used to get or set the maximum number of pages for horizontal

<p><b>('getFitToPagesWide Method' in the on-line documentation)</b></p> <p><b>IPageSetup.setFitToPagesWide</b> (<b>'setFitToPagesWide Method' in the on-line documentation</b>)</p>	<p>pagination. After this method is set, the worksheet can be scaled to fit all columns to the pages. For more information on implementation, refer to <a href="#">Configure Paper Scaling</a>.</p>
<p><b>IPageSetup.getFitToPagesTall</b> (<b>'getFitToPagesTall Method' in the on-line documentation</b>)</p> <p><b>IPageSetup.setFitToPagesTall</b> (<b>'setFitToPagesTall Method' in the on-line documentation</b>)</p>	<p>Used to get or set the maximum number of pages for vertical pagination. After this method is set, the worksheet can be scaled to fit all rows to the pages. For more information on implementation, refer to <a href="#">Configure Paper Scaling</a>.</p>
<p><b>IPageSetup.getIsPercentScale</b> (<b>'getIsPercentScale Method' in the on-line documentation</b>)</p> <p><b>IPageSetup.setIsPercentScale</b> (<b>'setIsPercentScale Method' in the on-line documentation</b>)</p>	<p>Used to get or set a boolean value to control how the worksheet is scaled while exporting to PDF.</p> <p>If the value is set to True, you can use the Zoom method of the IPageSetup interface and if the value is set to false, you can use the FitToPagesWide and FitToPagesTall method of the IPageSetup interface. For more information on implementation, refer to <a href="#">Configure Paper Scaling</a>.</p>
<p><b>IWorksheet.getHPPageBreaks</b> (<b>'getHPPageBreaks Method' in the on-line documentation</b>)</p>	<p>Used to get the horizontal page breaks that will split rows to multiple pages. However, this method doesn't work when the method setIsPercentScale is set to false. For more information on implementation, refer to <a href="#">Configure Page Breaks</a>.</p>
<p><b>IWorksheet.getVPageBreaks</b> (<b>'getVPageBreaks Method' in the on-line documentation</b>)</p>	<p>Used to get the vertical page breaks that will split columns to multiple pages. However, this method doesn't work when the method setIsPercentScale is set to false. For more information on implementation, refer to <a href="#">Configure Page Breaks</a>.</p>
<p><b>IPageSetup.getOrder</b> (<b>'getOrder Method' in the on-line documentation</b>)</p> <p><b>IPageSetup.setOrder</b> (<b>'setOrder Method' in the on-line documentation</b>)</p>	<p>Used to get or set the page order for each page. For more information on implementation, refer to <a href="#">Configure Page Order</a>.</p>
<p><b>IPageSetup.getBottomMargin</b> (<b>'getBottomMargin Method' in the on-line documentation</b>)</p> <p><b>IPageSetup.setBottomMargin</b> (<b>'setBottomMargin Method' in the on-line documentation</b>)</p> <p><b>IPageSetup.getFooterMargin</b> (<b>'getFooterMargin Method' in the on-line documentation</b>)</p> <p><b>IPageSetup.setFooterMargin</b> (<b>'setFooterMargin Method' in the on-line documentation</b>)</p>	<p>Used to get or set the bottom margins, footer margins, left margins, header margins, right margins and top margins for each page.</p> <p>For more information on implementation of margins in a page, refer to <a href="#">Configure Page Margins</a>.</p>

<p><b>IPageSetup.getLeftMargin</b> ('getLeftMargin Method' in the on-line documentation)</p> <p><b>IPageSetup.setLeftMargin</b> ('setLeftMargin Method' in the on-line documentation)</p> <p><b>IPageSetup.getHeaderMargin</b> ('getHeaderMargin Method' in the on-line documentation)</p> <p><b>IPageSetup.setHeaderMargin</b> ('setHeaderMargin Method' in the on-line documentation)</p> <p><b>IPageSetup.getRightMargin</b> ('getRightMargin Method' in the on-line documentation)</p> <p><b>IPageSetup.setRightMargin</b> ('setRightMargin Method' in the on-line documentation)</p> <p><b>IPageSetup.getTopMargin</b> ('getTopMargin Method' in the on-line documentation)</p> <p><b>IPageSetup.setTopMargin</b> ('setTopMargin Method' in the on-line documentation)</p>	
<p><b>IPageSetup.getDraft</b> ('getDraft Method' in the on-line documentation)</p> <p><b>IPageSetup.setDraft</b> ('setDraft Method' in the on-line documentation)</p>	Used to get or set the drafts that are saved for each page. For more information on implementation, refer to <a href="#">Configure Drafts</a> .
<p><b>IPageSetup.getPrintPageRange</b> ('getPrintPageRange Method' in the on-line documentation)</p> <p><b>IPageSetup.setPrintPageRange</b> ('setPrintPageRange Method' in the on-line documentation)</p>	Used to get or set the print page range while printing a worksheet. For more information on implementation, refer to <a href="#">Configure Print Page Range</a> .
<p><b>IPageSetup.getCenterHorizontally</b> ('getCenterHorizontally Method' in the on-line documentation)</p> <p><b>IPageSetup.setCenterHorizontally</b> ('setCenterHorizontally Method' in the on-line documentation)</p> <p><b>IPageSetup.getCenterVertically</b> ('getCenterVertically Method' in the on-line documentation)</p> <p><b>IPageSetup.setCenterVertically</b></p>	Used to get or set the page center for each page in horizontal and vertical directions. For more information on implementation, refer to <a href="#">Configure Page Center</a> .

('setCenterVertically Method' in the on-line documentation)	
<b>IPageSetup.getFirstPageNumber</b> ('getFirstPageNumber Method' in the on-line documentation)  <b>IPageSetup.setFirstPageNumber</b> ('setFirstPageNumber Method' in the on-line documentation)	Used to get or set the first page number while exporting a worksheet to PDF or while printing a worksheet. For more information on implementation, refer to <a href="#">Configure First Page Number</a> .

For more information on setting pagination, refer to [Configure Print Settings via Page Setup](#).

## Import and Export CSV File

This section summarizes how GcExcel Java handles the spreadsheet documents(.csv files).

While importing and exporting a workbook in order to open and save a csv file or stream, you can use the following methods of the **CsvOpenOptions** ('CsvOpenOptions Class' in the on-line documentation) class and the **CsvSaveOptions** ('CsvSaveOptions Class' in the on-line documentation) class in order to configure several open and save options in a workbook.

Methods	Description
<b>CsvOpenOptions.setConvertNumericData</b> ('setConvertNumericData Method' in the on-line documentation)  <b>CsvOpenOptions.getConvertNumericData</b> ('getConvertNumericData Method' in the on-line documentation)	Used to get or set a value that indicates whether the string in text file is converted to numeric data.
<b>CsvOpenOptions.setConvertDateTimeData</b> ('setConvertDateTimeData Method' in the on-line documentation)  <b>CsvOpenOptions.getConvertDateTimeData</b> ('getConvertDateTimeData Method' in the on-line documentation)	Used to get or set a value that indicates whether the string in text file is converted to date data.
<b>CsvOpenOptions.setSeparatorString</b> ('setSeparatorString Method' in the on-line documentation)  <b>CsvOpenOptions.getSeparatorString</b> ('getSeparatorString Method' in the on-line documentation)	Used to get or set the string value as a separator.
<b>CsvOpenOptions.setEncoding</b> ('setEncoding Method' in the on-line documentation)  <b>CsvOpenOptions.getEncoding</b> ('getEncoding Method' in the on-line documentation)	Used to get or set the default encoding which is UTF-8.
<b>CsvOpenOptions.getParseStyle</b> ('getParseStyle Method' in the on-line documentation)	Used to specify whether the style for parsed values should be applied while converting the string values to

<b>CsvOpenOptions.setParseStyle</b> ('setParseStyle Method' in the on-line documentation)	number or date time.
<b>CsvOpenOptions.setHasFormula</b> ('setHasFormula Method' in the on-line documentation) <b>CsvOpenOptions.getHasFormula</b> ('getHasFormula Method' in the on-line documentation)	Used to specify whether the text is formula if it starts with "=".
<b>CsvSaveOptions.setSeparatorString</b> ('setSeparatorString Method' in the on-line documentation) <b>CsvSaveOptions.getSeparatorString</b> ('getSeparatorString Method' in the on-line documentation)	Used to get or set the string value as the separator. By default, this value is a comma separator.
<b>CsvSaveOptions.setEncoding</b> ('setEncoding Method' in the on-line documentation) <b>CsvSaveOptions.getEncoding</b> ('getEncoding Method' in the on-line documentation)	Used to specify the default encoding which is UTF-8.
<b>CsvSaveOptions.getValueQuoteType</b> ('getValueQuoteType Method' in the on-line documentation) <b>CsvSaveOptions.setValueQuoteType</b> ('setValueQuoteType Method' in the on-line documentation)	Used to get or set how to quote values in the exported text file.
<b>CsvSaveOptions.getTrimLeadingBlankRowAndColumn</b> ('getTrimLeadingBlankRowAndColumn Method' in the on-line documentation) <b>CsvSaveOptions.setTrimLeadingBlankRowAndColumn</b> ('setTrimLeadingBlankRowAndColumn Method' in the on-line documentation)	Used to specify whether the leading blank rows and columns should be trimmed like in Excel.

Refer to the following example code in order to import a .csv file.

Java
<pre>Workbook workbook = new Workbook();  // Opening a CSV file workbook.open("documents\\source.csv", OpenFileFormat.Csv);  // Opening a CSV file using several open options CsvOpenOptions options = new CsvOpenOptions(); options.setSeparatorString("-"); workbook.open("documents\\source.csv", options);</pre>

Refer to the following example code in order to export a .csv file from a workbook or a particular worksheet in the workbook.

Java
------

```
// Saving a CSV file from workbook
Workbook workbook = new Workbook();

// Saving to a CSV file
workbook.save("SaveToCsvFile.csv", SaveFileFormat.Csv);

// Saving to a csv file with advanced settings
CsvSaveOptions options = new CsvSaveOptions();
options.setSeparatorString("-");
workbook.save("SaveToCsvFile.csv", options);
```

## Import and Export CSV Files with Delimiters

GcExcel Java provides support for opening and saving CSV files with custom delimiters for rows, cells and columns. Users can use any custom character of their choice as a delimiter. For instance - Comma (,) , Semicolon (;) , Quotes (" , ') , Braces ( {}, {} ) , pipes ( | ) , slashes ( / \ ) , Carat ( ^ ) , Pipe ( | ) , Tab ( \t ) etc.

Users can import and export the following three types of custom delimiters in CSV files as per their custom requirements and preferences. All these types of delimiters work independently and cannot be combined with each other.

1. **Column Delimiters** - These are the delimiters that separate the columns of a worksheet. By default, a column delimiter is of string type. Users can get or set the column delimiters using the options in the table shared below.

Settings	Description
<b>getColumnSeparator</b> ('getColumnSeparator Method' in the on-line documentation)  <b>setColumnSeparator</b> ('setColumnSeparator Method' in the on-line documentation)	These methods can be used to get or set the column delimiter while opening CSV files.
<b>getColumnSeparator</b> ('getColumnSeparator Method' in the on-line documentation)  <b>setColumnSeparator</b> ('setColumnSeparator Method' in the on-line documentation)	These methods can be used to get or set the column delimiter while saving CSV files.

2. **Row Delimiters** - These are the delimiters that separate the rows of a worksheet. By default, a row delimiter is of string type. Users can get or set the row delimiters using the options in the table shared below.

Settings	Description
<b>getRowSeparator</b> ('getRowSeparator Method' in the on-line documentation)  <b>setRowSeparator</b> ('setRowSeparator	These methods can be used to get or set the column delimiter while opening CSV files.

<b>Method' in the on-line documentation)</b>	
<b>getRowSeparator ('getRowSeparator Method' in the on-line documentation)</b>	These methods can be used to get or set the column delimiter while saving CSV files.
<b>setRowSeparator ('setRowSeparator Method' in the on-line documentation)</b>	

3. **Cell Delimiters** - These are the delimiters that separate the cells of a worksheet. By default, a cell delimiter is of char type. Users can get or set the cell delimiters using the options in the table shared below.

Settings	Description
<b>getCellSeparator ('getCellSeparator Method' in the on-line documentation)</b>	These methods can be used to get or set the column delimiter while opening CSV files.
<b>setCellSeparator ('setCellSeparator Method' in the on-line documentation)</b>	
<b>getCellSeparator ('getCellSeparator Method' in the on-line documentation)</b>	These methods can be used to get or set the column delimiter while saving CSV files.
<b>setCellSeparator ('setCellSeparator Method' in the on-line documentation)</b>	

## Using Code

Refer to the following example code in order to import and export CSV files with delimiters using **CsvOpenOptions ('CsvOpenOptions Class' in the on-line documentation)** class.

Java
<pre>// Initialize workbook Workbook workbook = new Workbook();  // Setting ColumnSeparator, RowSeparator &amp; CellOperator in CsvOpenOptions CsvOpenOptions openOption = new CsvOpenOptions(); openOption.setColumnSeparator(","); openOption.setRowSeparator("\r\n"); openOption.setCellSeparator('"');  // Opening csv in workbook workbook.open("test.csv", openOption);</pre>



```
// Saving workbook to csv
workbook.save("OpenCSVDelimiterRowColumnCell.csv");
```

Refer to the following example code in order to import and export CSV files with delimiters using **CsvSaveOptions** ('**CsvSaveOptions Class**' in the on-line documentation) class.

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);
Object data = new Object[][] {
{ "Name", "City", "Birthday", "Sex", "Weight", "Height" },
{ "Bob", "NewYork", new GregorianCalendar(1968, 6, 8), "male", 80, 180 },
{ "Betty", "NewYork", new GregorianCalendar(1972, 7, 3), "female", 72, 168 },
{ "Gary", "NewYork", new GregorianCalendar(1964, 3, 2), "male", 71, 179 },
{ "Hunk", "Washington", new GregorianCalendar(1972, 8, 8), "male", 80, 171 },
{ "Cherry", "Washington", new GregorianCalendar(1986, 2, 2), "female", 58, 161 },
{ "Eva", "Washington", new GregorianCalendar(1993, 2, 5), "female", 71, 180 } };

// Set data
worksheet.getRange("A1:F5").setValue(data);
worksheet.getRange("A:F").setColumnWidth(20);

// Setting ColumnSeparator/ RowSeparator & CellOperator in CSVSaveOptions
CsvSaveOptions saveOption = new CsvSaveOptions();
saveOption.setColumnSeparator(",");
saveOption.setRowSeparator("\r\n");
saveOption.setCellSeparator(' ');

// Saving workbook to csv
workbook.save("SaveCSVDelimiter.csv", saveOption);
```

## Import and Export JSON Stream

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

This topic includes the following tasks.

- **Import and Export JSON Stream for workbook**
- **Import and Export JSON Stream for worksheet**
- **SpreadJS SSJSON Support**
- **Import and Export SpreadJS JSON Files with Shapes**
- **Retreive Errors while Importing JSON Files**

## Import and Export JSON Stream for workbook

You can export a workbook to a json string/stream using the **toJson** ('**toJson Method** in the on-line documentation) method of the **IWorkbook** ('**IWorkbook Interface** in the on-line documentation) interface. You can also import a json string or stream to your workbook using the **fromJson** ('**fromJson Method** in the on-line documentation) method of the **IWorkbook** interface.

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

Java

```
// Create workbook
Workbook workbook = new Workbook();
Workbook workbook1 = new Workbook();

// Import an excel file
workbook.open("test.xlsx");

// Import or Export from or to a JSON string
OutputStream outputStream = new ByteArrayOutputStream();
workbook.toJson(outputStream);
ByteArrayOutputStream buffer = (ByteArrayOutputStream) outputStream;
byte[] bytes = buffer.toByteArray();
InputStream inputStream = new ByteArrayInputStream(bytes);
workbook1.fromJson(inputStream);

// Export workbook to an excel file
workbook1.save("json_out.xlsx");
```

## Import and Export JSON Stream for worksheet

You can export the information in a worksheet to a json string using the **toJson** method of the **IWorksheet** interface. Similarly, you can also import a json string to your worksheet using the **fromJson** of the **IWorksheet** interface. The worksheet can also be exported or imported to the same or another workbook.

It also enables you to view a large Excel file in SpreadJS. The Excel file can be opened in GcExcel and the json string of a worksheet can be exported using the **toJson** method. Further, the json string of the worksheet can be transferred to client to be loaded in SpreadJS.

### Limitations

- Importing worksheet json to another workbook on server might cause data loss or conflict
- Cell styles used in SpreadJS ssjson are lost in Excel after using Worksheet.toJson()
- SpreadJS doesn't support all the page settings of Excel. Hence, GcExcel does not get all the settings when imported from ssjson.

Refer to the following example code to export and import json string of a worksheet.

C#

```
Workbook workbook = new Workbook();

// ToJson&FromJson can be used in combination with spreadjs
// product:http://spread.grapecity.com/spreadjs/sheets/
```

```
// GrapeCity Documents for Excel import an excel file
String source = "ExcelJsonInput.xlsx";
workbook.open(source);

// Open the file
IWorkbook new_workbook = new Workbook();
new_workbook.open(source);

for (IWorksheet worksheet : workbook.getWorksheets()) {
    worksheet.getRange("A1:C4").setValue(new Object[][] { { "Device", "Quantity", "Unit Price" },
    { "T540p", 12, 9850 }, { "T570", 5, 7460 }, { "Y460", 6, 5400 }, { "Y460F", 8, 6240 } });

// GrapeCity Documents for Excel export a worksheet to json string
String json = worksheet.toJson();

// You can use the json string to initialize spreadjs product
// Product spreadjs will show the excel file contents
// You can use spreadjs product export a json string of worksheet

// GrapeCity Documents for Excel use the json string to update content of the
// corresponding worksheet
new_workbook.getWorksheets().get(worksheet.getName()).fromJson(json);

}
// GrapeCity Documents for Excel export workbook to an excel file
String export = "ExcelJsonOutput.xlsx";
new_workbook.save(export);
```

## SpreadJS SSJSON Support

GcExcel Java provides support for SpreadJS SSJSON. You can import a SSJSON file created with SpreadJS Designer and save it back after modifying it as per your preferences.

### Java

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

// Load SSJSON file
try
{
    FileInputStream stream = new FileInputStream("test.ssjson");
    workbook.fromJson(stream);
}
catch (Exception e)
{
    e.getMessage();
}

// Save file
workbook.save("workbook-ssjson.xlsx");
```

### Import and Export SpreadJS JSON Files with Shapes

GcExcel Java allows users to load and save Grapecity SpreadJS JSON files along with shapes. Besides supporting the import and export of SpreadJS JSON files containing shapes, users can also modify the exported SpreadJS JSON files with shapes and save them back to the original SpreadJS JSON files as and when required.

#### Java

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

// Initialize another workbookWithShape
Workbook workbookWithShape = new Workbook();

// Fetch default worksheet of workbookWithShape
IWorksheet worksheet = workbookWithShape.getWorksheets().get(0);

// Add a shape in worksheet
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Parallelogram, 1, 1, 200, 100);
shape.getLine().setDashStyle(LineDashStyle.Dash);
shape.getLine().setStyle(LineStyle.Single);
shape.getLine().setWeight(2);
IColorFormat color = shape.getFill().getColor();
shape.getLine().setTransparency(0.3);
color.setObjectThemeColor(ThemeColor.Accent6);

// Converting workbook containing shape to JsonString
String jsonString = workbookWithShape.toJson();

// GcExcel can load json string containing shapes
workbook.fromJson(jsonString);

// Saving workbook
workbook.save("4-LoadAndSaveSSJSONContainingShapes.xlsx");
```



**Note:** The following limitations must be kept in mind while exporting Excel files with vertical text to PDF -

- The orientation can only be set to 0, 90, -90 and 255. Other values will be treated as 0 while rendering the PDF file.
- If the font name starts with "@" and the orientation is 255, GcExcel will ignore the "@".

### Retrieve Errors while Importing JSON Files

GcExcel provides the option to get JSON errors, if any, while importing the JSON file using **fromJson** method of **IWorkbook** interface. The error message is displayed by the **getErrorMessage** property of **JsonError** class. Two types of error messages are supported:

- Formula JSON Error - Implemented using the **FormulaJsonError** class and can be raised in case of a formula error in JSON file
- Data Validation JSON Error - Implemented using the **DataValidationJsonError** class and can be raised in case of a data validation error in JSON file

Refer to the below example code which will display a formula JSON error as the JSON file containing formula error is imported in GcExcel.

Java

```
// create a new workbook
Workbook workbook = new Workbook();
// Open JSON file containing JSON errors
InputStream stream = new FileInputStream("ErrorJson.json");

List<JsonError> errors = workbook.fromJson(stream);
for (JsonError item : errors) {
    if (item instanceof FormulaJsonError) {
        FormulaJsonError fError = (FormulaJsonError) item;
        System.out
            .println(fError.getErrorMessage() + " "
                + workbook.getWorksheets().get(fError.getWorksheetName())
                    .getRange(fError.getRow(), fError.getColumn()).toString()
                + " " + fError.getFormula());
    }

    if (item instanceof DataValidationJsonError) {
        DataValidationJsonError dError = (DataValidationJsonError) item;
        System.out.println(dError.getErrorMessage() + " "
            +
            workbook.getWorksheets().get(dError.getWorksheetName()).getRange(dError.getRange().toString())
                + " " + dError.getErrorContent());
    }
}
```

### Limitation

If the data validation in JSON file has error in its formula, Data Validation JSON error will be generated.


## Import and Export Macros

This section summarizes how the import and export of Excel files containing macros is handled in GcExcel Java. Using GcExcel Java, users can load and save Excel files containing macros (.xlsm files) without any issues. Please note that GcExcel Java will not execute these macros.

Typically, this feature allows users to load and save the macro-enabled spreadsheets. Macros help automate repetitive tasks and hence, reduce significant amount of time while working with spreadsheets. Now, users can load such spreadsheets in GcExcel Java directly as Xlsm files, modify them easily and quickly and then save them back.

During the execution of import and export operations on the Excel files, all the macros will also be preserved concurrently along with the data. While opening and saving the Excel workbooks or Excel macro-enabled workbooks, macros will always be imported and exported respectively. The form controls and ActiveX controls are also supported during the import and export operations.

When the **OpenFileFormat** ('**OpenFileFormat Enumeration**' in the on-line documentation) is Xlsm, macros will be imported. When the **SaveFileFormat** ('**SaveFileFormat Enumeration**' in the on-line documentation) is Xlsm, macros will be exported.

 **Note:** While preserving the macros on import or export of Excel files, GcExcel will not execute these macros.


Refer to the following example code in order to import and export macros in spreadsheet documents.

Java

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

// Opening excel file containing macros
workbook.open("testfile.xlsx");

// Saving workbook with macros
workbook.save("5-LoadAndSaveMacros.xlsx");
```

 **Note:** The following limitations must be kept in mind while exporting Excel files with vertical text to PDF -

- The orientation can only be set to 0, 90, -90 and 255. Other values will be treated as 0 while rendering the PDF file.
- If the font name starts with "@" and the orientation is 255, GcExcel will ignore the "@".

## Import and Export OLE Objects

GcExcel Java allows users to preserve OLE objects while opening and saving an Excel file. This feature is extremely useful when users need to deal with import and export of linked objects and embedded objects while working with spreadsheets.

With extensive support for importing and exporting OLE Objects, users can insert linked and embedded objects in their spreadsheets and then preserve these objects while saving the files with .xlsx or .xlsm extension. This feature also facilitates users to use the object linking and embedding (OLE) in order to load and save data from other programs, such as MS Word or MS Excel.

### Example

For instance, let's say you work as a business analyst who wants to visualize information using charts.

You have a source file containing some data. But, you want the chart to be displayed in another file (called a destination file) that picks up data from the source file in order to create charts in the destination file. Now, whenever any changes are done in the data in the source file, obviously you would also want the chart to be updated (or in other words, the destination file to be updated).

That's where the role of supporting the import and export of OLE objects comes into picture. In such a scenario, GcExcel Java will ensure that the original data remains intact in the source file and the destination file represents the updated linked information (updated charts in this example) without impacting the storage of the original data.

Refer to the following example code in order to import and export spreadsheets containing OLE objects.

Java

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

// Opening workbook with OLE object
workbook.open("OleObjectExcelFile.xlsx");

// Saving workbook with OLE object
workbook.save("OleOutExcel.xlsx");
```

## Convert to Image

GcExcel allows you to convert a worksheet, any specified range and various shape types to images. Hence, making it convenient to use the converted images directly in other documents, like Word, PDF or a PPT document. The supported image formats for conversion are PNG, JPG/JPEG, EMF, GIF and WMF.



**Note:** To convert images with transparency, PNG like image format should be used as JPG/JPEG format doesn't support image transparency. In JDK1.6, if a GIF is exported from shape, it will have black background because of the limitation of java platform. For transparent background, the version of jdk must be higher than 1.6

### Convert Worksheet to Image

A worksheet can be converted to image using the **toImage** method of **IWorksheet** interface. The converted image displays the rectangular area of the worksheet enclosed under cell A1 and the last cell where any data or shape is present. For eg, if a worksheet contains a shape or data in the range D5:F9, the converted image will display the area under the range A1:F9.

A blank worksheet cannot be converted to image.

Refer to the following example code to convert a worksheet to image.

#### Java

```
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);
//Add data
worksheet.getRange("S50").setValue(10);
//Save worksheet to image
worksheet.toImage("ConvertWorksheetToImage.png");
```

Refer to the following example code to convert a worksheet to image from existing file.

#### Java

```
// create a png file stream
FileOutputStream outputStream = new FileOutputStream("ConvertWorksheetToImage.png");
// create a new workbook
Workbook workbook = new Workbook();
FileInputStream fileStream = new FileInputStream("Worksheet.xlsx");

// Open a xlsx file contains a group shape
```

```
workbook.open(fileStream);
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Export the shape to image
worksheet.toImage(outputStream, ImageType.PNG);

// close the image stream
outputStream.close();
```

### Convert Range to Image

A specific range in a worksheet can be converted to image using the **toImage** method of the **IRange** interface. The resulting image displays the rectangular area of the worksheet enclosed under the specified range.

Refer to the following example code to convert a specified range to image.

#### Java

```
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

//Add data
worksheet.getRange("D10:F14").setValue(new Object[][] {{ "Device", "Quantity", "Unit Price" }, { "T540p", 12, 9850 },
    { "T570", 5, 7460 },
    { "Y460", 6, 5400 },
    { "Y460F", 8, 6240 }});

IRange range = worksheet.getRange("D10:F14");

//Save range to image
range.toImage("ConvertRangeToImage.png");
```

Refer to the following example code to convert a specified range to image from an existing file.

#### Java

```
// create a png file stream
FileOutputStream outputStream = new FileOutputStream("ConvertRangeToImage.png");
// create a new workbook
Workbook workbook = new Workbook();
FileInputStream fileStream = new FileInputStream("RangeWorkbook.xlsx");

// Open a xlsx file contains a group shape
workbook.open(fileStream);
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Export the shape to image
worksheet.getRange("A1:C5").toImage(outputStream, ImageType.PNG);
// close the image stream
outputStream.close();
```



## Convert Shape to Image

GcExcel allows you to convert various shape types to image using the **toImage** method of the **IShape** interface. The shape types include shapes like chart, picture, slicer and autoshape. The resulting image displays the rectangular area of the worksheet enclosed under the shape.

Refer to the following example code to convert an autoshape to image.

### Java

```
Workbook workbook = new Workbook();
IWorksheet worksheet = workbook.getWorksheets().get(0);

//Add an oval
IShape shape = worksheet.getShapes().addShape(AutoShapeType.Oval, 20, 20, 200, 100);

//Save oval to image
shape.toImage("ConvertShapeToImage.png");
```

Refer to the following example code to convert an autoshape to image from existing file.

### Java

```
// create a png file stream
FileOutputStream outputStream = new FileOutputStream("ConvertShapeToImage.png");
// create a new workbook
Workbook workbook = new Workbook();
FileInputStream fileStream = new FileInputStream("ShapeWorkbook.xlsx");

// Open a xlsx file contains a group shape
workbook.open(fileStream);
IWorksheet worksheet = workbook.getWorksheets().get(0);
// Export the shape to image
worksheet.getShapes().get(0).toImage(outputStream, ImageType.PNG);
// close the image stream
outputStream.close();
```

## Configure JDK 8 Date Time API

With the introduction of the new date and time libraries and robust APIs in Java 8, the legacy date and calendar APIs (`java.util.Date` and `java.util.Calendar` application programming interfaces) have been migrated in order to support JSR310 implementations, handle concurrency issues and ensure thread safety while working with Java applications. The new Date and Time application programming interfaces are standardized by ISO (International Organization for Standardization) keeping in mind the consistency models for important entities like the date, time, duration and period.

GcExcel Java provides extensive support for JDK 8 and hence the configuration of the new JDK 8 Date Time API along with the new libraries, packages (including `java.time`, `java.time.chrono`, `java.time.format`, `java.time.temporal` and `java.time.zone`) and subpackages (`LocalDate`, `LocalTime`, `LocalDateTime`, `Instant`, `Period`, `Duration` etc). All the new classes are extensible, immutable, ensure enhanced thread safety and are self-sufficient to cater to all the Java Date and Time API requirements; thus eliminating the need for any third-party APIs like Joda-Time while working with Java applications. Using GcExcel Java, users can handle all the new date and time APIs while working in Java Integrated Development Environments with Java Development Kit 8 or higher installed on their systems.



**Note:** In order to use JDK 8 Date Time API, users must add the following dependency file to their project-

`gcexcel-extension.x.x.x.jar`

Refer to the following example code in order to configure and use JDK 8 Date and Time APIs in GcExcel Java.

Java

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

// Fetch default worksheet
IWorksheet worksheet = workbook.getWorksheets().get(0);

// Fetch the cell range A1
IRange a1 = worksheet.getRange("A1");

/* Java 8 introduces a new package java.time which contains lots of new
   date/time types and sub-packages to support JSR310.
   GcExcel can handle these new types when working with Java 8 or upper*/

// Setting Cell A1 date time value
a1.setValue(java.time.LocalDateTime.now());

// Get cell's date time value
// LocalDateTime java8Date = (java.time.LocalDateTime)a1.getValue();

// Formatting A1 cell
a1.setNumberFormat("m/d/yyyy h:mm");

// Setting column "A" width
a1.setColumnWidth(30);
```


```
// Saving workbook  
workbook.save("7-SetJDK8DatettimeValue.xlsx");
```

## Release Notes

### Current Release Notes

Refer to the release notes for the major releases of the product.

- [Release Notes for Version 3.2.0](#)
- [Release Notes for Version 3.1.0](#)
- [Release Notes for Version 3.0.0](#)
- [Release Notes for Version 2.2.0](#)
- [Release Notes for Version 2.1.0](#)

 **Note :** GcExcel Java currently does not provide support for Smart Art Graphics, exporting Excel files to XPS and SVG files, importing XLS files and the Form Controls.

## Release Notes for Version 3.2.0

### Enhancements from the Previous Release

The following features has been added with this version of the product:

- Support for generating PDF Form from Excel Templates.
- Support using sparklines and tables in Excel Templates.
- Support defining Fixed layout for Excel report and fill data in specific range.
- Support exporting workbook/worksheet/range to HTML.
- Support Digital Signatures API: add and sign signature lines, add and sign non-visible signatures, verify signatures, etc.
- Pivot Table enhancements: create multiple files from one Pivot Field, Defer updating, Sorting, Field layout settings, etc.
- Support adjustment of shape z-order.
- Support image quality when exporting to PDF.
- Support Picture Transparency when adding Images to Excel.
- Support more SpreadJS features: show or hide horizontal and vertical grid lines, freeze trailing rows/columns, etc.

### Resolved Issues

The following issues have been resolved since the last release.

- Object reference error on converting the Workbook to JSON with DataValidation definitions.
- ToJSON method throws error when cell has formatter on Linux.
- FromJSON method takes long time to import ssjson file using GcExcel.
- Some content displays incompletely in exported PDF.
- Program does not end and CPU utilisation is 100% when exporting to PDF.
- There is messy code in exported image generated by sheet toImage() method on Linux OS.

## Release Notes for Version 3.1.0

## Enhancements from the Previous Release

The following features has been added with this version of the product:

- Support for charts, images and conditional formatting in Templates.
- Support exporting formulas in Templates
- Support global settings in Templates.
- Support converting Excel objects(shape) to image formats.
- Support password protected workbook and worksheet.
- Support adding Error Bars in Chart.
- Support text angle of chart title, axis tick label and data label.
- Support alignment of Shape's TextFrame.
- Add image to specific range.
- Support Gradient Fill Type enum in Shapes.
- Support creating chart/shape/pictures with a custom name.
- Enhanced Background image support for printing to PDF.
- Get pagination info for printing to PDF.
- Support Transparent Cell Background color in PDF.
- Support worksheet JSON I/O.
- Support Outline column to display hierarchical data in saved PDF.
- Support data binding of Range, Table and Worksheet.
- Return errors from JSON Import in GcExcel.

## Resolved Issues

The following issues have been resolved since the last release.

- Filtered data cannot be re-displayed after JSON(made by SJS) I/O in GcExcel.
- Exception occurs on loading specific ssjson.
- Exception may occur on exporting certain Excel sheets with charts to PDF
- Hiding fixed columns and rows causes incorrect display in Excel file.
- Pagination is inconsistent with SpreadJS when the form is exported to PDF.
- JSON file size is bigger when converted using GcExcel JAVA vs Online designer tool.
- The Value property value of the ComboBox cell is lost after JSON I/O.
- NullPointerException may occur on loading certain Excel file and saving it.
- Conditional formatting is lost if the rule references another sheet.

## Release Notes for Version 3.0.0

### Enhancements from the Previous Release

The following features has been added with this version of the product:

- The support for templates have been added to generate Excel reports.
- The support for converting Excel spreadsheets having Slicers to PDF documents have been added to the package.
- The support for New Excel 2016 Chart types have been added to the package.
- The support for Security options while saving to PDF have been added to the package.
- The support for document properties while saving to PDF have been added to the package.
- The API has been enhanced to support Protect Workbook features.

- The support for Chart Sheet option has been added.
- The Support for shape with hyperlink has been added.
- The Support for Group/Ungroup shapes have been added.
- Now, users can calculate Outline Subtotal.
- Now, users can get the Precedents and Dependents of formula cell.
- Now, the Pivot Table's Grand Totals and Report Layout options are similar to MExcel.
- The support for Shape Adjustment has been provided.
- The support for sheet background image to PDF has been provided.
- Now, user can export Excel files with multiple images to PDF with reduced file size.
- The support for License Workbook instance has been added.
- Now, user can rename Pivot fields and Data Fields.
- The support for Cell tags of GrapeCity SpreadJS has been added.
- The support for Cell types of GrapeCity SpreadJS has been added.
- The support for Best fit rows/columns feature of GrapeCity SpreadJS has been added.

## Resolved Issues

The following issues have been resolved since the last release.

- The `NullReferenceException` no more occurs on using `Workbook.ToJson()` and `Workbook.Save()` methods.
- Now, user can set Icon for `IconCriteria`.
- Now, the `_xlfn` prefix is not added before IFNA formula while converting to JSON.
- Fixed the issue where the precision of calculated result was incorrect.
- User can export to PDF with a specified culture
- Cell types are retained after JSON(made by SJS) I/O in GcExcel
- Row/Col Header and every cells are retained after JSON(made by SJS) I/O in GcExcel
- Row/Cols with empty date are retained after JSON(made by SJS) I/O in GcExcel
- No longer messy code while debugging GcExcel code

## Release Notes for Version 2.2.0

### Enhancements from the Previous Release

The following features have been added with this version of the product.

- Excel files with [shapes can be exported to PDF](#).
- [Ranges between different workbooks](#) can be copied.
- Worksheet between different workbooks [can be copied or moved](#).
- [Adjusting page breaks](#) after inserting or deleting rows or columns can be controlled.
- [The row, column or cell delimiter](#) can be customized while loading or saving a CSV file.
- [The tail repeated rows](#) and [right repeated columns](#) can be set when saving to PDF.
- [Paste options are supported](#) during copying and pasting ranges.
- [IRange.Find\(\) and IRange.Replace\(\) methods](#) are supported.
- [Different kinds of pivot table styles](#) can be shown or hidden.
- [Pivot table styles](#) can be exported to PDF.
- [The number format](#) setting for each pivot field is supported.
- [Japanese ruby](#) characters can be preserved after executing the Excel I/O.
- [Users can get and customize each page setting](#) before saving to a PDF file.
- Any sheet range can be [rendered inside a PDF file](#).

- Rows or columns can be [kept together](#) when saving to PDF.
- Multiple workbooks can be [saved to one PDF file](#).
- [Specific pages](#) from spreadsheet can be exported to PDF.
- [Multiple spreadsheet pages](#) can be saved into one PDF page.
- [IRange.AutoFit method](#) to fit rows or columns is supported.
- [IRange.FormulaArrayR1C1](#) method to get or set array formula in R1C1 format is supported.
- [More import flags are supported](#) while opening an Excel file.
- [OLE Objects](#) will be preserved after Excel I/O.
- [Shrink to fit feature for wrapped text](#) is supported while saving to a PDF file.

## Resolved Issues

The following issues have been resolved since the last release.

- GcExcel Java no longer ignores 'ignore\_empty' parameter in the TEXTJOIN formula.
- Fixed the issue of large JSON file generation when using toJson() method in a particular Workbook.
- UsedRange.setValue method now sets proper values to the range when Formula is set to Empty.

## Release Notes for Version 2.1.0

### Enhancements from the Previous Release

The following features have been added with this version of the product.

- The performance of workbook.fromJson() method has been enhanced when the JSON file contains multiple styles.
- Users can now [import and export spreadsheets that contain macros](#). While these will not be executed, the macros will now be preserved when saving.
- The support for [loading and saving GrapeCity SpreadJS JSON files with shapes](#) have been added.
- Users can now [set rich text format in the cells](#) by applying different styles to the textual information entered in the cell.
- While working with custom named styles, users can now [modify an existing style and add it to the Styles collection](#).
- Users can now [export Excel files with vertical text to PDF](#).
- Now, users can [insert any background image to the worksheet](#) including their organization logo, custom watermark or a wallpaper of their choice without any issues.
- PDFBox can now be installed automatically for all versions of Eclipse Maven plugin.
- Extensive [support for the new Date Time API](#) that has been introduced recently with JDK 8 has been provided.
- The pivot table has been enhanced in order to support the date field group in Excel 2016.
- Some overloads have been added for Open and Save methods to avoid passing file format.

## Resolved Issues

The following issues have been resolved since the last release.

- The **Workbook.calculate()** method now evaluates the cell values correctly.
- While saving an Excel file to open XML format, the logical value of the cell is now calculated without any errors.
- Opening the stream returned by **HttpServletRequest.getInputStream()** method no longer throws an exception.
- Saving an Excel file to a PDF file while the used font is null, no longer throws the NullPointerException.
- Loading SSJSON file with null values no longer throws an exception.
- While saving an Excel file to PDF, the merged range in a table now renders appropriately without any issues.

- Loading SSJSON file now renders the hidden rows correctly while saving an Excel file to PDF.



## Index

**Access a Range, 24-25**  
**Access Areas in a Range, 25-26**  
**Access Cells, Rows and Columns in a Range, 26-27**  
**Add and Delete Table Columns and Rows, 236-237**  
**Add Slicer in Pivot Table, 261-263**  
**Add Slicer in Table, 260-261**  
**Add Validations, 95-97**  
**Adjust Column Width and Row Height, 379-380**  
**Area Chart, 203-205**  
**Auto Fit Row Height and Column Width, 40-41**  
**Auto-Filter Table with Slicer, 265-266**  
**Average Rule, 90**  
**Axis and Other Lines, 192-195**  
**Background Image, 160-161**  
**Bar Chart, 205-207**  
**Box Whisker, 221-222**  
**Cell Context, 298-302**  
**Cell Expansion, 298**  
**Cell Types, 51-53**  
**Cell Value Rule, 89**  
**Chart, 171-172**  
**Chart Area, 175-176**  
**Chart Sheet, 229-232**  
**Chart Title, 173-175**  
**Chart Types, 201-203**  
**Charts, 322-327**  
**Color Scale Rule, 90-91**  
**Column Chart, 207-209**  
**Combo Chart, 209-211**  
**Comments, 69-71**  
**Conditional Formatting, 88-89 , 302-303**  
**Configure Chart, 173**  
**Configure Chart Axis, 195-198**  
**Configure Chart Series, 181-187**

**Configure Columns to Repeat at Left and Right, 277-278**

**Configure Fonts and Set Style, 342-343**

**Configure JDK 8 Date Time API, 402-403**

**Configure Page Breaks, 274-276**

**Configure Page Header and Footer, 272-273**

**Configure Page Settings, 273-274**

**Configure Paper Settings, 276-277**

**Configure Print Area, 277**

**Configure Print Page Range, 280-281**

**Configure Rows to Repeat at Top and Bottom, 278-279**

**Configure Sheet Print Settings, 281-282**

**Configure Slicer Layout, 266-267**

**configure-drafts, 279-280**

**Contacting Sales, 17**

**Control Pagination, 350-351**

**Control Position of Overlapping Shapes, 162-163**

**Convert to Image, 399-401**

**Create and Delete Chart, 172-173**

**Create and Delete Tables, 232-233**

**Create and Set Custom Named Style, 168-170**

**Create Excel Report using Tempate, 332-338**

**Create Pivot Table, 240-241**

**Create Row or Column Group, 79-81**

**Create Workbook, 61**

**Custom Functions, 141-147**

**Customize Chart Objects, 176-177**

**Customize Shape Format and Shape Text, 151-155**

**Customize Worksheets, 45-47**

**Cut or Copy Across Sheets, 66**

**Cut or Copy Cell Ranges, 27-31**

**Cut or Copy Shape, Slicer, Chart and Picture, 31-33**

**Cut or Copy Slicer, 267-269**

**Data Bar Rule, 91-92**

**Data Binding, 98-103**

**Data Label, 198-200**

**Data Source Binding, 331-332**

- Data Validations, 94-95
- Date Occurring Rule, 89-90
- Delete Blank Pages From Middle, 357-358
- Delete Validation, 97-98
- Digital Signatures, 103-115
- Duplicate Slicer, 269-270
- Enable or Disable Calculation Engine, 66-67
- End User License Agreement, 17
- Error Bars, 187-191
- Export Borders, 347-349
- Export Conditional Formatting, 349-350
- Export Custom Page Information, 360-361
- Export Different Headers On Different Pages, 358-359
- Export Fills, 367
- Export Last Page Without Headers, 359-360
- Export Multiple Sheets To One Page, 354-356
- Export Picture, 367-368
- Export Pivot Table Styles And Format, 343-346
- Export Shapes, 346-347
- Export Signature Lines, 376
- Export Slicers, 375-376
- Export Sparkline, 368-369
- Export Specific Pages to PDF, 361-362
- Export Table, 369-370
- Export Text, 370-373
- Export to a PDF File, 341-342
- Export to an HTML File, 382-385
- Export Vertical Text, 373-374
- Export Worksheet to PDF, 365-367
- Expression Rule, 94
- Features, 18
- File Operations, 339
- Filter, 77-79
- Find and Replace Data, 33-35
- Fixed Layout, 307-310
- Floor, 198

- Formula Functions, 116-134**
- Formulas, 115-116**
- Freeze Panes in a Worksheet, 43-44**
- Freeze Trailing Panes in a Worksheet, 44-45**
- Funnel, 228-229**
- GcExcel Java Overview, 8**
- Get Row and Column Count, 35**
- Getting Started, 11-12**
- Global Settings, 303-307**
- Group, 79**
- Group or Ungroup Shapes, 157-159**
- Hide Rows and Columns, 35-36**
- Histogram, 222-223**
- Hyperlink on Shape, 155-157**
- Hyperlinks, 71-73**
- Icon Sets Rule, 93-94**
- Image Transparency, 162**
- Import and Export .xlsx Document, 339-341**
- Import and Export CSV File, 389-391**
- Import and Export CSV Files with Delimiters, 391-393**
- Import and Export JSON Stream, 393-397**
- Import and Export Macros, 397-398**
- Import and Export OLE Objects, 398-399**
- Insert And Delete Cell Ranges, 36-37**
- Insert and Delete Rows and Columns, 37-39**
- Keep Rows Together Over Page Breaks, 356-357**
- Key Features, 9-10**
- Legends, 200-201**
- License Information, 14-16**
- Line Chart, 211-213**
- Merge Cells, 39**
- Modify Slicer with Custom Style, 264**
- Modify Table Layout, 238-239**
- Modify Table Layout for Slicer Style, 264-265**
- Modify Table with Custom Style, 238**
- Modify Tables, 233-235**

- Modify Validation, 98**
- Open and Save Workbook, 62-63**
- Outline Column, 84-88**
- Outline Subtotals, 82-84**
- Pareto Chart, 225-226**
- PDF Form Builder, 310-322**
- Pie Chart, 213-214**
- Pivot Table, 239-240**
- Pivot Table Settings, 241-250**
- Pivot Table Style, 250-256**
- Plot Area, 176**
- Precedents and Dependents, 138-141**
- Print Settings, 271-272**
- Protect Workbook, 63-66**
- Quick Start, 12-14**
- Quote Prefix, 53-54**
- Radar Chart, 219-221**
- Range Operations, 24**
- Redistribution, 17**
- Release Notes, 404**
- Release Notes for Version 2.1.0, 407-408**
- Release Notes for Version 2.2.0, 406-407**
- Release Notes for Version 3.0.0, 405-406**
- Release Notes for Version 3.1.0, 404-405**
- Release Notes for Version 3.2.0, 404**
- Remove a Group, 81-82**
- Render Excel Range Inside PDF, 351-354**
- Rich Text, 56-61**
- Save Multiple Workbooks to Single PDF, 362-365**
- Series, 177-181**
- Set Array Formula, 137-138**
- Set Formula to Range, 134-135**
- Set Row Height and Column Width, 40**
- Set Sheet Styling, 165-168**
- Set Table Formula, 135-137**
- Set Values to a Range, 39-40**

- Shape Adjustment, 159-160
- Shapes And Pictures, 147-151
- Shrink To Fit With Text Wrap, 374-375
- Size and Position of Image, 161-162
- Slicer, 259-260
- Slicer Style, 263
- Sort, 73-77
- Sparkline, 256-259
- Sparklines, 329-331
- Specialized Chart, 226
- Statistical Chart, 221
- Stock Chart, 214-216
- Styles, 163-165
- Summary Row, 82
- Sunburst, 226-227
- Support Background Color Transparency, 381-382
- Support Document Properties, 379
- Support Security Options, 376-379
- Support Sheet Background Image, 380-381
- Surface Chart, 216-217
- Table, 232
- Table Filters, 236
- Table Sort, 235-236
- Table Style, 237-238
- Tables, 327-329
- Tags, 54-56
- Technical Support, 16
- Template Configuration, 286-287
- Template Fields, 287-291
- Template Properties, 291-298
- Templates, 283-286
- Theme, 170-171
- Top Bottom Rule, 92
- Treemap, 227-228
- Unique Rule, 93
- Use Do Filter Operation, 270-271

**Walls, 191-192**

**Waterfall Chart, 223-225**

**Work with Used Range, 41-42**

**Work with Worksheets, 19-24**

**Workbook, 61**

**Workbook Views, 67-69**

**Working With Page Setup, 385-389**

**Worksheet, 18-19**

**Worksheet Views, 47-51**

**XY (Scatter) chart, 217-219**